40th International Symposium on Theoretical Aspects of Computer Science

STACS 2023, March 7–9, 2023, Hamburg, Germany

Edited by

Petra Berenbrink

Patricia Bouyer

Anuj Dawar

Mamadou Moustapha Kanté

LIPIcs – Vol. 254 – STACS 2023 www.dagstuhl.de/lipics

Editors

Petra Berenbrink
University of Hamburg, Germany
petra.berenbrink@uni-hamburg.de

Patricia Bouyer
Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, Gif-sur-Yvette, France
bouyer@lsv.fr

Anuj Dawar
University of Cambridge, UK
anuj.dawar@cl.cam.ac.uk

Mamadou Moustapha Kanté
Université Clermont Auvergne, Clermont Auvergne INP, LIMOS, CNRS, Aubière, France
mamadou.kante@uca.fr

ACM Classification 2012
Mathematics of computing → Combinatorics; Mathematics of computing → Graph theory; Theory of computation → Formal languages and automata theory; Theory of computation → Logic; Theory of computation → Design and analysis of algorithms; Theory of computation → Computational complexity and cryptography; Theory of computation → Models of computation

ISBN 978-3-95977-266-2

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/978-3-95977-266-2.

Publication date
March, 2023

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

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.STACS.2023.0
ISBN 978-3-95977-266-2 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

Editorial Board

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The International Symposium on Theoretical Aspects of Computer Science (STACS) conference series is an internationally leading forum for original research on theoretical aspects of computer science.

For the first time, STACS 2023 consists of two tracks, A and B. Track A is dedicated to algorithms and data structures, complexity and games. Track B covers automata, logic, semantics and theory of programming.

STACS is held alternately in France and in Germany. This year's conference, taking place in Hamburg (University of Hamburg) from March 7 to March 9, is the 40th in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), Kiel (2013), Lyon (2014), München (2015), Orléans (2016), Hannover (2017), Caen (2018), Berlin (2019), Montpellier (2020), Saarbrücken (2021, taking place virtually), and Marseille (2022, taking place virtually).

The interest in STACS has remained at a very high level over the past years. The STACS 2023 call for papers led to 183 submissions (148 for Track A and 35 for Track B) with authors from 34 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. For the ninth time within the STACS conference series, there was also a rebuttal period during which authors could submit remarks to the PC concerning the reviews of their papers. In addition, and for the third time, STACS 2023 employed a lightweight double-blind reviewing process: submissions should not reveal the identity of the authors in any way. However, it was still possible for authors to disseminate their ideas or draft versions of their paper as they normally would, for instance by posting drafts on the web or giving talks on their results. The committee selected 52 papers during for publication (41 for Track A and 11 for Track B). This means an acceptance rate around 28%. As co-chairs of the program committee, we would like to sincerely thank all its members and the 285 external reviewers for their valuable work. In particular, there were intense and interesting discussions inside the PC committee. The very high quality of the submissions made the selection an extremely difficult task.

We would like to express our thanks to the three invited speakers: Karoliina Lethinen (CNRS, Aix-Marseille Université, France), Eva Rotenberg (Technical University of Denmark, Denmark), and Moshe Y. Vardi (Rice University, USA).

We thank Michael Wagner from the LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series. We would also like to thank the local organising team at University of Hamburg for all their help with the web pages, registration and preparing the program. Our special thank to Felix Biermeier, Christohper Hahn, Lukas Hinze and Hamed Hosseinpour.

Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Conference Organisation

Program Committee

Track A

Isolde Adler	University of Bamberg, Germany
Petra Berenbrink	University of Hamburg, Germany, co-chair
Mathilde Bouvel	Loria, France
Maike Buchin	University of Bochum, Germany
Jean Cardinal	Université libre de Bruxelles, Belgium
Amin Coja-Oghlan	Dortmund University, Germany
Franziska Eberle	London School of Economics and Political Science, UK
Omar Fawzi	INRIA Lyon, France
George Giakkoupis	INRIA Rennes, France
Danny Hermelin	Ben-Gurion University, Israel
Mamadou Moustapha Kanté	Université Clermont Auvergne, France, co-chair
O-joung Kwon	Hanyang University, South Korea
Michael Lampis	Université Paris Dauphine, France
Andrea Marino	University of Florence, Italy
Matthias Mnich	Hamburg University of Technology, Germany
Arne Meier	University of Hannover, Germany
Nabil Mustafa	Université Sorbonne Paris Nord, France
Joseph Seffi Naor	Technion, Israel
Daniel Paulusma	University of Durham, UK
Kirk Pruhs	University of Pittsburgh, USA
Daniel Schmand	University of Bremen, Germany
Takeharu Shiraga	Chuo University, Japan
Maya Stein	Universidad Chile, Chile
Sébastien Tavenas	Université Savoie Mont Blanc, France
Victor Zamaraev	University of Liverpool, UK

Track B

Luca Aceto	Reykjavik University, Iceland and Gran Sasso Science Institute, Italy
S Akshay	IIT Bombay, India
Achim Blumensath	Masaryk University, Czech Republic
Patricia Bouyer	CNRS, France, co-chair
Anuj Dawar	University of Cambridge, UK, co-chair
Silvia Ghilezan	University of Novi-Sad and Mathematical Institute SASA, Serbia
Stefan Göller	University of Kassel, Germany
Emmanuel Jeandel	University of Lorraine, France
Cynthia Kop	Radboud University Nijmegen, Netherlands
Daniel Neider	Universität Oldenburg, Germany
Marie Van Den Bogaard	Université Gustave Eiffel, France
James Worrell	University of Oxford, UK

Steering Committee

Dietmar Berwanger	LMF, CNRS, Université Paris-Saclay
Arnaud Durand	IMJ, Univ. Paris 7
Cyril Nicaud	LIGM, Université Paris-Est
Natacha Portier	LIP, ENS Lyon
Sylvain Schmitz	IRIF, Université de Paris
Ioan Todinca	LIFO, Université d'Orléans (co-chair)
Olaf Beyersdorff	Jena
Henning Fernau	Trier
Arne Meier	Hannover
Rolf Niedermeier	Berlin
Heiko Röglin	Bonn
Thomas Schwentick	Dortmund (co-chair)

Local Organising Committee (University of Hamburg)

Petra Berenbrink
Felix Biermeier
Christopher Hahn
Lukas Hinze
Hamed Hosseinpour
Katrin Koester

Subreviewers

285 external subreviewers assisted the PC. We apologize to any subreviewers who do not appear in this list (because their reviews were entered manually into Easychair).

Andreas Abels
Antonis Achilleos
Duncan Adamson
Akanksha Agrawal
Jungho Ahn
Antoine Amarilli
Elli Anastasiadi
Haris Angelidakis
Simon Apers
Guillaume Aubrun
Nikhil Balaji
Max Bannach
Jérémy Barbay
Timon Barlag
Veronica Becher
Nicolas Bedaride
Rémy Belmonte
Pierre Bergé
Benjamin Bergougnoux
Felix Biermeier
Hans L. Bodlaender
Mikołaj Bojańczyk
Benjamin Bordais
Adam Bouland
Nicolas Bousquet
Léonard Brice
Srecko Brlek
Kevin Buchin
Andrei Bulatov
Martin Böhm
Silvio Capobianco
Olivier Carton
Antonio Casares
Katrin Casel
Sankardeep Chakraborty
Aggeliki Chalki
Dmitry Chistikov
Anders Claesson
Thomas Colcombet
Alessio Conte
Denis Cornaz
Radu Curticapean

Justin Dallant
Peter Davies
Guilherme de Castro Mendes Gomes
Paloma de Lima
Fabien De Montgolfier
Elie De Panafieu
Argyrios Deligkas
Holger Dell
Max Deppert
Carola Doerr
Igor Dolinka
Riccardo Dondi
Vladimir Dragovic
Ran Duan
Guillaume Ducoffe
Fabien Dufoulon
Lubomira Dvorakova
Martin Dyer
Kord Eickmeyer
Khaled Elbassioni
Yury Elkin
Bruno Escoffier
Guy Even
Léo Exibard
Hamza Fawzi
Henning Fernau
Thomas Fernique
Guillaume Fertin
Johannes K. Fichte
Nathanaël Fijalkow
Jean-Christophe Filliatre
David Fischer
Pamela Fleischmann
Pierre Fraigniaud
Nora Frankl
Kaito Fujii
Esther Galby
Guilhem Gamard
Moses Ganardi
Sabrina Alexandra Gaube
Pawel Gawrychowski
Colin Geniet
Loukas Georgiadis
Konstantinos Georgiou
Archontia Giannopoulou
Julian Golak
Petr Golovach
Alexander Golovnev
R Govind
Roland Grappe
Martin Grohe
Anselm Haak
Christoph Haase
Christopher Hahn
Maximilian Hahn-Klimroth
Emmanuel Hainry
Thekla Hamm
Yassine Hamoudi
Tesshu Hanaka
Kristoffer Arnsfelt Hansen
Markus Hecher
Klaus Heeger
Lukas Hintze
Kiya Hironori
Juho Hirvonen
Felix Hommelsheim
Hamed Hosseinpour
John Iacono
Sharat Ibrahimpur
Sandy Irani
Takehiro Ito
Yuval Itzhaki
Riko Jacob
Lars Jaffke
Klaus Jansen
Jiaqing Jiang
Jie-Hong Roland Jiang
Vincent Jugé
Paul Jungeblut
Sven Jäger
Dominik Kaaser
Matthias Kaul
Edon Kelmendi
George Kenison
Thomas Kesselheim
Eun Jung Kim
Sándor Kisfaludi-Bak
Peter Kling
Dušan Knop
Yusuke Kobayashi
Florent Koechlin
Pascal Koiran
Balagopal Komarath
Michael Kompatscher
Lukasz Kowalik
Lena Krieg
Dominik Krupke
Mrinal Kumar
Kazuhiro Kurita
Noleen Köhler
Dominik Köppl
Manuel Lafond
Joseph Landsberg
Julien Leroy
Roie Levin
Nutan Limaye
Chih-Hung Liu
Raul Lopes
Dimitrios Los
Jack H. Lutz

Ramanujan M. Sridharan
Guillaume Malod
Florin Manea
Isja Mannens
Alessio Mansutti
Pasin Manurangsi
Victor Marsault
Barnaby Martin
Richard Mayr
Andrew McGregor
Antoine Meyer
Takuya Mieno
Mirjana Mikalački
Melanija Mitrović
Hendrik Molter
Kaushik Mondal
Pedro Montealegre
Nils Morawietz
Shay Mozes
Haiko Muller
Tobias Mömke
Giorgi Nadiradze
Jelani Nelson
Daniel Neuen
Eike Neumann
Reino Niskanen
Pierre Ohlmann
Dan Olteanu
Giacomo Ortali
Yota Otachi
Aadil Oufkir
Sang-il Oum
Tommaso Padoan
Arnau Padrol
Nikos Parotsidis
Francesco Pasquale
Christophe Paul
Daria Pchelina
Tomáš Peitl
Vincent Penelle
Simon Perdrix
Anthony Perez
Giulio Ermanno Pibiri
Marta Piecyk
G. Michele Pinna
Elias Pitschmann
Pierre Popoli
Aleksandr Popov
Lionel Pournin
M. Praveen
Nicola Prezza
Romain Péchoux
Jakub Radoszewski
Vijayaragunathan Ramamoorthi
Rajiv Raman
Raghavendra Rao B V
Jean-Francois Raskin
Ran Raz
Guus Regts
Eric Remila
Jérémie Roland
Maurice Rolvien
Mirko Rossi
Ignaz Rutter
Andrew Ryzhikov
Harald Räcke
Chandan Saha
Subhayan Saha
Mohammad Salavatipour
Saket Saurabh
Olga Scheftelowitsch
Kevin Schewior
Martin Schirneck
Jens Schlöter
Christiane Schmidt
Melanie Schmidt
Sylvain Schmitz
Moshe Schwartz
Torben Schürenberg
Thomas Seiller
Pavel Semukhin
Suhail Sherif
Masahiro Shibata
Mahsa Shirmohammadi
Mark Siggers
Florian Sikora
Ana Silva
Francesco Silvestri
Bertrand Simon
Kirill Simonov
George Skretas
Friedrich Slivovsky
Patrick Sole
Frank Staals
Tobias Stamm
Georgios Stamoulis
Benjamin Steinberg
Milos Stojakovic
Yann Strozecki
Donald Stull
Prafullkumar Tale
Jakub Tarnawski
Véronique Terrier
Sharma V. Thankachan
Suhas Thejaswi
Guillaume Theyssier
Ioan Todinca
Patrick Totzke
Vera Traub
Konstantinos Tsakalidis

Ivor van der Hoog
Prashant Vasudevan
Adrian Vetta
Cosimo Vinci
Dejan Vukobratovic
Koichi Wada
Magnus Wahlström
Chunhao Wang
Jianxin Wang
Philipp Warode
Armin Weiss
Klaus Wich
Arne Winterhof
Philipp Woelfel
Petra Wolf
Xiang-Gen Xia
Georg Zetzsche
Ruizhe Zhang
Yiming Zhao
Samson Zhou
Philipp Zschoche
Jeroen Zuiddam
Jakub Łącki

List of Authors

Erhard Aichinger (4)
Institute for Algebra, Johannes Kepler Universität Linz, Austria

Maxim Akhmedov (5)
Department of Mathematical Logic and Algorithms, Moscow State University, Russia

Cezar-Mihail Alexandru (6)
Department of Computer Science, University of Bristol, UK

Georgios Amanatidis (7)
University of Essex, Colchester, UK

Antoine Amarilli (8)
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Maxim Babenko (5)
Higher School of Economics, Moscow, Russia

Pascal Baumann (9)
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Johannes Bausch (54)
University of Cambridge, UK

Benjamin Bergougnoux (11)
University of Warsaw, Poland

Pierre Bergé (10)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Markus Bläser (12)
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany

Thomas Bläsius (13)
Karlsruhe Institute of Technology, Germany

Narek Bojikian (14)
Humboldt-Universität zu Berlin, Germany

Édouard Bonnet (10, 15)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Nader H. Bshouty (16)
Department of Computer Science, Technion, Haifa, Israel

Massimo Cairo (17)
Department of Computer Science, University of Helsinki, Finland

Florent Capelli (18)
Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 – CRIStAL, F-59000 Lille, France

Arnaud Carayol (19)
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Dror Chawin (20)
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Sravanthi Chede (21)
Indian Institute of Technology Ropar, Rupnagar, India

Vera Chekan (14)
Humboldt-Universität zu Berlin, Germany

Suryajith Chillara (22)
International Institute of Information Technology, Hyderabad, India

Marek Chrobak (23)
University of California at Riverside, CA, USA

Lorenzo Clemente (24)
University of Warsaw, Poland

Ruiwen Dong (25, 26)
Department of Computer Science, University of Oxford, UK

Maria Donten-Bury (24)
University of Warsaw, Poland

Philippe Duchon (19)
Univ. Bordeaux, CNRS UMR 5800, LaBRI, F-33400 Talence, France

Marc Dufay (27)
École Polytechnique and IRIF, Palaiseau, France

Pavel Dvořák (6)
Tata Institute of Fundamental Research, Mumbai, India; Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Zdeněk Dvořák (28)
Computer Science Institute, Charles University, Prague, Czech Republic

Hugues Déprés (10)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Nicolas El Maalouly (29)
Department of Computer Science,
ETH Zürich, Switzerland

Jessica Enright (30)
School of Computing Science,
University of Glasgow, UK

Emmanuel Filiot (32)
Université libre de Bruxelles, Belgium

Tobias Friedrich (13)
Hasso Plattner Institute,
University of Potsdam, Germany

Sevag Gharibian (54)
Universität Paderborn, Germany

Ugo Giocanti (15)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Valentin Gledel (34)
Department of Mathematics and Mathematical Statistics, Umeå University, Sweden

Kasper Green Larsen (31)
Aarhus University, Denmark

Coral Grichener (22)
Google Research, Herzliya, Israel

Simon Grünbacher (4)
Institute for Algebra, Johannes Kepler Universität Linz, Austria

Leszek Gąsieniec (33)
University of Liverpool, UK

Samuel Haney (23)
Tumult Labs, Durham, NC, USA

Ishay Haviv (20)
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Klaus Heeger (35)
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Falko Hegerfeld (14)
Humboldt-Universität zu Berlin, Germany

Monika Henzinger (36)
Institute of Science and Technology Austria, Klosterneuburg, Austria

Shengyu Huang (37)
Department of Computer Science, EPFL, Lausanne, Switzerland

Reijo Jaakkola (38)
Tampere University, Finland

Ismaël Jecker (32)
University of Warsaw, Poland

Haim Kaplan (53)
Tel Aviv University, Israel

Maximilian Katzmann (13)
Karlsruhe Institute of Technology, Germany

Shahbaz Khan (17)
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

Pieter Kleer (7)
Tilburg University, The Netherlands

Tomohiro Koana (39)
Faculty IV, Institute of Software Engineering and Theoretical Computer Science, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Florent Koechlin (19)
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Balagopal Komarath (40)
IIT Gandhinagar, India

Christian Konrad (6, 41)
Department of Computer Science, University of Bristol, UK

Tuukka Korhonen (11)
University of Bergen, Norway

Stefan Kratsch (14)
Humboldt-Universität zu Berlin, Germany

Anant Kumar (40)
IIT Gandhinagar, India

Antti Kuusisto (38)
Tampere University, Finland;
University of Helsinki, Finland

François Le Gall (42)
Graduate School of Mathematics, Nagoya University, Japan

Karoliina Lehtinen (1)
CNRS, Aix-Marseille University, LIS, Marseille, France

Haohong Li (44)
Department of Computer Science,
Lafayette College, Easton, PA, USA

Mehraneh Liaee (23)
Northeastern University, Boston, MA, USA

Chih-Hung Liu (37)
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

Dimitrios Los (45)
Department of Computer Science & Technology, University of Cambridge, UK

Benjamin Lévêque (43)
Laboratoire G-SCOP, Grenoble INP, Université Grenoble-Alpes, France

Christof Löding (32)
RWTH Aachen University, Germany

Konrad Majewski (46)
Institute of Informatics, University of Warsaw, Poland

Claire Mathieu (27)
CNRS Paris, France

Hendrik Mayer (12)
Massachusetts Institute of Technology, Cambridge, MA, USA

Filip Mazowiecki (24)
University of Warsaw, Poland

Kitty Meeks (30)
School of Computing Science, University of Glasgow, UK

Roland Meyer (9)
TU Braunschweig, Germany

Suchismita Mishra (40)
Universidad Andrés Bello, Santiago, Chile

Masayuki Miyamoto (42)
Graduate School of Mathematics, Nagoya University, Japan

Augusto Modanese (47)
Aalto University, Espoo, Finland

Hendrik Molter (30)
Department of Computer Science and Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Mikaël Monet (8)
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Mikael Møller Høgsgaard (31)
Aarhus University, Denmark

Moritz Mühlenthaler (43)
Laboratoire G-SCOP, Grenoble INP, Université Grenoble-Alpes, France

Kheeran K. Naidu (6, 41)
Department of Computer Science, University of Bristol, UK

Satyadev Nandakumar (48)
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Jesper Nederlof (11)
Utrecht University, The Netherlands

Stefan Neumann (36)
KTH Royal Institute of Technology, Stockholm, Sweden

Cyril Nicaud (19)
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

André Nichterlein (35)
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Rolf Niedermeier (35)
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Harumichi Nishimura (42)
Graduate School of Informatics, Nagoya University, Japan

Ora Nova Fandina (31)
Aarhus University, Denmark

Naoto Ohsaka (49)
CyberAgent, Inc., Tokyo, Japan

Nacim Oijid (34)
Univ. Lyon, Université Lyon 1, LIRIS UMR CNRS 5205, F-69621, Lyon, France

Jędrzej Olkowski (50)
Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland

Patrice Ossona de Mendez (15)
Centre d'Analyse et de Mathématique Sociales CNRS UMR 8557, Paris, France; Computer Science Institute, Charles University (IUUK), Prague, Czech Republic

Debmalya Panigrahi (23)
Duke University, Durham, NC, USA

Charles Paperman (51)
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Michał Pilipczuk (24, 46, 50)
University of Warsaw, Poland

Subin Pulari (48)
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Rajmohan Rajaraman (23)
Northeastern University, Boston, MA, USA

Gwenaël Richomme (52)
LIRMM, Université Paul-Valéry Montpellier 3, Université de Montpellier, CNRS, Montpellier, France

Romeo Rizzi (17)
Department of Computer Science, University of Verona, Italy

Matthieu Rosenfeld (52)
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Eva Rotenberg (2)
Technical University of Denmark, Lyngby, Denmark

Daniel Rutschmann (37)
Department of Applied Mathematics and Computer Science, Technical University of Denmark, Copenhagen, Denmark

Mateusz Rychlicki (50)
School of Computing, University of Leeds, UK

Harald Räcke (36)
TU München, Germany

Yaniv Sadeh (53)
Tel Aviv University, Israel

Sylvain Salvati (51)
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Thomas Sauerwald (45)
Department of Computer Science & Technology, University of Cambridge, UK

Stefan Schmid (36)
TU Berlin, Germany;
Fraunhofer SIT, Darmstadt, Germany

Sebastian Schmidt (17)
Department of Computer Science, University of Helsinki, Finland

Aditi Sethia (40)
IIT Gandhinagar, India

Amir Shpilka (22)
Tel Aviv University, Israel

Devansh Shringi (12)
University of Toronto, Canada

Anil Shukla (21)
Indian Institute of Technology Ropar, Rupnagar, India

Marek Sokołowski (46)
Institute of Informatics, University of Warsaw, Poland

Claire Soyez-Martin (51)
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Paul G. Spirakis (33)
University of Liverpool, UK

Grzegorz Stachowiak (33)
University of Wrocław, Poland

Daniel Stephan (13)
GSV Algorithm Consulting UG (haftungsbeschränkt) & Co. KG, Potsdam, Germany

Arun Steward (41)
Department of Computer Science, University of Bristol, UK

Yann Strozecki (18)
Université Paris Saclay, UVSQ, DAVID, France

Ravi Sundaram (23)
Northeastern University, Boston, MA, USA

Thomas Suzan (43)
Laboratoire G-SCOP, Grenoble INP, Université Grenoble-Alpes, France

Stéphan Thomassé (15)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Alexandru I. Tomescu (17)
Department of Computer Science, University of Helsinki, Finland

Moshe Y. Vardi (3)
Rice University, Houston, TX, USA

Miikka Vilander (38)
Tampere University, Finland

Rémi Watrigant (10)
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

James D. Watson (54)
University College London, UK

Sarah Winter (32)
Université libre de Bruxelles, Belgium

Karol Węgrzycki (50)
Saarland University, Saarbrücken, Germany;
Max Planck Institute for Informatics,
Saarbrücken, Germany

Ge Xia (44)
Department of Computer Science,
Lafayette College, Easton, PA, USA

Neal E. Young (23)
University of California at Riverside, CA, USA

Georg Zetzsche (9)
Max Planck Institute for Software Systems
(MPI-SWS), Kaiserslautern, Germany

Hang Zhou (27)
École Polytechnique, Palaiseau, France

Elia C. Zirondelli (17)
Department of Mathematics,
University of Trento, Italy

Philipp Zschoche (55)
Faculty IV, Algorithmics and Computational
Complexity, Technische Universität Berlin,
Germany

Anna Zych-Pawlewicz (50)
Institute of Informatics,
University of Warsaw, Poland

A Brief History of History-Determinism

Karoliina Lehtinen
CNRS, Aix-Marseille University, LIS, Marseille, France

Abstract

Most nondeterministic automata models are more expressive, or at least more succinct, than their deterministic counterparts; however, this comes at a cost, as deterministic automata tend to have better algorithmic properties. History-deterministic automata are an intermediate model that allows a restricted form of nondeterminism: all nondeterministic choices must be resolvable on-the-fly, with only the knowledge of the word prefix read so far – as opposed to general nondeterminism, which allows for guessing the future of the word. History-deterministic automata combine some of the algorithmic benefits of determinism with some of the increased power of nondeterminism, thus enjoying (some of) the best of both worlds.

History-determinism, as it is understood today, has its roots in several independently invented notions: Kupferman, Safra and Vardi's automata recognising tree languages derived from word languages [11] (a notion that has been later referred to as automata that are *good-for-trees* [1]), Henzinger and Piterman's *good-for-games* automata [9], and Colcombet's history-deterministic automata, introduced in his work on regular cost-automata [6]. In the ω-regular setting, where they were initially most studied, the notions of good-for-trees, good-for-games and history-determinism are equivalent, despite differences in their definitions. The key algorithmic appeal of these automata is that like deterministic automata, they have good compositional properties. This makes them particularly useful for applications such as reactive synthesis, where composition of games and automata is at the heart of effective solutions.

Since then, history-determinism has received its fair share of attention, not least because of its relevance to synthesis. Indeed it turns out to be a natural and useful form of nondeterminism more broadly, and can be generalised to all sorts of different automata models: alternating automata [2], pushdown automata [12, 8], timed automata [10, 5], Parikh automata [7], and quantiative automata [3], to name a few. In each of these models, history-determinism offers some trade-offs between the power of nondeterminism and the algorithmic properties of determinism. In particular, depending on the model, they can be either more expressive or more succinct than their deterministic counterparts, while retaining better algorithmic properties – in particular with respect to deciding universality, language inclusion and games – than fully nondeterministic automata.

The drive to extend history-determinism to more powerful automata models has also lead to a better understanding of the properties of these automata, of how they compare to related notions (such as good-for-games automata and determinisability by pruning), and of the various games and tools used to study them.

This talk aims to give a broad introduction to the notion of history determinism as well as an overview of some of the recent developements on the topic. It will also highlight some of the many problems that remain open. It is loosely based on a recent survey, written jointly with Udi Boker, which gives an informal presentation of what are, in our view, the key aspects of history-determinism [4].

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

Keywords and phrases History-determinism, nondeterminism, automata, good-for-games

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.1

Category Invited Talk

Related Version *Full Version*: https://dl.acm.org/doi/10.1145/3584676.3584682 [4]

© Karoliina Lehtinen;
licensed under Creative Commons License CC-BY 4.0

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 1; pp. 1:1–1:2

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

References

1 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the presence of a diverse or unknown future. In *Proceedings of ICALP*, pages 89–100, 2013. doi:10.1007/978-3-642-39212-2_11.

2 Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *Proceedings of CONCUR*, pages 19:1–19:16, 2019. doi:10.4230/LIPIcs.CONCUR.2019.19.

3 Udi Boker and Karoliina Lehtinen. History determinism vs. good for gameness in quantitative automata. In *Proc. of FSTTCS*, pages 35:1–35:20, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.38.

4 Udi Boker and Karoliina Lehtinen. When a little nondeterminism goes a long way: An introduction to history-determinism. *ACM SIGLOG News*, 10(1):24–51, Feb 2023. doi:10.1145/3584676.3584682.

5 Sougata Bose, Thomas A. Henzinger, Karoliina Lehtinen, Sven Schewe, and Patrick Totzke. History-deterministic timed automata are not determinizable. In *Proceedings of RP*, pages 67–76, 2022. doi:10.1007/978-3-031-19135-0_5.

6 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proceedings of ICALP*, pages 139–150, 2009. doi:10.1007/978-3-642-02930-1_12.

7 Enzo Erlich, Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. History-deterministic parikh automata. *CoRR*, 2022. doi:10.48550/arXiv.2209.07745.

8 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A bit of nondeterminism makes pushdown automata expressive and succinct, 2022. doi:10.48550/arXiv.2105.02611.

9 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proceedings of CSL*, pages 395–410, 2006. doi:10.1007/11874683_26.

10 Thomas A. Henzinger, Karoliina Lehtinen, and Patrick Totzke. History-deterministic timed automata. In *Proceedings of CONCUR*, pages 14:1–14:21, 2022. doi:10.4230/LIPIcs.CONCUR.2022.14.

11 Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006. Conference version in 1996. doi:10.1016/j.apal.2005.06.009.

12 Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω-pushdown automata. *Log. Methods Comput. Sci.*, 18(1), 2022. Conference version at LICS 2020. doi:10.46298/lmcs-18(1:3)2022.

Amortised Analysis of Dynamic Data Structures

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Abstract

In dynamic data structures, one is interested in efficiently facilitating queries to a data set, while being able to efficiently perform updates as the data set undergoes changes. Often, relaxing the efficiency measure to the amortised setting allows for simpler algorithms. A well-known example of a data structure with amortised guarantees is the splay tree by Sleator and Tarjan [14].

Similarly, in data structures for dynamic graphs, one is interested in efficiently maintaining some information about the graph, or facilitating queries, as the graph undergoes changes in the form of insertion and deletion of edges. Examples of such information include connectivity, planarity, and approximate sparsity of the graph: is the graph presently connected? Is it planar? Has its arboricity grossly exceeded some specified number $\widetilde{\alpha}$? The related queries could be: is a connected to b? Are the edges uv and uw consecutive in the ordering around u in its current planar embedding? Or, report the $O(\alpha)$ out-edges of vertex x.

In this talk, we will see Brodal and Fagerberg's amortised algorithm for orienting sparse graphs (i.e. of arboricity $\leq \alpha$), so that each vertex has $O(\alpha)$ out-edges [2]. The algorithm itself is extremely simple, and uses an elegant amortised argument in its analysis. Then, we will visit the problem of dynamic planarity testing: is the graph presently planar? Here, we will see an elegant amortised reduction to the seemingly easier problem, where planarity-violating edges may be detected and rejected [5]. We will see a sketch of how the current state-of-the-art algorithm for efficient planarity testing [8] uses ideas similar to those in [2] to analyse the behaviour of a greedy algorithm via a possibly inefficient algorithm with provably low recourse [9]. If time permits, we will touch upon a recent simple amortised data structure for maintaining information in dynamic forests [10], which builds on ideas from splay trees.

The talk concludes with some open questions in the area.

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

Keywords and phrases Amortised analysis, splaying, dynamic graphs, planarity testing

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.2

Category Invited Talk

Funding *Eva Rotenberg*: Partially supported by the Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis", the VILLUM Foundation grant 37507 "Efficient Recomputations for Changeful Problems", and the Carlsberg Young Researcher Award CF21-0302 "Graph Algorithms with Geometric Applications".

References

1. Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Alg.*, 1(2):243–264, 2005.
2. Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representation of sparse graphs. In *WADS '99*, volume 1663 of *Lecture Notes in Computer Science*, pages 342–351. Springer, 1999. doi:10.1007/3-540-48447-7_34.
3. Richard Cole. On the dynamic finger conjecture for splay trees. part II: the proof. *SIAM J. Comput.*, 30(1):44–85, 2000. doi:10.1137/S009753979732699X.
4. Richard Cole, Bud Mishra, Jeanette P. Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. part I: splay sorting log n-block sequences. *SIAM J. Comput.*, 30(1):1–43, 2000. doi:10.1137/S0097539797326988.

© Eva Rotenberg;
licensed under Creative Commons License CC-BY 4.0

5 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification: I. planarity testing and minimum spanning trees. *Journal of Computer and Systems Sciences*, 52(1):3–27, February 1996. doi:10.1006/jcss.1996.0002.

6 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing with applications. *Journal of the ACM*, 46(1):28–91, 1999. doi:10.1145/300515.300517.

7 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, 2001. doi:10.1145/502090.502095.

8 Jacob Holm and Eva Rotenberg. Fully-Dynamic Planarity Testing in Polylogarithmic Time. In *STOC 2020*, pages 167–180. ACM, 2020. doi:10.1145/3357713.3384249.

9 Jacob Holm and Eva Rotenberg. Worst-case polylog incremental spqr-trees: Embeddings, planarity, and triconnectivity. In *SODA 2020*, pages 2378–2397. SIAM, 2020. doi:10.1137/1.9781611975994.146.

10 Jacob Holm, Eva Rotenberg, and Alice Ryhl. Splay Top Trees. In *SOSA '23*. SIAM, 2023.

11 John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.

12 Johannes A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In *STOC '94*, pages 706–715, 1994. doi:10.1145/195058.195439.

13 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

14 Daniel D. Sleator and Robert E. Tarjan. Self-Adjusting Binary Search Trees. *J. ACM*, 32(3):652–686, 1985. doi:10.1145/3828.3835.

Logical Algorithmics: From Theory to Practice

Moshe Y. Vardi
Rice University, Houston, TX, USA

Abstract

The standard approach to algorithm development is to focus on a specific problem and develop for it a specific algorithm. Codd's introduction of the relational model in 1970 included two fundamental ideas: (1) Relations provide a universal data representation formalism, and (2) Relational databases can be queried using first-order logic. Realizing these ideas required the development of a meta-algorithm, which takes a declarative query and executes it with respect to a database. In this talk, I will describe this approach, which I call Logical Algorithmics, in detail, and explore its profound ramification.

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

Keywords and phrases Logic, Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.3

Category Invited Talk

Funding Work supported in part by NSF grants IIS-1527668, CCF-1704883, IIS-1830549, CNS-2016656, DoD MURI grant N00014-20-1-2787, and an award from the Maryland Procurement Office.

References

1 Jeffrey M. Dudek, Vu Phan, and Moshe Y. Vardi. ADDMC: weighted model counting with algebraic decision diagrams. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1468–1476. AAAI Press, 2020. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5505.

2 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000. doi:10.1006/jcss.2000.1713.

3 Guoqiang Pan and Moshe Y. Vardi. Symbolic techniques in satisfiability solving. *J. Autom. Reason.*, 35(1-3):25–50, 2005. doi:10.1007/s10817-005-9009-7.

4 Vu H. N. Phan and Moshe Y. Vardi. DPO: dynamic-programming optimization on hybrid constraints. *CoRR*, abs/2205.08632, 2022. doi:10.48550/arXiv.2205.08632.

5 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982. doi:10.1145/800070.802186.

6 Moshe Y. Vardi. On the complexity of bounded-variable queries. In Mihalis Yannakakis and Serge Abiteboul, editors, *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA*, pages 266–276. ACM Press, 1995. doi:10.1145/212433.212474.

© Moshe Y. Vardi;
licensed under Creative Commons License CC-BY 4.0

The Complexity of Checking Quasi-Identities over Finite Algebras with a Mal'cev Term

Erhard Aichinger
Institute for Algebra, Johannes Kepler Universität Linz, Austria

Simon Grünbacher
Institute for Algebra, Johannes Kepler Universität Linz, Austria

Abstract

We consider finite algebraic structures and ask whether every solution of a given system of equations satisfies some other equation. This can be formulated as checking the validity of certain first order formulae called *quasi-identities*. Checking the validity of quasi-identities is closely linked to solving systems of equations. For systems of equations over finite algebras with finitely many fundamental operations, a complete P/NPC dichotomy is known, while the situation appears to be more complicated for single equations. The complexity of checking the validity of a quasi-identity lies between the complexity of term equivalence (checking whether two terms induce the same function) and the complexity of solving systems of polynomial equations. We prove that for each finite algebra with a Mal'cev term and finitely many fundamental operations, checking the validity of quasi-identities is coNP-complete if the algebra is not abelian, and in P when the algebra is abelian.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Theory of computation → Complexity classes

Keywords and phrases quasi-identities, conditional identities, systems of equations

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.4

Funding Supported by the Austrian Science Fund FWF P33878: Equations in Universal Algebra.

Acknowledgements The authors thank M. Behrisch for discussions on this problem, the referees of a previous version for their criticism that helped to improve the result, and the referees of the present version for several suggestions improving the presentation.

1 Introduction

The computational complexity of solving equations over some fixed finite algebra has been an active field of research for the last two decades, and several different problems related to solving equations have been studied in the literature. The most general problem that we consider is that of solving systems of polynomial equations (PolSysSat). For this problem, Goldmann and Russel [9] have proved a P/NPC dichotomy for groups, which was later generalized to algebras in congruence modular varieties by Larose and Zádori [15, 22]. For arbitrary finite algebras with finitely many fundamental operations, a dichotomy follows from [3, 23] because solving polynomial systems can be seen as a constraint satisfaction problem [15, Theorem 2.2]. If the input is restricted to a single equation (PolSat), it becomes easier for some algebras, including nilpotent rings and groups [12]. No dichotomy theorem is known for PolSat and in fact, recent results suggest that such a dichotomy might not exist [21, 14]. The situation is similar for the problem of checking whether two polynomials induce the same function (PolEqv).

In this paper, we ask whether all solutions of a system of term equations over an algebraic structure $\mathbf{A} = (A, (f_i)_{i\in I})$ satisfy some other equation. Solving this problem, we can determine whether a set $S = \{\boldsymbol{x} \in A^n \mid \bigwedge_{i\in\underline{k}} s_i(\boldsymbol{x}) = t_i(\boldsymbol{x})\}$ defined as the solution set of a system of term equations is contained in another set $U = \{\boldsymbol{x} \in A^n \mid u(\boldsymbol{x}) = v(\boldsymbol{x})\}$.


© Erhard Aichinger and Simon Grünbacher;
licensed under Creative Commons License CC-BY 4.0

(For $k \in \mathbb{N}_0$, we use $\underline{k}$ as an abbreviation for $\{1, 2, \ldots, k\}$.) This problem arises naturally in algebraic geometry and has therefore motivated considerable mathematical insights: for algebraically closed fields, Hilbert's Nullstellensatz reduces this question to the radical membership problem of a multivariate polynomial ring, and in a finite field $\mathbb{F}_q$, a polynomial $u \in \mathbb{F}_q[x_1, \ldots, x_n]$ vanishes at all solutions of $s_1(x_1, \ldots, x_n) = \cdots = s_k(x_1, \ldots, x_n) = 0$ if and only if there are $a_1, \ldots, a_k, b_1, \ldots, b_n \in \mathbb{F}_q[x_1, \ldots, x_n]$ such that $u = \sum_{i=1}^{k} a_i\, s_i + \sum_{i=1}^{n} b_i\, (x_i^q - x_i)$ ([19], [6, Theorem 7]). In the present note, we consider this problem for finite algebras from the viewpoint of universal algebra [5, 17], and we seek to determine its computational complexity. For an algebraic structure $\mathbf{A} = (A, (f_i)_{i \in I})$, an *equation* is a formula $s(x_1, \ldots, x_n) = t(x_1, \ldots, x_n)$, where s and t are terms built from the operation symbols f_i and the variables $x_1, \ldots, x_n$. A *solution* to this equation is a tuple $(a_1, \ldots, a_n) \in A^n$ such that $s(a_1, \ldots, a_n) = t(a_1, \ldots, a_n)$. Given $k, n \in \mathbb{N}$, equations $s_i(x_1, \ldots, x_n) = t_i(x_1, \ldots, x_n)$ $(i \in \{1, \ldots, k\})$ and an equation $u(x_1, \ldots, x_n) = v(x_1, \ldots, x_n)$, the solutions of $\bigwedge_{i \in \underline{k}} s_i(x_1, \ldots, x_n) = t_i(x_1, \ldots, x_n)$ are contained in the solutions of $u(x_1, \ldots, x_n) = v(x_1, \ldots, x_n)$ if and only if the first order formula

$$\forall \boldsymbol{x} \,:\, \Big(\bigwedge_{i \in \underline{k}} s_i(\boldsymbol{x}) = t_i(\boldsymbol{x}) \Big) \Rightarrow u(\boldsymbol{x}) = v(\boldsymbol{x})$$

holds in $\mathbf{A}$. Such a formula is called a *conditional identity* or *quasi-identity*. For a given algebra $\mathbf{A}$, the problem QuasiIdVal($\mathbf{A}$) is to decide whether a given quasi-identity holds in $\mathbf{A}$. For example, in the group S_3, $\forall x_1, x_2 \,:\, (x_1 \cdot (x_1 \cdot x_1) = 1 \wedge x_2 \cdot (x_2 \cdot x_2) = 1) \Rightarrow x_1 \cdot x_2 = x_2 \cdot x_1$ is a valid quasi-identity expressing that all elements of order dividing 3 commute. For semigroups, this decision problem has been investigated in [20].

▶ **Definition 1.1.** Let $\mathbf{A}$ be an algebra. Then the *quasi-identity validity* problem over $\mathbf{A}$, QuasiIdVal($\mathbf{A}$), is defined as follows: Given terms $s_1, t_1, \ldots, s_k, t_k, u, v$ over the variables $(x_i)_{i \in \underline{n}}$ in the language of the algebra $\mathbf{A}$, determine whether

$$\forall \boldsymbol{a} \in A^n : \Big(\bigwedge_{i \in \underline{k}} s_i(\boldsymbol{a}) = t_i(\boldsymbol{a}) \Big) \Rightarrow u(\boldsymbol{a}) = v(\boldsymbol{a})$$

holds.

We refer to $\bigwedge_{i \in \underline{k}} s_i(\boldsymbol{x}) = t_i(\boldsymbol{x})$ as the *precondition* and to $u(\boldsymbol{x}) = v(\boldsymbol{x})$ as the *conclusion* of the quasi-identity. The length of the input is defined by $l := (\sum_{i=1}^{k} ||s_i|| + ||t_i||) + ||u|| + ||v||$, where $||t||$ denotes the *length* of t as defined, e.g., in [2]. There are constants $c_1, c_2 \in \mathbb{R}$ such that over a finite alphabet with at least two letters, we can encode the input into a string of length x with $c_1\, l \leq x \leq c_2\, l \log(l)$; the $\log(l)$ factor comes from the fact that the input terms may contain at most l different variables, for which we find names of length $\leq c_3 \log(l)$. Since our results are a mere P/coNPC-distinction, measuring computation time in terms of l is a sufficient degree of precision. We will use the notions from complexity theory as they are defined in [18]; in particular, coNP-completeness is to be understood with respect to polynomial time many-one reductions. For a finite algebra of finite type, a tuple $\boldsymbol{a}$ from A^n for which the precondition is true and the conclusion is false can serve as a certificate for the answer "no", and therefore the problem is in coNP. Since QuasiIdVal is closely related to solving polynomial systems and checking identities, the complexity of QuasiIdVal is often determined by the connection to these problems.

2 Complexity determined by the relation to other problems on terms and polynomials

For a finite algebra $\mathbf{A}$, we first compare QuasiIdVal($\mathbf{A}$) to the problem PolSysSat($\mathbf{A}$) of solving systems of polynomial equations over $\mathbf{A}$, which was studied, e.g., in [15]. Here, a *polynomial equation* over $\mathbf{A}$ is a formula $s(x_1,\dots,x_n,b_1,\dots,b_m) = t(x_1,\dots,x_n,b_1,\dots,b_m)$, where s,t are terms and $b_1,\dots,b_m \in A$; a *solution* is an $\boldsymbol{a} \in A^n$ with

$$s(a_1,\dots,a_n,b_1,\dots,b_m) = t(a_1,\dots,a_n,b_1,\dots,b_m).$$

We first observe that PolSysSat($\mathbf{A}$) is harder than QuasiIdVal($\mathbf{A}$) because an instance of (the negation of) QuasiIdVal($\mathbf{A}$) can be reduced to solving a constant number of instances of PolSysSat($\mathbf{A}$): given an instance

$$\forall \boldsymbol{x} \in A^n : (\bigwedge_{i\in\underline{k}} s_i(\boldsymbol{x}) = t_i(\boldsymbol{x})) \Rightarrow u(\boldsymbol{x}) = v(\boldsymbol{x}) \tag{2.1}$$

of QuasiIdVal($\mathbf{A}$), we observe that (2.1) is not valid if and only if there are $a,b \in A$ with $a \neq b$ such that the system

$$(\bigwedge_{i\in\underline{k}} s_i(\boldsymbol{x}) = t_i(\boldsymbol{x})) \wedge u(\boldsymbol{x}) = a \wedge v(\boldsymbol{x}) = b \tag{2.2}$$

has a solution. We note that this reduction from (the negation of) QuasiIdVal to PolSysSat is not a many-one reduction, but just a truth table reduction: we solve $|A|\cdot(|A|-1)$ polynomial systems in order to find a counterexample to the validity of a given quasi-identity.

When $\mathbf{A}$ has all constant functions in its term operations, which means that each polynomial function of $\mathbf{A}$ is a term function, and $|A| \geq 2$, then PolSysSat($\mathbf{A}$) can be reduced to (the negation of) QuasiIdVal($\mathbf{A}$) by observing that the system $\bigwedge_{i\in\underline{k}} p_i(\boldsymbol{x}) = q_i(\boldsymbol{x})$ has a solution if and only if

$$(\bigwedge_{i\in\underline{k}} p_i(\boldsymbol{x}) = q_i(\boldsymbol{x})) \Rightarrow y = z$$

is not valid, where y,z are distinct variables that do not appear among the x_i's. Without constants, this reduction shows that QuasiIdVal($\mathbf{A}$) is harder than solving systems of *term* equations (TermSysSat($\mathbf{A}$)). If $\mathbf{A}$ is a group or a ring, every system of term equations is satisfied by $\boldsymbol{x} = (1,\dots,1)$ (respectively $\boldsymbol{x} = (0,\dots,0)$). This means that solving systems of term equations is trivial for groups and rings, and therefore for these algebras, the relation to TermSysSat does not yield a meaningful lower bound on the complexity of QuasiIdVal($\mathbf{A}$).

Such a bound can be obtained by comparing QuasiIdVal($\mathbf{A}$) to the term equivalence problem TermEqv($\mathbf{A}$). This is the problem that asks whether two given terms s,t induce the same function on $\mathbf{A}$. The quasi-identity validity problem is at least as hard as checking the validity of a single term equality because $\forall \boldsymbol{x} \in A^n : s(\boldsymbol{x}) = t(\boldsymbol{x})$ is valid if and only if the quasi-identity

$$\forall \boldsymbol{x} \in A^n, y \in A : y = y \Rightarrow s(\boldsymbol{x}) = t(\boldsymbol{x})$$

holds. For an algebra $\mathbf{A}$, let Clo($\mathbf{A}$) denote the set of its finitary term functions (this set has also been called the *clone* of $\mathbf{A}$). Strengthening the relation between TermEqv and QuasiIdVal given above, we have that for every algebra $\mathbf{B}$ of finite type with Clo($\mathbf{B}$) $\subseteq$

$\mathrm{Clo}(\mathbf{A})$, TermEqv($\mathbf{B}$) can be reduced to QuasiIdVal($\mathbf{A}$); we explain this reduction by an example. Let $\mathbf{A} = (A, \cdot)$ be a group, and let $\mathbf{B} = (B, f, g)$ with $f(x, y) := y \cdot (x \cdot y)$ and $g(x) := x \cdot x$. Suppose that we want to check whether the equality

$$f(g(x_1), f(x_2, x_3)) = g(x_3) \tag{2.3}$$

is valid in $\mathbf{B}$. Then by replacing f and g with their definition in terms of the operation $\cdot$, we obtain that (2.3) is valid in $\mathbf{B}$ if and only if the quasi-identity

$$x_1 = x_1 \Rightarrow (x_3 \cdot (x_2 \cdot x_3)) \cdot ((x_1 \cdot x_1) \cdot (x_3 \cdot (x_2 \cdot x_3))) = x_3 \cdot x_3 \tag{2.4}$$

holds in $\mathbf{A}$. However, in this example, we see that the quasi-identity in (2.4) is longer than the equality in (2.3) from which we started. In general, the reduction given above may produce a quasi-identity whose size is exponential in the length of the given equality, and therefore does not qualify as a polynomial time many-one reduction. This exponential increase in length is avoided if we introduce variables for all subterms that appear in (2.3). We observe that (2.3) is valid in $\mathbf{B}$ if and only if the quasi-identity

$$(z_1 = x_1 \cdot x_1 \wedge z_2 = x_3 \cdot (x_2 \cdot x_3) \wedge z_3 = z_2 \cdot (z_1 \cdot z_2) \wedge z_4 = x_3 \cdot x_3) \Rightarrow z_3 = z_4$$

holds in $\mathbf{A}$, and the size of the quasi-identity obtained in this way is bounded by a polynomial in the size of the input equality. Hence in this way, we obtain a polynomial time reduction of TermEqv($\mathbf{B}$) to QuasiIdVal($\mathbf{A}$). The problem TermEqv has been investigated for some classes of finite groups and rings (see e.g. [4, 9]). In [13], it is proved that for every non-nilpotent group $\mathbf{A}$, there is an algebra $\mathbf{B}$ with $\mathrm{Clo}(\mathbf{B}) = \mathrm{Clo}(\mathbf{A})$ such that TermEqv($\mathbf{B}$) is coNP-complete; this implies that every non-nilpotent group has a coNP-complete quasi-identity validity problem. We summarize these consequences of the literature in Table 1. In this table, we do not require that a ring has a unit element; a ring is *nilpotent* if there

Table 1 Complexity of the studied problems as known before the present note.

$\mathbf{A}$	TermEqv($\mathbf{A}$)	QuasiIdVal($\mathbf{A}$)	PolSysSat($\mathbf{A}$)
module/abelian group/zero ring	P	P	P ([9, 15])
nonabelian nilpotent group	P ([9])	open	NPC ([9])
non-nilpotent solvable group	partially open	coNPC ([13])	NPC ([9])
non-solvable group	coNPC ([9])	coNPC	NPC ([9])
non-zero nilpotent ring	P ([4])	open	NPC ([15])
non-nilpotent ring	coNPC ([4])	coNPC	NPC ([15])

is a $k \in \mathbb{N}$ such that every product of at least k elements is 0, and a ring $\mathbf{R}$ with $r \cdot s = 0$ for all $r, s \in R$ is called a *zero ring*. Theorem 3.1 establishes that in both cases in which the complexity of QuasiIdVal is referred to as *open* in the above table, the answer is *coNP-complete*.

In [20], M. Volkov constructs a 10-element semigroup $\mathbf{Q}$ such that TermEqv($\mathbf{Q}$) is in P and QuasiIdVal($\mathbf{Q}$) is coNP-complete. Combining the results of the present note with [9, 4], we obtain that every finite nilpotent nonabelian group and every finite nilpotent nonzero ring, such as the quaternion group and the ring $2\mathbb{Z}_8$, have a tractable term equivalence and a coNP-complete quasi-identity validity problem.

3 Complexity for finite Mal'cev algebras

Groups and rings are all contained in the larger class of Mal'cev algebras. An algebra $\mathbf{A}$ is a *Mal'cev algebra* if it has a ternary term function $M \in \mathrm{Clo}(\mathbf{A})$ (called a *Mal'cev term*) such that $M(a,b,b) = M(b,b,a) = a$ for all $a, b \in A$ (cf. [16, 17]). For a group $M(x,y,z) = xy^{-1}z$, and for a ring $M(x,y,z) = x - y + z$ are examples of Mal'cev terms. In the present note, we show that for a finite Mal'cev algebra of finite type (i.e., having finitely many fundamental operations), QuasiIdVal is either in P or coNP-complete, and that the dividing line is the same as for PolSysSat. This dividing line can be expressed using a notion from universal algebra. Denoting the set of n-ary term functions of an algebra $\mathbf{A}$ by $\mathrm{Clo}_n(\mathbf{A})$, the algebra $\mathbf{A}$ is called *abelian* if for all $m \in \mathbb{N}$, for all $t \in \mathrm{Clo}_{1+m}(\mathbf{A})$ and for all $a, b \in A$ and $\boldsymbol{c}, \boldsymbol{d} \in A^m$ with $t(a, \boldsymbol{c}) = t(a, \boldsymbol{d})$, also $t(b, \boldsymbol{c}) = t(b, \boldsymbol{d})$ holds [17, Definition 4.146]. A group is abelian in this sense if and only if its operation is commutative, i.e., it is abelian in the sense of classic algebra, and a ring is abelian if and only if it is a zero ring. The main result of this note, proved in Section 5, is:

▶ **Theorem 3.1.** *Let $\mathbf{A}$ be a finite nonabelian Mal'cev algebra of finite type. Then QuasiIdVal($\mathbf{A}$) is coNP-complete.*

If $\mathbf{A}$ is abelian, then it follows from [15] that PolSysSat($\mathbf{A}$) is in P. Then as observed at the beginning of Section 2, the validity of a given quasi-identity Φ can be determined by checking the solvability of $|A| \cdot (|A| - 1)$ systems of polynomial equations. Hence we obtain:

▶ **Corollary 3.2.** *Let $\mathbf{A}$ be a finite Mal'cev algebra of finite type. Then QuasiIdVal($\mathbf{A}$) is in P if $\mathbf{A}$ is abelian, and coNP-complete otherwise.*

4 Preliminaries on Mal'cev Algebras

Our proof of Theorem 3.1 requires the notions *congruence* and *commutator* from universal algebra, and we therefore introduce these briefly. A *congruence relation* of an algebra $\mathbf{A} = (A, F)$ is an equivalence relation α on A that satisfies the *compatibility condition*

$$(a_1, b_1) \in \alpha, \dots (a_n, b_n) \in \alpha \Longrightarrow (f(a_1, \dots, a_n), f(b_1, \dots, b_n)) \in \alpha$$

for each $n \in \mathbb{N}$, for each n-ary basic operation f of $\mathbf{A}$, and for each $\boldsymbol{a}, \boldsymbol{b} \in A^n$. For such an α and for an n-ary basic operation f of $\mathbf{A}$, the operation $f_\alpha(a_1/\alpha, \dots, a_n/\alpha) := f(a_1, \dots, a_n)/\alpha$ is well-defined; this allows to define a factor algebra $\mathbf{A}/\alpha$. For a group $\mathbf{G}$, each congruence relation α is of the form

$$\alpha = \{(g_1, g_2) \in G \times G \mid g_1^{-1} g_2 \in N_\alpha\},$$

where N_α is a normal subgroup of $\mathbf{G}$; similarly, every congruence of a ring $\mathbf{R}$ is of the form

$$\alpha = \{(r_1, r_2) \in R \times R \mid r_1 - r_2 \in I_\alpha\}$$

for some ideal I_α of $\mathbf{R}$. For $k \in \mathbb{N}$ and $\boldsymbol{a} = (a_1, \dots, a_k)$ and $\boldsymbol{b} = (b_1, \dots, b_k) \in A^k$, we write $\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})$ for the smallest congruence relation on A containing $\{(a_1, b_1), \dots, (a_k, b_k)\}$ as a subset, and call $\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})$ the *congruence* on $\mathbf{A}$ that is *generated by* $\{(a_1, b_1), \dots, (a_k, b_k)\}$. For the algebras that we consider, the generated congruence has a useful description using term operations. In fact, if $\mathbf{A}$ is a Mal'cev algebra, we have

$$\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}) = \{(t(\boldsymbol{a}, \boldsymbol{e}), t(\boldsymbol{b}, \boldsymbol{e})) \mid m \in \mathbb{N}, t \in \mathrm{Clo}_{k+m}(\mathbf{A}), \boldsymbol{e} \in A^m\};$$

it is easy to verify that the right hand side of this equation is a subuniverse of $\mathbf{A} \times \mathbf{A}$ containing $0_A = \{(a, a) \mid a \in A\}$; in a Mal'cev algebra, all subuniverses of $\mathbf{A} \times \mathbf{A}$ containing 0_A are congruence relations (cf., e.g., [11, Lemma 5.22]). We denote the set of congruence relations on $\mathbf{A}$ by $\mathrm{Con}(\mathbf{A})$.

The other concept from universal algebra that we use is the *commutator*. Here, one associates a congruence relation γ, denoted by $[\alpha, \beta]$, with every pair of congruences α, β from $\mathrm{Con}(\mathbf{A})$. The universal algebraic construction generalizes the *commutator subgroup* $[N, M]$ of two normal subgroups N, M of G, which is the subgroup generated by $\{n^{-1}m^{-1}nm \mid n \in N, m \in M\}$, and the *ideal product* IJ of two ideals I, J in rings, which is the ideal generated by $\{ij \mid i \in I, j \in J\} \cup \{ji \mid i \in I, j \in J\}$. For generalizing these concepts to arbitrary universal algebras, one starts by defining a relation $C(\alpha, \beta; \eta)$ between three congruences α, β, η that is designed to guarantee that $[\alpha, \beta] \leq \gamma$. Doing this formally, we say that for $\alpha, \beta, \eta \in \mathrm{Con}(\mathbf{A})$, the congruence α *centralizes* β *modulo* η if for all $m, n \in \mathbb{N}$, for all $\boldsymbol{a}, \boldsymbol{b} \in A^m$ and $\boldsymbol{c}, \boldsymbol{d} \in A^n$ with $\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}) \leq \alpha$ and $\Theta_{\mathbf{A}}(\boldsymbol{c}, \boldsymbol{d}) \leq \beta$ and for all $t \in \mathrm{Clo}_{m+n}(\mathbf{A})$ we have

$$(t(\boldsymbol{a}, \boldsymbol{c}), t(\boldsymbol{a}, \boldsymbol{d})) \in \eta \Rightarrow (t(\boldsymbol{b}, \boldsymbol{c}), t(\boldsymbol{b}, \boldsymbol{d})) \in \eta. \tag{4.1}$$

The *commutator* of α and β, denoted by $[\alpha, \beta]$, is then defined to be the smallest congruence relation η on $\mathbf{A}$ such that α centralizes β modulo η. What is given here is a reformulation of [8, Definition 3.2(2)]; other sources give slightly different, but equivalent, definitions of the commutator. For example, in [17, Definition 4.148], m is restricted to be equal to 1, which gives an equivalent condition (a rough explanation of this equivalence is that in the implication (4.1), we may change $\boldsymbol{a}$ to $\boldsymbol{b}$ by changing one component of $\boldsymbol{a}$ at a time, repeating this m times, see also [1, Proposition 2.1(2)]). Using the concept of commutators, we see that an algebra $\mathbf{A}$ is abelian if and only if $[1_A, 1_A] = 0_A$.

From this definition, it is not easy to determine the commutator $[\alpha, \beta]$ of two congruences, and therefore one seeks descriptions of $[\alpha, \beta]$ that allow us to compute its elements more directly. In Lemmas 4.2 and 4.3, we provide parametrizations of those commutators $[\alpha, \beta]$ where one of α and β is equal to $1_A = A \times A$. It is known that if $\mathbf{A}$ has a Mal'cev term, then $[\alpha, \beta] = [\beta, \alpha]$ ([17, Exercise 4.156(13)], a proof is written in [1, Lemma 2.5]). Hence $[\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A] = [1_A, \Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})]$.

▶ **Definition 4.1.** Let $\mathbf{A}$ be an algebra, let $k, m \in \mathbb{N}$, and let $t \in \mathrm{Clo}_{k+m}(\mathbf{A})$. Then $\boldsymbol{z} \in A^m$ is called a *right zero* of t if $t(\boldsymbol{x}, \boldsymbol{z}) = t(\boldsymbol{y}, \boldsymbol{z})$ for all $\boldsymbol{x}, \boldsymbol{y} \in A^k$.

▶ **Lemma 4.2.** *Let $\mathbf{A}$ be a Mal'cev algebra, let $k \in \mathbb{N}$, and let $\boldsymbol{a}, \boldsymbol{b} \in A^k$. Then $[\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A]$ $= \{(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \mid m \in \mathbb{N}, \boldsymbol{w} \in A^m,$ and $t \in \mathrm{Clo}_{k+m}(\mathbf{A})$ such that t has a right zero$\}$.*

Proof. We define

$$\Psi(\boldsymbol{a}, \boldsymbol{b}) := \{(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \mid m \in \mathbb{N}, \boldsymbol{w} \in A^m, t \in \mathrm{Clo}_{k+m}(\mathbf{A}), t \text{ has a right zero}\}.$$

For $\supseteq$, we observe that for a term t with a right zero $\boldsymbol{z}$, we have $t(\boldsymbol{a}, \boldsymbol{z}) = t(\boldsymbol{b}, \boldsymbol{z})$. Now using the term condition (4.1) (for $t'(\boldsymbol{x}, \boldsymbol{y}) := t(\boldsymbol{y}, \boldsymbol{x})$ and $\eta := [1_A, \Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})]$) and observing that, obviously, $\Theta_{\mathbf{A}}(\boldsymbol{z}, \boldsymbol{w}) \leq 1_A$, we obtain $(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \in [1_A, \Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})]$.

For $\subseteq$, we first show that $\Psi(\boldsymbol{a}, \boldsymbol{b})$ is a congruence of $\mathbf{A}$. To this end, we show that for all $n, p \in \mathbb{N}$, $s \in \mathrm{Clo}_{n+p}(\mathbf{A})$, $(c_1, d_1), \ldots, (c_n, d_n) \in \Psi(\boldsymbol{a}, \boldsymbol{b})$ and $\boldsymbol{e} \in A^p$, we have

$$(s(c_1, \ldots, c_n, \boldsymbol{e}), s(d_1, \ldots, d_n, \boldsymbol{e})) \in \Psi(\boldsymbol{a}, \boldsymbol{b}). \tag{4.2}$$

For each $i \in \underline{n}$, let t_i be a term function with right zero $\boldsymbol{z}^{(i)}$ and let $\boldsymbol{w}^{(i)}$ be a tuple from $\mathbf{A}$ such that $(c_i, d_i) = (t_i(\boldsymbol{a}, \boldsymbol{w}^{(i)}), t_i(\boldsymbol{b}, \boldsymbol{w}^{(i)}))$. Let

$$s'(\boldsymbol{x}, \boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(n)}, \boldsymbol{z}) := s(t_1(\boldsymbol{x}, \boldsymbol{y}^{(1)}), \dots, t_n(\boldsymbol{x}, \boldsymbol{y}^{(n)}), \boldsymbol{z}).$$

Since $\boldsymbol{z}^{(i)}$ is a right zero of t_i, $(\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(n)}, \boldsymbol{e})$ is a right zero of s'. Hence

$$(s'(\boldsymbol{a}, \boldsymbol{w}^{(1)}, \dots, \boldsymbol{w}^{(n)}, \boldsymbol{e}), s'(\boldsymbol{b}, \boldsymbol{w}^{(1)}, \dots, \boldsymbol{w}^{(n)}, \boldsymbol{e})) \in \Psi(\boldsymbol{a}, \boldsymbol{b}),$$

and thus $(s(\boldsymbol{c}, \boldsymbol{e}), s(\boldsymbol{d}, \boldsymbol{e})) \in \Psi(\boldsymbol{a}, \boldsymbol{b})$, completing the proof of (4.2).

From (4.2), we see that $\Psi(\boldsymbol{a}, \boldsymbol{b})$ is a subuniverse of $\mathbf{A} \times \mathbf{A}$ that contains the diagonal 0_A as a subset. Since $\mathbf{A}$ is a Mal'cev algebra, this implies that $\Psi(\boldsymbol{a}, \boldsymbol{b})$ is a congruence of $\mathbf{A}$.

Next, we show that $\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})$ centralizes 1_A modulo $\Psi(\boldsymbol{a}, \boldsymbol{b})$. To this end, let $f, g \in A$ with $(f, g) \in \Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})$, let $m \in \mathbb{N}$, let $\boldsymbol{c}, \boldsymbol{d} \in A^m$, and let $t \in \mathrm{Clo}_{1+m}(\boldsymbol{A})$ be such that $(t(f, \boldsymbol{c}), t(f, \boldsymbol{d})) \in \Psi(\boldsymbol{a}, \boldsymbol{b})$. For proving the centralizing property, we need to show $(t(g, \boldsymbol{c}), t(g, \boldsymbol{d})) \in \Psi(\boldsymbol{a}, \boldsymbol{b})$. First, we observe that since $(f, g) \in \Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})$, there are $n \in \mathbb{N}$, a term function $r \in \mathrm{Clo}_{k+n}(\mathbf{A})$ and $\boldsymbol{e} \in A^n$ such that $f = r(\boldsymbol{a}, \boldsymbol{e})$ and $g = r(\boldsymbol{b}, \boldsymbol{e})$. Denoting the Mal'cev term of $\mathbf{A}$ by $M(x, y, z)$, we define a term function $q \in \mathrm{Clo}_{k+(m+n+m+1)}(\mathbf{A})$ by

$$q(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}, w) := M(t(r(\boldsymbol{x}, \boldsymbol{u}), \boldsymbol{y}), t(r(\boldsymbol{x}, \boldsymbol{u}), \boldsymbol{v}), t(w, \boldsymbol{v})).$$

Then $(\boldsymbol{y}, \boldsymbol{u}, \boldsymbol{v}, w) := (\boldsymbol{c}, \boldsymbol{e}, \boldsymbol{c}, g)$ is a right zero of q because for all $\boldsymbol{x}, \boldsymbol{x}' \in A^k$, we have

$$\begin{aligned} q(\boldsymbol{x}, \boldsymbol{c}, \boldsymbol{e}, \boldsymbol{c}, g) &= M(t(r(\boldsymbol{x}, \boldsymbol{e}), \boldsymbol{c}), t(r(\boldsymbol{x}, \boldsymbol{e}), \boldsymbol{c}), t(g, \boldsymbol{c})) \\ &= t(g, \boldsymbol{c}) \\ &= M(t(r(\boldsymbol{x}', \boldsymbol{e}), \boldsymbol{c}), t(r(\boldsymbol{x}', \boldsymbol{e}), \boldsymbol{c}), t(g, \boldsymbol{c})) \\ &= q(\boldsymbol{x}', \boldsymbol{c}, \boldsymbol{e}, \boldsymbol{c}, g). \end{aligned}$$

Hence

$$(q(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{c}, g), q(\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{c}, g)) \in \Psi(\boldsymbol{a}, \boldsymbol{b}).$$

We have

$$\begin{aligned} q(\boldsymbol{a}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{c}, g) &= M(t(r(\boldsymbol{a}, \boldsymbol{e}), \boldsymbol{d}), t(r(\boldsymbol{a}, \boldsymbol{e}), \boldsymbol{c}), t(g, \boldsymbol{c})) \\ &= M(t(f, \boldsymbol{d}), t(f, \boldsymbol{c}), t(g, \boldsymbol{c})) \end{aligned}$$

and

$$\begin{aligned} q(\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{c}, g)) &= M(t(r(\boldsymbol{b}, \boldsymbol{e}), \boldsymbol{d}), t(r(\boldsymbol{b}, \boldsymbol{e}), \boldsymbol{c}), t(g, \boldsymbol{c})) \\ &= M(t(g, \boldsymbol{d}), t(g, \boldsymbol{c}), t(g, \boldsymbol{c})) \\ &= t(g, \boldsymbol{d}). \end{aligned}$$

From the definition of $\Psi(\boldsymbol{a}, \boldsymbol{b})$, we therefore obtain

$$\big(M(t(f, \boldsymbol{d}), t(f, \boldsymbol{c}), t(g, \boldsymbol{c})),\ t(g, \boldsymbol{d})\big) \in \Psi(\boldsymbol{a}, \boldsymbol{b}).$$

Since $(t(f, \boldsymbol{c}), t(f, \boldsymbol{d})) \in \Psi(\boldsymbol{a}, \boldsymbol{b})$, we have that $M(t(f, \boldsymbol{d}), t(f, \boldsymbol{c}), t(g, \boldsymbol{c}))$ is congruent modulo $\Psi(\boldsymbol{a}, \boldsymbol{b})$ to $M(t(f, \boldsymbol{d}), t(f, \boldsymbol{d}), t(g, \boldsymbol{c})) = t(g, \boldsymbol{c})$. Thus we have $(t(g, \boldsymbol{c}), t(g, \boldsymbol{d})) \in \Psi(\boldsymbol{a}, \boldsymbol{b})$. Hence $\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})$ centralizes 1_A modulo $\Psi(\boldsymbol{a}, \boldsymbol{b})$. Since $[\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A]$ is defined as the intersection of all congruences ψ such that $\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b})$ centralizes 1_A modulo ψ, we obtain $[\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A] \subseteq \Psi(\boldsymbol{a}, \boldsymbol{b})$. ◀

The next lemma tells that we can find one single term to parametrize commutators of the form $[\alpha, 1_A]$.

▶ **Lemma 4.3.** *Let* $\mathbf{A}$ *be a finite Mal'cev algebra and let* $k \in \mathbb{N}$. *Then there exist* $m \in \mathbb{N}$ *and* $t \in \mathrm{Clo}_{k+m}(\mathbf{A})$ *such that*

$$t \text{ has a right zero, and for all } \boldsymbol{a}, \boldsymbol{b} \in A^k, \quad \text{we have } [\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A] = \{(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \mid \boldsymbol{w} \in A^m\}. \quad (4.3)$$

Proof. For each $m \in \mathbb{N}$ and each $t \in \mathrm{Clo}_{k+m}(\mathbf{A})$ with a right zero, let

$$\Psi(t, \boldsymbol{a}, \boldsymbol{b}) := \{(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \mid \boldsymbol{w} \in A^m\}.$$

We order the terms in

$$T := \{t \in \mathrm{Clo}_{k+m}(\mathbf{A}) \mid m \in \mathbb{N}, t \text{ has a right zero}\}$$

by $t_1 \leq t_2$ if $\Psi(t_1, \boldsymbol{a}, \boldsymbol{b}) \subseteq \Psi(t_2, \boldsymbol{a}, \boldsymbol{b})$ for all $\boldsymbol{a}, \boldsymbol{b} \in A^k$. The relation $\leq$ is a quasi-order. For the term function $\pi(x_1, \dots, x_k, y) := y$, we have $\Psi(\pi, \boldsymbol{a}, \boldsymbol{b}) = 0_A$. Since $\Psi(t, \boldsymbol{a}, \boldsymbol{b})$ can take only finitely many values, there is $t \in T$ such that $\pi \leq t$ and t is maximal in T with respect to $\leq$. Let $m \in \mathbb{N}$ be such that $t \in \mathrm{Clo}_{k+m}(\mathbf{A})$. Our claim is that this t satisfies (4.3). By Lemma 4.2, we have $[\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A] \supseteq \{(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \mid \boldsymbol{w} \in A^m\}$ for all $\boldsymbol{a}, \boldsymbol{b} \in A^k$.

For proving the "$\subseteq$"-inclusion of (4.3), suppose that there are $\boldsymbol{c}, \boldsymbol{d} \in A^k$, $(f, g) \in [\Theta_{\mathbf{A}}(\boldsymbol{c}, \boldsymbol{d}), 1_A]$ and $(f, g) \notin \{(t(\boldsymbol{c}, \boldsymbol{w}), t(\boldsymbol{d}, \boldsymbol{w})) \mid \boldsymbol{w} \in A^m\}$. By Lemma 4.2, there are $n \in \mathbb{N}$, an $s \in \mathrm{Clo}_{k+n}(\mathbf{A})$ with a right zero and $\boldsymbol{e} \in A^n$ such that $(f, g) = (s(\boldsymbol{c}, \boldsymbol{e}), s(\boldsymbol{d}, \boldsymbol{e}))$. Denoting the Mal'cev term of $\mathbf{A}$ by $M(x, y, z)$, we define a term function $r \in \mathrm{Clo}_{k+(n+m+m)}(\mathbf{A})$ by

$$r(\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) := M(s(\boldsymbol{u}, \boldsymbol{x}), t(\boldsymbol{u}, \boldsymbol{y}), t(\boldsymbol{u}, \boldsymbol{z})).$$

Let $\boldsymbol{h}, \boldsymbol{i}$ be the right zeros of s and t, respectively. Then $(\boldsymbol{h}, \boldsymbol{i}, \boldsymbol{i})$ is a right zero of r. We first show that $t \leq r$. To this end, let $\boldsymbol{a}, \boldsymbol{b} \in A^k$ and $\boldsymbol{w} \in A^m$. We want to show that $(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \in \Psi(r, \boldsymbol{a}, \boldsymbol{b})$. Since $\boldsymbol{h}$ is a right zero of s, we have $s(\boldsymbol{a}, \boldsymbol{h}) = s(\boldsymbol{b}, \boldsymbol{h})$, and thus $(s(\boldsymbol{a}, \boldsymbol{h}), s(\boldsymbol{b}, \boldsymbol{h})) \in 0_A = \Psi(\pi, \boldsymbol{a}, \boldsymbol{b})$. Since $\pi \leq t$, we therefore have $(s(\boldsymbol{a}, \boldsymbol{h}), s(\boldsymbol{b}, \boldsymbol{h})) \in \Psi(t, \boldsymbol{a}, \boldsymbol{b})$ and thus there is $\boldsymbol{v} \in A^m$ such that $(t(\boldsymbol{a}, \boldsymbol{v}), t(\boldsymbol{b}, \boldsymbol{v})) = (s(\boldsymbol{a}, \boldsymbol{h}), s(\boldsymbol{b}, \boldsymbol{h}))$. Hence

$$\begin{aligned}(r(\boldsymbol{a}, \boldsymbol{h}, \boldsymbol{v}, \boldsymbol{w}), r(\boldsymbol{b}, \boldsymbol{h}, \boldsymbol{v}, \boldsymbol{w})) &= \big(M(s(\boldsymbol{a}, \boldsymbol{h}), t(\boldsymbol{a}, \boldsymbol{v}), t(\boldsymbol{a}, \boldsymbol{w})), M(s(\boldsymbol{b}, \boldsymbol{h}), t(\boldsymbol{b}, \boldsymbol{v}), t(\boldsymbol{b}, \boldsymbol{w}))\big) \\ &= \big(M(s(\boldsymbol{a}, \boldsymbol{h}), s(\boldsymbol{a}, \boldsymbol{h}), t(\boldsymbol{a}, \boldsymbol{w})), M(s(\boldsymbol{b}, \boldsymbol{h}), s(\boldsymbol{b}, \boldsymbol{h}), t(\boldsymbol{b}, \boldsymbol{w}))\big) \\ &= (t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})),\end{aligned}$$

and thus $(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \in \Psi(r, \boldsymbol{a}, \boldsymbol{b})$. Hence $\Psi(t, \boldsymbol{a}, \boldsymbol{b}) \subseteq \Psi(r, \boldsymbol{a}, \boldsymbol{b})$ and thus $t \leq r$.

We will now establish that $r \not\leq t$. To this end, we observe that (f, g) is an element of $\Psi(r, \boldsymbol{c}, \boldsymbol{d})$ because

$$(s(\boldsymbol{c}, \boldsymbol{e}), s(\boldsymbol{d}, \boldsymbol{e})) = (r(\boldsymbol{c}, \boldsymbol{e}, \boldsymbol{i}, \boldsymbol{i}), r(\boldsymbol{d}, \boldsymbol{e}, \boldsymbol{i}, \boldsymbol{i})) \in \Psi(r, \boldsymbol{c}, \boldsymbol{d}).$$

Since $(f, g) \in \Psi(r, \boldsymbol{c}, \boldsymbol{d})$ and by assumption $(f, g) \notin \Psi(t, \boldsymbol{c}, \boldsymbol{d})$, we have $r \not\leq t$.

Now $t \leq r$ and $r \not\leq t$ contradict the maximality of t, which completes the proof that

$$[\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A] \subseteq \{(t(\boldsymbol{a}, \boldsymbol{w}), t(\boldsymbol{b}, \boldsymbol{w})) \mid \boldsymbol{w} \in A^m\}$$

for all $\boldsymbol{a}, \boldsymbol{b} \in A^k$. ◀

5 Graphs and the completeness proof

For establishing coNP-completeness, we use an NP-complete problem from graph theory. In our proof, we take the formal viewpoint to consider a *graph* as a relational structure $\mathbb{G} = (G, \rho)$ such that $\rho \subseteq G \times G$ is symmetric. We call the graph $\mathbb{G}$ *loopless* if there is no $v \in G$ with $(v, v) \in \rho$, and we say that $\mathbb{G}$ *contains a triangle* if there are $u, v, w \in G$ with $(u, v) \in \rho$, $(v, w) \in \rho$, $(u, w) \in \rho$. For a graph $\mathbb{H}$, the computational problem $\mathbb{H}$-Coloring studied in [10] asks whether for a finite input graph $\mathbb{G}$, there exists a homomorphism from $\mathbb{G}$ to $\mathbb{H}$. Theorem 1 of [10] states that if $\mathbb{H}$ is loopless and not bipartite, then $\mathbb{H}$-Coloring is NP-complete. Since a bipartite graph cannot contain a triangle, we obtain:

▶ **Corollary 5.1** ([10]). *Let $\mathbb{H}$ be a finite loopless graph that contains a triangle. Then $\mathbb{H}$-Coloring is NP-complete.*

Let $\mathbf{A}$ be a finite Mal'cev algebra and let $\mu \in \mathrm{Con}(\mathbf{A})$. The *difference graph* of $\mathbf{A}$ with respect to μ is the graph $\mathbb{H}_\mu = (H_\mu, \rho_\mu)$, where the set of vertices H_μ is equal to A^2. The set of edges ρ_μ is defined by

$$\rho_\mu = \{(\boldsymbol{a}, \boldsymbol{b}) \mid \boldsymbol{a}, \boldsymbol{b} \in A^2,\ \mu \leq [\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{b}), 1_A]\}.$$

Intuitively, we draw an edge between two vectors $\boldsymbol{a}, \boldsymbol{b}$ from A^2 if $\boldsymbol{a}, \boldsymbol{b}$ are "sufficiently different". Here, "sufficiently different" means that the smallest congruence collapsing $\boldsymbol{a}$ and $\boldsymbol{b}$ is still large enough to have its commutator with 1_A above μ.

▶ **Lemma 5.2.** *Let $\mathbf{A}$ be a finite Mal'cev algebra, and let $\mu \in \mathrm{Con}(\mathbf{A})$ with $\mu > 0_A$. Then the difference graph $\mathbb{H}_\mu = (H_\mu, \rho_\mu)$ is loopless.*

Proof. Let $\boldsymbol{a} \in A^2$. We have to show that $(\boldsymbol{a}, \boldsymbol{a})$ is not an edge of $\mathbb{H}_\mu$. Suppose that $(\boldsymbol{a}, \boldsymbol{a}) \in \rho_\mu$. Then from the definition of ρ_μ, we obtain $\mu \leq [\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{a}), 1_A]$. Clearly, $\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{a}) = 0_A$. Furthermore, by [17, Lemma 4.149(i)], the commutator $[\alpha, \beta]$ is always contained in the intersection $\alpha \cap \beta$, and therefore $[\Theta_{\mathbf{A}}(\boldsymbol{a}, \boldsymbol{a}), 1_A] = [0_A, 1_A] = 0_A$. Hence $\mu \leq 0_A$, contradicting the assumption $\mu > 0_A$. Thus $(\boldsymbol{a}, \boldsymbol{a})$ is not an edge of $\mathbb{H}_\mu$, and therefore $\mathbb{H}_\mu$ is loopless. ◀

▶ **Lemma 5.3.** *Let $\mathbf{A}$ be a finite nonabelian Mal'cev algebra. Then there is $\beta \in \mathrm{Con}(\mathbf{A})$ such that $\beta > 0_A$ and $\mathbb{H}_\beta = (H_\beta, \rho_\beta)$ has a triangle.*

Proof. Let ζ be the center of $\mathbf{A}$, i.e., the largest congruence with $[\zeta, 1_A] = 0_A$. We note that from [17, Lemma 4.149(ii)] it follows that such a largest congruence exists: in fact, ζ is the join of all congruences ζ' with $[\zeta', 1_A] = 0_A$. In particular, every congruence $\zeta' \in \mathrm{Con}(\mathbf{A})$ with $[\zeta', 1_A] = 0_A$ satisfies $\zeta' \leq \zeta$. Since $\mathbf{A}$ is nonabelian, $\zeta < 1_A$. Thus there are $a, b \in A$ such that $(a, b) \notin \zeta$. Let

$$\beta := [\Theta_{\mathbf{A}}(a, b), 1_A].$$

We first show that $\beta > 0_A$. Suppose $\beta = 0_A$. Then $[\Theta_{\mathbf{A}}(a, b), 1_A] = 0_A$, and therefore $\Theta_{\mathbf{A}}(a, b) \leq \zeta$. Then $(a, b) \in \zeta$, contradicting the choice of a and b. Hence $\beta > 0_A$. Next, we show that $\mathbb{H}_\beta$ has a triangle. To this end, we consider the vertices $\boldsymbol{u} = \binom{a}{a}$, $\boldsymbol{v} = \binom{a}{b}$, $\boldsymbol{w} = \binom{b}{b}$ of $\mathbb{H}_\beta$. For showing that $(\boldsymbol{u}, \boldsymbol{v})$ is an edge of $\mathbb{H}_\beta$, we observe that $(\boldsymbol{u}, \boldsymbol{v}) \in \rho_\beta$ if and only if $\beta \leq [\Theta_{\mathbf{A}}(\binom{a}{a}, \binom{a}{b}), 1_A]$. Now $\Theta_{\mathbf{A}}(\binom{a}{a}, \binom{a}{b})$ is the smallest congruence containing $\{(a, a), (a, b)\}$ and therefore $\Theta_{\mathbf{A}}(\binom{a}{a}, \binom{a}{b}) = \Theta_{\mathbf{A}}(a, b)$. Hence $\beta = [\Theta_{\mathbf{A}}(a, b), 1_A] = [\Theta_{\mathbf{A}}(\binom{a}{a}, \binom{a}{b}), 1_A]$, and therefore $(\boldsymbol{u}, \boldsymbol{v}) \in \rho_\beta$. Similarly, $(\boldsymbol{u}, \boldsymbol{w})$ and $(\boldsymbol{v}, \boldsymbol{w})$ are edges of $\mathbb{H}_\beta$. ◀

On a set $\mathcal{G}$ of graphs, we can define a quasi-order by $\mathbb{G} \preceq \mathbb{H}$ if there is a homomorphism from $\mathbb{G}$ to $\mathbb{H}$. We say that $\mathbb{G}$ is *maximal* in $\mathcal{G}$ with respect to $\preceq$ if for every $\mathbb{H} \in \mathcal{G}$ with $\mathbb{G} \preceq \mathbb{H}$, we also have $\mathbb{H} \preceq \mathbb{G}$. If $\mathcal{G}$ is finite and nonempty, it must contain at least one maximal element.

▶ **Lemma 5.4.** *Let $\mathbf{A}$ be a finite nonabelian Mal'cev algebra. For each $\gamma \in \mathrm{Con}(\mathbf{A})$, let $\mathbb{H}_\gamma$ be the difference graph of $\mathbf{A}$ with respect to γ. Let $\beta \in \mathrm{Con}(\mathbf{A})$ be such that $\beta > 0_A$ and $\mathbb{H}_\beta$ has a triangle. Let $\mu \in \mathrm{Con}(\mathbf{A})$ be such that $\mu > 0_A$ and $\mathbb{H}_\mu = (H_\mu, \rho_\mu)$ is maximal with respect to $\preceq$ in*

$$\{\mathbb{H}_\alpha \mid \alpha \in \mathrm{Con}(\mathbf{A}), \alpha > 0_A, \mathbb{H}_\beta \preceq \mathbb{H}_\alpha\}.$$

Let $m \in \mathbb{N}$ and let $t(x, y, z_1, \ldots, z_m)$ be a term in the language of $\mathbf{A}$ such that its induced term function $t^{\mathbf{A}} \in \mathrm{Clo}_{2+m}(\mathbf{A})$ satisfies (4.3)*; such a term exists by Lemma 4.3. Let $\mathbb{G} = (G, \rho^{\mathbb{G}})$ be a graph. We assume that $G \cap H_\mu = \varnothing$. Let Φ be the quasi-identity*

$$\Big(\bigwedge_{(u,v) \in \rho^{\mathbb{G}} \cup \rho_\mu} \big(a = t(x_u, y_u, \boldsymbol{z}_{(u,v)}) \ \wedge \ b = t(x_v, y_v, \boldsymbol{z}_{(u,v)})\big) \Big) \Rightarrow a = b$$

in the variables $\{x_u \mid u \in G \cup H_\mu\} \cup \{y_u \mid u \in G \cup H_\mu\} \cup \{(\boldsymbol{z}_{(u,v)})_i \mid i \in \underline{m}, (u,v) \in \rho^{\mathbb{G}} \cup \rho_\mu\} \cup \{a, b\}$. Then $\mathbb{G} \preceq \mathbb{H}_\mu$ if and only if Φ is not valid in $\mathbf{A}$.

Proof. For the "only if"-direction, let f be a homomorphism from $\mathbb{G}$ to $\mathbb{H}_\mu$. We set the variables in Φ in a way that contradicts the validity of Φ. First, we assign values to a and b such that $(a, b) \in \mu \setminus 0_A$. Next, we assign values to the variables x_u, y_u with $u \in G$. To this end, for each $u \in G$, we set $x_u, y_u \in A$ such that $\binom{x_u}{y_u} = f(u)$. Then for each pair $(u, v) \in \rho^{\mathbb{G}}$, the fact that f is a homomorphism yields $(f(u), f(v)) \in \rho_\mu$, and therefore we have $\mu \leq [\Theta(\binom{x_u}{y_u}, \binom{x_v}{y_v}), 1_A]$. Since $(a, b) \in \mu$, Lemma 4.3 allows us to find $\boldsymbol{z}_{(u,v)} \in A^m$ such that $t(x_u, y_u, \boldsymbol{z}_{(u,v)}) = a$ and $t(x_v, y_v, \boldsymbol{z}_{(u,v)}) = b$. In the next step, we assign values to the variables x_u, y_u with $u \in H_\mu$. To this end, we observe that each $u \in H_\mu$ is an element of A^2, and hence there are $x_u, y_u \in A$ such that $u = \binom{x_u}{y_u}$. Then for each $(u, v) \in \rho_\mu$, we have $(\binom{x_u}{y_u}, \binom{x_v}{y_v}) = (u, v) \in \rho_\mu$. Hence from the definition of ρ_μ, we obtain

$$\mu \leq [\Theta(\tbinom{x_u}{y_u}, \tbinom{x_v}{y_v}), 1_A].$$

Thus from Lemma 4.3 we obtain $\boldsymbol{z}_{(u,v)} \in A^m$ such that $t(x_u, y_u, \boldsymbol{z}_{(u,v)}) = a$ and $t(x_v, y_v, \boldsymbol{z}_{(u,v)}) = b$. Now this assignment of the variables confirms that Φ is not valid in $\mathbf{A}$.

For the "if"-direction, we assume that that the variables in Φ are assigned such that Φ does not hold and we construct a homomorphism $f : \mathbb{G} \to \mathbb{H}_\mu$. Let $\tau := \Theta_{\mathbf{A}}(a, b)$. Since $a \neq b$, we have $\tau > 0_A$. We first define a mapping g from $\mathbb{H}_\mu$ to $\mathbb{H}_\tau$ by

$$g(u) = \tbinom{x_u}{y_u} \text{ for all } u \in H_\mu.$$

Next, we prove that g is a homomorphism from $\mathbb{H}_\mu$ to $\mathbb{H}_\tau$. Since Φ does not hold, its precondition is fulfilled, and therefore for each $(u, v) \in \rho_\mu$, we have $a = t(x_u, y_u, \boldsymbol{z}_{(u,v)})$ and $b = t(x_v, y_v, \boldsymbol{z}_{(u,v)})$. Hence setting $\boldsymbol{a} := \binom{x_u}{y_u}$ and $\boldsymbol{b} := \binom{x_v}{y_v}$ in Lemma 4.2, we obtain $(a, b) \in [\Theta(\binom{x_u}{y_u}, \binom{x_v}{y_v}), 1_A]$, and therefore $\tau \leq [\Theta(\binom{x_u}{y_u}, \binom{x_v}{y_v}), 1_A]$. From the definition of the difference graph $\mathbb{H}_\tau$, we now see that $(\binom{x_u}{y_u}, \binom{x_v}{y_v}) = (g(u), g(v))$ is an edge of $\mathbb{H}_\tau$. Hence g is a homomorphism from $\mathbb{H}_\mu$ to $\mathbb{H}_\tau$, which implies $\mathbb{H}_\mu \preceq \mathbb{H}_\tau$. Since $\mathbb{H}_\beta \preceq \mathbb{H}_\mu$, we have $\mathbb{H}_\beta \preceq \mathbb{H}_\tau$ and thus $\mathbb{H}_\tau \in \{\mathbb{H}_\alpha \mid \alpha \in \mathrm{Con}(\mathbf{A}), \alpha > 0_A, \mathbb{H}_\beta \preceq \mathbb{H}_\alpha\}$. By the maximality of $\mathbb{H}_\mu$ within this set, we therefore have $\mathbb{H}_\tau \preceq \mathbb{H}_\mu$, which yields a graph homomorphism $h : \mathbb{H}_\tau \to \mathbb{H}_\mu$.

Next, we define a homomorphism $j : \mathbb{G} \to \mathbb{H}_\tau$. For $u \in G$, we define $j(u) := \binom{x_u}{y_u}$. Using the same reasoning as for g, we show that j is a homomorphism: assume that $(u, v) \in \rho^{\mathbb{G}}$. Since Φ is not satisfied, we have $t(x_u, y_u, \boldsymbol{z}_{(u,v)}) = a$ and $t(x_v, y_v, \boldsymbol{z}_{(u,v)}) = b$. Hence $(a, b) \in [\Theta(\binom{x_u}{y_u}, \binom{x_v}{y_v}), 1_A]$, which implies that $\tau \leq [\Theta(\binom{x_u}{y_u}, \binom{x_v}{y_v}), 1_A]$. Thus $(j(u), j(v)) = (\binom{x_u}{y_u}, \binom{x_v}{y_v})$ is an edge of $\mathbb{H}_\tau$, and therefore j is a homomorphism from $\mathbb{G}$ to $\mathbb{H}_\tau$.

Now $f := h \circ j$ is the required homomorphism from $\mathbb{G}$ to $\mathbb{H}_\mu$. ◀

We will now prove the main result.

Proof of Theorem 3.1. From Lemma 5.3, we obtain $\beta \in \mathrm{Con}(\mathbf{A})$ such that the difference graph $\mathbb{H}_\beta$ has a triangle. Let $\mathbb{H}_\mu$ be as in the assumptions of Lemma 5.4. Since $\mu > 0_A$, $\mathbb{H}_\mu$ is loopless. Now from $\mathbb{H}_\beta \preceq \mathbb{H}_\mu$, we obtain that $\mathbb{H}_\mu$ contains a triangle. Thus from Corollary 5.1, we obtain that $\mathbb{H}_\mu$-coloring is NP-complete. By Lemma 5.4, the existence of a $\mathbb{H}_\mu$-coloring of a given graph $\mathbb{G}$ can be determined by checking the validity of a quasi-identity Φ that can be computed in time polynomial in the size of $\mathbb{G}$. This implies that QuasiIdVal($\mathbf{A}$) is coNP-complete. ◀

Proof of Corollary 3.2. Given Theorem 3.1, we only have to verify that QuasiIdVal($\mathbf{A}$) is in P when $\mathbf{A}$ is an abelian finite Mal'cev algebra of finite type. In Section 2, we observed that QuasiIdVal($\mathbf{A}$) can be reduced to PolSysSat($\mathbf{A}$) using a truth table reduction. In fact, a counterexample to the validity of a quasi-identity can be found as the solution of one of $|A| \cdot (|A| - 1)$ many polynomial systems: one passes from a quasi-identity such as (2.1) to the systems given in (2.2). From this reduction, we see that it suffices to show that PolSysSat($\mathbf{A}$) is in P when $\mathbf{A}$ is an abelian finite Mal'cev algebra of finite type. Since a Mal'cev algebra generates a congruence modular variety, it follows from [15, Corollary 3.14] (which is proved using [7, Theorem 33]) that in this case PolSysSat($\mathbf{A}$) can be solved in polynomial time. ◀

As special cases, we obtain:

▶ **Corollary 5.5.** *For a finite group* $\mathbf{G}$*,* QuasiIdVal($\mathbf{G}$) *is in P if* $\mathbf{G}$ *is abelian, and coNP-complete otherwise. For a finite ring* $\mathbf{R}$ *(not necessarily commutative and not necessarily with unit),* QuasiIdVal($\mathbf{R}$) *is in P if* $\mathbf{R}$ *is a zero ring, i.e.,* $R \cdot R = 0$*, and coNP-complete otherwise.*

References

1. Erhard Aichinger. The polynomial functions of certain algebras that are simple modulo their center. In *Contributions to general algebra. 17*, pages 9–24. Heyn, Klagenfurt, 2006.
2. Erhard Aichinger, Nebojša Mudrinski, and Jakub Opršal. Complexity of term representations of finitary functions. *Internat. J. Algebra Comput.*, 28(6):1101–1118, 2018. doi:10.1142/S0218196718500480.
3. Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330, 2017. doi:10.1109/FOCS.2017.37.
4. Stanley Burris and John Lawrence. The equivalence problem for finite rings. *J. Symbolic Comput.*, 15(1):67–71, 1993. doi:10.1142/S021819671100625X.
5. Stanley Burris and Hanamantagouda P. Sankappanavar. *A course in universal algebra*. Springer New York Heidelberg Berlin, 1981.
6. Pete L. Clark. The Combinatorial Nullstellensätze revisited. *Electron. J. Combin.*, 21(4):Paper 4.15, 17, 2014. doi:10.37236/4359.

7 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. Comput.*, 28(1):57–104 (electronic), 1999. doi:10.1137/S0097539794266766.

8 Ralph Freese and Ralph N. McKenzie. *Commutator Theory for Congruence Modular varieties*, volume 125 of *London Math. Soc. Lecture Note Ser.* Cambridge University Press, 1987.

9 Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Inform. and Comput.*, 178(1):253–262, 2002. doi:10.1006/inco.2002.3173.

10 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Combin. Theory Ser. B*, 48(1):92–110, 1990. doi:10.1016/0095-8956(90)90132-J.

11 David Hobby and Ralph N. McKenzie. *The structure of finite algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, Providence, RI, 1988. doi:10.1090/conm/076.

12 Gábor Horváth. The complexity of the equivalence and equation solvability problems over nilpotent rings and groups. *Algebra universalis*, 66(4):391–403, 2011. doi:10.1007/s00012-011-0163-y.

13 Gábor Horváth and Csaba Szabó. The extended equivalence and equation solvability problems for groups. *Discrete Mathematics & Theoretical Computer Science*, Vol. 13 no. 4, 2011. doi:10.46298/dmtcs.536.

14 Paweł M. Idziak, Piotr Kawałek, and Jacek Krzaczkowski. Intermediate problems in modular circuits satisfiability. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, pages 578–590. Association for Computing Machinery, 2020. doi:10.1145/3373718.3394780.

15 Benoit Larose and László Zádori. Taylor terms, constraint satisfaction and the complexity of polynomial equations over finite algebras. *Internat. J. Algebra Comput.*, 16(3):563–581, 2006. doi:10.1142/S0218196706003116.

16 Anatoly I. Mal'cev. On the general theory of algebraic systems. *Mat. Sb. N.S.*, 35(77):3–20, 1954.

17 Ralph N. McKenzie, George F. McNulty, and Walter F. Taylor. *Algebras, lattices, varieties, Volume I.* Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, California, 1987.

18 Michael Sipser. *Introduction to the Theory of Computation.* Cengage Learning, Boston, MA, USA, 3rd edition, 2012. URL: https://books.google.at/books?id=1aMKAAAAQBAJ.

19 Guy Terjanian. Sur les corps finis. *C. R. Acad. Sci. Paris Sér. A-B*, 262:A167–A169, 1966.

20 Mikhail V. Volkov. Checking quasi-identities in a finite semigroup may be computationally hard. *Studia Logica*, 78(1-2):349–356, 2004. doi:10.1007/s11225-005-0356-5.

21 Armin Weiß. Hardness of equations over finite solvable groups under the exponential time hypothesis. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 102:1–102:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.102.

22 László Zádori. Solvability of systems of polynomial equations over finite algebras. *Internat. J. Algebra Comput.*, 17(4):821–835, 2007. doi:10.1142/S0218196707003809.

23 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *Journal of the ACM*, 67(5):1–78, 2020. doi:10.1145/3402029.

Packing Odd Walks and Trails in Multiterminal Networks

Maxim Akhmedov[1]
Department of Mathematical Logic and Algorithms, Moscow State University, Russia

Maxim Babenko[1]
Higher School of Economics, Moscow, Russia

Abstract

Let G be an undirected network with a distinguished set of *terminals* $T \subseteq V(G)$ and edge *capacities* $cap : E(G) \to \mathbb{R}_+$. By an *odd* T-walk we mean a walk in G (with possible vertex and edge self-intersections) connecting two distinct terminals and consisting of an odd number of edges. Inspired by the work of Schrijver and Seymour on odd path packing for two terminals, we consider packings of odd T-walks subject to capacities cap.

First, we present a strongly polynomial time algorithm for constructing a maximum fractional packing of odd T-walks. For even integer capacities, our algorithm constructs a packing that is half-integer. Additionally, if $cap(\delta(v))$ is divisible by 4 for any $v \in V(G)-T$, our algorithm constructs an integer packing.

Second, we establish and prove the corresponding min-max relation.

Third, if G is inner Eulerian (i.e. degrees of all nodes in $V(G) - T$ are even) and $cap(e) = 2$ for all $e \in E$, we show that there exists an integer packing of odd T-trails (i.e. odd T-walks with no repeated edges) of the same value as in case of odd T-walks, and this packing can be found in polynomial time.

To achieve the above goals, we establish a connection between packings of odd T-walks and T-trails and certain multiflow problems in undirected and bidirected graphs.

2012 ACM Subject Classification Theory of computation → Network optimization

Keywords and phrases Odd path, signed and bidirected graph, multiflow, polynomial algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.5

1 Introduction

Hereinafter, for graph G we use notation $V(G)$ (resp. $E(G)$) to denote the set of vertices (resp. edges) of G.

Consider an undirected network G with a distinguished set of *terminals* $T \subseteq V(G)$ and edge capacities $cap : E(G) \to \mathbb{R}_+$. We use the notions of **walks** and **paths**; the former allow arbitrary edge and vertex self-intersections, while the latter forbid any self-intersections. Additionally, we consider **trails** that allow vertex self-intersections but not edge self-intersections. Any path is a trail and any trail is a walk, but not vice versa. A ***T*-walk** (resp. ***T*-trail** or ***T*-path**) is a walk (resp. trail or path) connecting two distinct vertices in T (note that its intermediate vertices may also be in T).

By a (fractional) **packing** of T-walks (resp. T-trails, T-paths) subject to capacities cap we mean a weighted collection $\mathcal{P} = \{\alpha_1 \cdot W_1, \ldots, \alpha_m \cdot W_m\}$, where W_i are T-walks (resp. T-trails, T-paths) and $\alpha_i \in \mathbb{R}_+$ are **weights** such that $\sum_i \alpha_i n_i(e) \le cap(e)$ for any $e \in E(G)$, where $n_i(e) = 0, 1, \ldots$ denotes the number of occurrences of e in W_i. If all α_i are integer (resp. $\frac{1}{k}$-integer, i.e. become integer after multiplying by k) then the whole packing is said to be **integer** (resp. **$\frac{1}{k}$-integer**). The **value** of $\mathcal{P}$ (denoted by $\|\mathcal{P}\|$) is $\sum_i \alpha_i$; a packing of maximum value will be referred to as **maximum**.

[1] Corresponding author

© Maxim Akhmedov and Maxim Babenko;
licensed under Creative Commons License CC-BY 4.0

If one imposes no additional restrictions, the values of maximum packings of T-walks, T-trails and T-paths coincide; this follows from the fact that any walk can be reduced into a path by removing its cyclic parts. Also for $|T| = 2$ and integer edge capacities the value of a maximum fractional packing equals the value of a maximum integer packing (by the max-flow integrality theorem [10, Cor. 10.3a]), while for $|T| \geq 3$ a maximum packing may be half-integer [10, Sec. 73.2].

Now consider a much harder case where T-walks (resp. T-trails, T-paths) comprising a packing are required to be **odd**, i.e. to consist of an odd number of edges. Now an attempt to transform a walk into a path (or even a walk into a trail) by a similar decycling approach fails since it may alter the parity.

Our original source of inspiration lies in the work of Schrijver and Seymour [11], who established a min-max formula for the value of a maximum fractional packing of odd T-paths for $T = \{s, t\}$. At its dual side, the formula involves enumerating (not necessarily induced) subgraphs H of G that contain both s and t but no odd $s - t$ path and upper-bounding the value of packings by a certain "capacity" of H. In a sense, such H is analogous to an $s - t$ cut in the standard max-flow-min-cut-theorem [10, Th. 10.3] and is called an **odd path barrier**.

The above result is established for just $|T| = 2$, only concerns fractional packings and, moreover, is non-constructive. In case of integer capacities one should ultimately aim for a min-max formula and a polynomial algorithm for constructing a maximum integer packing of odd T-paths. These questions, unfortunately, seem to be notoriously hard. In particular, [12, Sec. 3.3] shows that checking if a given graph contains a pair of edge-disjoint odd T-trails is NP-hard already for $|T| = 2$.

1.1 Our results

The present paper deals with the **multiterminal** version of the problem (allowing arbitrary number of terminals T) but considers packings of odd T-walks and T-trails rather than odd T-paths.

First, for packings of odd T-walks and real-valued capacities we present a polynomial time reduction (Theorem 9) from a maximum odd T-walk packing problem to a maximum multiflow problem for a special commodity graph family due to Karzanov [6]. For even integer capacities, our algorithm produces a half-integer packing. Also, if capacities are even integers and $cap(\delta(v))$ (which is, as usual, the sum of $cap(e)$ for all edges e incident to v) is divisible by 4 for any $v \in V(G) - T$, a maximum packing can be made integer.

Second, we present a min-max formula (Theorem 12) for maximum odd T-walk packings. It is strikingly similar to the one due to Schrijver and Seymour [11] (for odd $s - t$ paths) and involves, at its dual side, subgraphs of G containing no odd T-walks.

Third, we extend the above results to odd T-trail packings. Consider the unit-capacity case. Then a Schrijver–Seymour-type min-max relation does not hold for integer packings even if one assumes that $|T| = 2$ and the underlying graph is **inner Eulerian** (i.e. degrees of all non-terminal verticies are even). An example of such a "bad" instance can be found in [11, Sec.3]. (There it was given for the case of odd T-paths rather than odd T-trails but it turns the example works in both cases.)

The fractionality status of such a packing problem seems to be open. We partially resolve it by proving that for an inner Eulerian graph with unit capacities and an arbitrary number of terminals an optimum packing of T-trails can always be chosen half-integer (and also can be found in polynomial time). If all capacities are multiplied by 2, the inner Eulerianness condition becomes $cap(\delta(v))$ being divisible by 4 for all $v \in V(G) - T$, which is equivalent to the condition from the first result, and our optimum packing becomes integer.

We prove (Theorem 14) that there exists a packing of odd T-trails of the same value as in the case of odd T-walks, and this packing can be found in polynomial time. In other words, odd T-walks forming a maximum integer packing can always be rearranged ("untangled") to ensure that none of them has edge self-intersections.

1.2 Our techniques

The algorithm that deals with odd T-walks is based on a reduction to a certain multiflow problem [6] and some graph symmetrization. For the min-max formula regarding packing of odd T-walks, we indicate how optimum collections of cuts (in the sense of the above multiflow problem) correspond to minimum odd T-walk barriers.

The algorithm dealing with odd T-trails attracts additional combinatorial ideas. Loosely speaking, it constructs a maximum integer packing consisting of odd T-walks W_i. If all of these T-walks W_i are already T-trails (i.e. have no edge self-intersections), then we are done. Otherwise, for walk W_i with edge self-intersections, we either simplify W_i (while maintaining its parity) or find a **redundant** edge in G whose removal does not decrease the number of T-walks in the current packing, drop it, and repeat. The existence of a redundant edge is proved by a novel characterization of integer odd T-walk packings in terms of T-trail packings in inner Eulerian **bidirected** graphs [1] and relies on the corresponding min-max theorem.

1.3 Related work

There is also a solid body of recent research devoted to path packings in unit-capacitated graphs. (Note that here the notions of integer walk and trail packings coincide.)

While even for $T = \{s, t\}$ the problem of finding a maximum integer packing of odd T-trails in general networks does not seem to be tractable, a certain lower bound for the maximum value of such packings (relating it to odd T-trail covers) is known [5] (also see [4] for a weaker bound).

Note that if one is interested in packing odd T-trails (rather than odd T-walks) in graphs with integer capacities larger than 1, then the problem does not seem to be directly reducible to unit capacities. Indeed, splitting each edge and solving the problem in the unit-capacitated case, one will face challenges with edge self-intersections when attempting to return back to the original graph.

These challenges seem to be quite fundamental, and, in particular, we are not aware of any prior art concerning capacitated versions of the maximum odd T-trail integral packing problem. Our algorithm for constructing a maximum packing of odd T-trails is able to deal with edge self-intersections by certain T-walk "untangling" but this battle is not won easily.

Another related (but still substantially different) area of research concerns integer packing of vertex-disjoint A-paths in **group-labeled** graphs. Here each edge $xy \in E$ is endowed with an element $g(x, y)$ of group Γ (obeying $g(x, y) = -g(y, x)$). Path P with both (distinct) ends in $A \subseteq V(G)$ is called a **non-zero A-path** if the sum of all group elements corresponding to (directed) edges of P is non-zero. (This also extends to non-Abelian groups.) In [3] a polynomial algorithm for constructing a maximum integer packing of vertex-disjoint A-paths is given. See also [9] for a similar treatment involving permutation groups.

With an appropriate choice of group Γ and edge labels, non-zero A-paths may express various well-studied notions, e.g. the much-celebrated Mader's integer packings of vertex-disjoint $\mathcal{S}$-paths [8], [10, Sec. 73.1].

Note that if $\Gamma = \mathbb{Z}_2$ and $g(x, y) = 1$ for all edges xy one gets the odd parity constraint for paths comprising a packing. The latter motivates adding such Γ as a direct group summand in the Mader's case above hoping to capture the parity restriction. This approach, however, will not work as expected: now a path could either be odd *or* connect terminals in distinct $\mathcal{S}$-classes (while we were certainly hoping for paths that simultaneously have ends in distinct $\mathcal{S}$-classes *and* have odd length).

2 Walks, trails, packings and other notation

Consider an undirected loopless graph G with possible parallel edges. In this paper we deal with certain families of path-like objects in G differing in kinds of allowed self-intersections. Formally:

▶ **Definition 1.** *Given $x, y \in V(G)$, an $x - y$ **walk** is a sequence $W = (e_1, e_2, \dots, e_l)$, where $e_i \in E$ are such that $e_i = v_{i-1}v_i$ for $v_0 = x$, $v_l = y$ and $v_1, \dots, v_{l-1} \in V(G)$.*

*Here l is called the **length** of W. A walk is called **even** or **odd** depending on the parity of its length. Vertices x and y are called the **endpoints** of W and $v_1, \dots, v_{l-1}$ are called **intermediate** (for W).*

Note that some of vertices v_i of W may coincide, allowing a walk to visit the same vertex multiple times and traverse same edge multiple times.

▶ **Definition 2.** *An $x-y$ **trail** (resp. **path**) is an $x-y$ walk W with all edges (resp. vertices) being distinct. An $x - x$ walk (resp. trail) is called **cyclic**.*

▶ **Definition 3.** *Let $T \subseteq V(G)$ be a distinguished set of vertices called **terminals**. A **T-walk** (resp. **T-trail**, **T-path**) is an $x - y$ walk (resp. $x - y$ trail, $x - y$ path) for two distinct $x, y \in T$. (Note that unless explicitly stated otherwise, intermediate vertices of such walks are allowed to be terminals.)*

▶ **Definition 4.** *Graph G is called **inner Eulerian with respect to T** (or simply **inner Eulerian** if T is clear from context) if for any $v \in V(G) - T$ the degree of v in G is even.*

▶ **Definition 5.** *Given edge capacities $cap : E(G) \to \mathbb{R}_+$, a weighted multiset $\mathcal{P} = \{\alpha_1 \cdot W_1, \dots, \alpha_m \cdot W_m\}$, where $\alpha_i \in \mathbb{R}_+$ are **weights** and each W_i is a walk, is said to be a (fractional) **walk packing** if for any $e \in E(G)$ the **load** $\mathcal{P}(e) := \sum_i \alpha_i n_i(e)$ of edge e does not exceed $cap(e)$, where $n_i(e) = 0, 1, \dots$ is the number of occurrences of e in W_i.*

*$\|\mathcal{P}\| := \sum_i \alpha_i$ is called the **value** of $\mathcal{P}$. If all $\alpha_i \in \mathbb{Z}_+$ then $\mathcal{P}$ is called **integer**.*

If $\mathcal{P}, \mathcal{Q}$ are packings and $\alpha \in \mathbb{R}_+$, $\mathcal{P} + \mathcal{Q}$ denotes a union of weighted multisets and $\alpha \cdot \mathcal{P}$ denotes the result of multiplying all weights in $\mathcal{P}$ by α.

When walks comprising a packing are restricted in some way, the analogous terminology is applied to the packing as a whole. In particular, one may speak of T-walk (resp. T-trail, T-path) packings $\mathcal{P}$ indicating that walks in $\mathcal{P}$ are, in fact, T-walks (resp. T-trails, T-paths).

▶ **Definition 6.** *A triple (G, T, cap) consisting of an undirected graph G, terminal set $T \subseteq V(G)$ and capacity function $cap : E(G) \to \mathbb{R}_+$, is called a **network**.*

Two notable special cases of constant capacity function to appear throughout our paper are $\mathbf{1}(e) := 1$ and $\mathbf{2}(e) := 2$ for any $e \in E(G)$.

▶ **Definition 7.** *Consider network $N = (G, T, cap)$ together with undirected graph H such that $V(H) = T$ (called the **commodity graph**). A **multi-commodity flow** (or simply a **multiflow**) in network N with commodity graph H is a T-walk packing $\mathcal{P}$ such that for any T-walk W in $\mathcal{P}$ its (distinct) endpoints are connected by an edge in H.*

We also employ the following graph-theoretic notation:

▶ **Definition 8.**

- *Given graph G and $A \subseteq V(G), v \in V(G)$, $\delta(v)$ denotes the set of edges incident to v, $\delta(A)$ denotes the set of edges with exactly one endpoint in A and $\gamma(A)$ denotes the set of edges with both endpoints in A;*
- *For function $f : X \to \mathbb{R}$ and $Y \subseteq X$, $f(Y)$ is defined as $\sum_{x \in Y} f(x)$; e.g. for a set of vertices A, $f(\delta(A))$ is the total value of f over all edges with exactly one endpoint in A;*
- *Given edge capacities $cap\colon E(G) \to \mathbb{R}_+$ in graph G and $S, T \subseteq V(G), S \cap T = \varnothing$, an $S - T$ **cut** is a vertex set C such that $S \subseteq C \subseteq V(G) - T$; the **capacity** of cut C is $cap(\delta(C))$; the minimum capacity of an $S - T$ cut is denoted by $\lambda(S, T)$;*
- *When graph G is not clear from the context, it is specified explicitly, e.g. δ_G, γ_G and λ_G.*

3 Odd T-walk packing algorithm

Let (G, T, cap) be a network. In this section we introduce an auxiliary network $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ constructed from G and employ it to provide a strongly polynomial time algorithm for finding a maximum odd T-walk packing. This network also plays a crucial role in further sections.

Construct graph $\widetilde{G}$ with $V(\widetilde{G}) := V(G) \sqcup V(G)'$, where $V(G)'$ is a disjoint copy of $V(G)$, i.e. each vertex $v \in V(G)$ has its own copy $v' \in V(G)'$, and $E(\widetilde{G}) := \{u'v, uv' \mid uv \in E(G)\}$. Also, let $v'' := v$ for $v \in V(G)$. If $x, y \in V(G)$, vertices x and x' are called **symmetric** to each other, and similarly for edges xy' and $x'y$. For a vertex set (resp. an edge set or a walk) X, let X' be the vertex set (resp. edge set or walk) consisting of vertices (or edges) symmetric to ones in X. Let $\widetilde{T} := T \sqcup T'$. Finally, define capacities on edges of $\widetilde{G}$ as $\widetilde{cap}(uv') := \widetilde{cap}(u'v) := \frac{1}{2} cap(uv)$ for any $uv \in E(G)$.

The following theorem encapsulates the first of our results announced in Section 1.

▶ **Theorem 9** (Odd T-walk packing)**.** *Given network (G, T, cap), it is possible to construct a maximum fractional odd T-walk packing $\mathcal{P}$ in (G, T, cap) in strongly polynomial time.*

If all capacities are non-negative even integers, the resulting $\mathcal{P}$ is half-integer. If additionally $cap(\delta(v))$ is divisible by 4 for all $v \in V(G) - T$, $\mathcal{P}$ is integer.

Proof. Note that $\widetilde{G}$ is bipartite, so for distinct $x, y \in T$ any $x - y'$ walk in $\widetilde{G}$ is odd and corresponds to an odd $x - y$ walk in G.

Construct commodity graph H_T as follows: $V(H_T) := \widetilde{T}$ and $E(H_T) := \{t_i t'_j \mid i \neq j\}$. Note that H_T is isomorphic to $K_{|T|,|T|}$ without a perfect matching (see Figure 1).

Consider an arbitrary fractional odd T-walk packing $\mathcal{P} = \{\alpha_1 \cdot W_1, \ldots, \alpha_m \cdot W_m\}$ in (G, T, cap) of value p. Denote endpoints of W_k as $t_{k,1}$ and $t_{k,2}$; this walk corresponds to a pair of $t_{k,1} - t'_{k,2}$ walk $\widetilde{W}_k$ and $t'_{k,1} - t_{k,2}$ walk $\widetilde{W}'_k$ in $\widetilde{G}$, which are symmetric to each other. Packing $\widetilde{\mathcal{P}} := \{\alpha_1 \cdot \widetilde{W}_1, \ldots, \alpha_m \cdot \widetilde{W}_m\}$ in $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ is of value p. For any $xy \in E(G)$ holds $\widetilde{\mathcal{P}}(xy') + \widetilde{\mathcal{P}}(x'y) = \mathcal{P}(xy)$ since each occurrence of xy in some walk W_k in $\mathcal{P}$ corresponds to exactly one occurrence of either $x'y$ or xy' in $\widetilde{W}_k$. The same properties hold for packing $\widetilde{\mathcal{P}}' = \{\alpha_1 \cdot \widetilde{W}'_1, \ldots, \alpha_m \cdot \widetilde{W}'_m\}$, which is the symmetric counterpart of $\widetilde{\mathcal{P}}$.

Construct $\mathcal{Q} := \frac{1}{2}(\widetilde{\mathcal{P}} + \widetilde{\mathcal{P}}')$; the value of $\mathcal{Q}$ is also p. For any $xy \in E(G)$ holds $\widetilde{\mathcal{P}}'(xy') = \widetilde{\mathcal{P}}(x'y)$, thus $\mathcal{Q}(xy') = \frac{1}{2}(\widetilde{\mathcal{P}}(xy') + \widetilde{\mathcal{P}}'(xy')) = \frac{1}{2}(\widetilde{\mathcal{P}}(xy') + \widetilde{\mathcal{P}}(x'y)) = \frac{1}{2}\mathcal{P}(xy) \leq \frac{1}{2} cap(xy) = \widetilde{cap}(xy')$, i.e. $\mathcal{Q}$ is a (self-symmetric) multiflow in $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ with commodity graph H_T. Thus p does not exceed the value of a maximum fractional multiflow in $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ with commodity graph H_T.

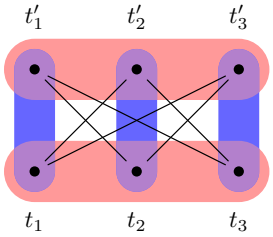


Figure 1 Commodity graph H_T; anticlique family $\mathcal{A}_1$ is in red and $\mathcal{A}_2$ is in blue.

Conversely, consider a fractional multiflow $\mathcal{Q}$ of value q in network $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ with commodity graph H_T. Construct an odd T-walk packing $\mathcal{P}$ of value q in (G, T, cap) by taking preimages of all weighted walks in $\mathcal{Q}$ with their respective weights. Clearly, for $xy \in E(G)$ holds $\mathcal{P}(xy) = \mathcal{Q}(xy') + \mathcal{Q}(x'y) \leq \widetilde{cap}(xy') + \widetilde{cap}(x'y) = cap(xy)$. Thus, q does not exceed the value of a maximum fractional odd T-walk packing in (G, T, cap).

Therefore, the maximum value of a fractional odd T-walk packing in (G, T, cap) equals the value of a maximum fractional multiflow in $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ with commodity graph H_T. To conclude the proof, we utilize the following result due to Karzanov [6]:

▶ **Theorem 10.** *Let (G, T, cap) be a network with commodity graph H. Denote by $\mathcal{A}$ the family of all inclusion-wise maximal anticliques (i.e. independent sets) in H. Suppose $\mathcal{A}$ can be split into two subfamilies $\mathcal{A}_1, \mathcal{A}_2$ such that all anticliques in each family $\mathcal{A}_i$ are pairwise disjoint.*

Then a maximum multiflow in (G, T, cap) can be found in strongly polynomial time. If, additionally, cap are integers and $cap(\delta(v))$ is even for any $v \in V(G) - T$, then the resulting multiflow is integer.

Note that the family of anticliques in H_T obeys the property from Theorem 10. Indeed, define $\mathcal{A}_1 := \{T, T'\}$ and $\mathcal{A}_2 := \{\{t, t'\} \mid t \in T\}$ (see Figure 1). If some maximal anticlique contains a terminal and its symmetric copy, then it must belong to $\mathcal{A}_2$; otherwise it cannot contain both a vertex from T and a vertex from T', thus it belongs to $\mathcal{A}_1$. Also, if $cap(\delta(v))$ is divisble by 4 for any $v \in V(G) - T$, then $\widetilde{cap}(\delta(v))$ is even for any $v \in V(\widetilde{G}) - \widetilde{T}$. Therefore, applying Theorem 10 finishes the proof. ◀

4 Odd T-walk barrier

In this section we provide a combinatorial description of barrier structure that defines a tight upper bound for the value of a maximum odd T-walk packing, which is our second result announced in Section 1. This characterization is surprisingly similar to the corresponding barrier structure for maximum odd $s - t$ path packings due to Schrijver and Seymour [11]. A strong duality is proven using the equivalence with multiflows from Section 3.

▶ **Definition 11.** *Given network (G, T, cap), a (not necessarily induced) subgraph B of G with $T \subseteq V(B)$ is called an **odd T-walk barrier** if there is no odd T-walk in B.*

*The **capacity** $cap(B)$ of barrier B is defined as $\frac{1}{2}cap(I(B)) + cap(U(B))$, where for an arbitrary (not necessarily induced) subgraph H of G we use the following notation:*

- *$I(H) := \{xy \in E(G) \mid xy \in \delta_G(V(H))\}$ (informally, the edge leaves H and does not return);*
- *$U(H) := \{xy \in E(G) \mid x, y \in V(H), xy \in E(G) - E(H)\}$ (informally, the edge takes a U-turn by leaving H and immediately returning back).*

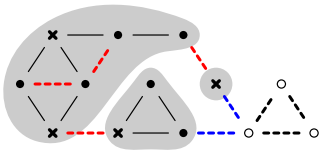


Figure 2 An example of an odd T-walk barrier B; vertices in T are crosses, vertices in $V(B) - T$ are black dots, vertices not in $V(B)$ are white dots; edges in $E(B)$ are solid, edges not in $E(B)$ are dashed; edges in $I(B)$ are blue, edges in $U(B)$ are red.

It is easy to verify that the capacity of any barrier B is an upper bound for the value of any odd T-walk packing $\mathcal{P}$. Indeed, any odd T-walk W endowed with weight α in $\mathcal{P}$ is not entirely contained in B; thus it either visits some vertex $v \in V(G) - V(B)$ or traverses some edge $xy \in E(G) - E(B)$ such that $x, y \in V(B)$. In the former case it reserves α units of capacity of at least two edges in $I(B)$, and in the latter case it reserves α units of capacity of at least one edge in $U(B)$. Therefore $\|\mathcal{P}\| \leq cap(B)$. The strong duality also holds:

▶ **Theorem 12** (see Appendix). *Let (G, T, cap) be a network. If $\mathcal{P}$ ranges over odd T-walk packings and B ranges over odd T-walk barriers, then $\max_{\mathcal{P}} \|\mathcal{P}\| = \min_{B} cap(B)$.*

The min-max formula above enables strengthening the statement of Theorem 9 as follows.

▶ **Corollary 13.** *Given network (G, T, cap), let $\mathcal{P}$ be a maximum fractional odd T-walk packing in (G, T, cap). If all capacities are non-negative even integers, $\|\mathcal{P}\|$ is integer. If additionally $cap(\delta(v))$ is divisible by 4 for all $v \in V(G) - T$, $\|\mathcal{P}\|$ is even integer.*

Proof. By Theorem 12, $\|\mathcal{P}\| = \frac{1}{2} cap(I(B)) + cap(U(B))$ for minimum odd T-walk barrier B. If all capacities are even integers, then $cap(U(B))$ and $cap(I(B))$ are also even, therefore the first part of the statement is trivial. Let $A := V(G) - V(B)$, note that $\delta(A) = I(B)$. Under the second condition, note the following congruence:

$$0 \equiv \sum_{v \in A} cap(\delta(v)) \equiv 2cap(\gamma(A)) + cap(\delta(A)) \equiv cap(I(B)) \pmod 4$$

Therefore, $\frac{1}{2} cap(I(B))$ is also an even integer. ◀

5 Odd T-trail packing algorithm

Hereinafter we focus on network $(G, T, \mathbf{2})$ for an inner Eulerian graph G. Since all capacities are 2, each edge can be traversed by at most two walks in an integer packing. Our ultimate goal is to construct a maximum integer T-trail packing. The third result announced in Section 1 is as follows:

▶ **Theorem 14.** *Given network $(G, T, \mathbf{2})$ with inner Eulerian G, it is possible to construct a maximum integer packing of odd T-trails in polynomial time. This packing is also a maximum fractional packing of odd T-walks in $(G, T, \mathbf{2})$.*

We use a chemistry-inspired notation: replace each edge in G with two **valencies** each of which may be occupied by a walk. More formally:

▶ **Definition 15.** *For edge $e \in E(G)$, denote e^1 and e^2 to be two **valencies** of e; edge e is called **underlying** for e^1 and e^2. Define the **valence graph** G^{12} to be the graph on the same vertices as G with valencies regarded as edges.*

In what follows, instead of integer odd T-walk packings in $(G, T, \mathbf{2})$ we shall be dealing with integer odd T-trail packings in $(G^{12}, T, \mathbf{1})$, which effectively are sets of edge-disjoint odd T-trails in G^{12}. However, a T-trail in G^{12} may correspond to a non edge-simple T-walk in G once we replace valencies with their underlying edges. This is captured as follows:

▶ **Definition 16.** *Edge $e \in E(G)$ is called **irregular** for T-trail W in G^{12} if W traverses both valencies e^1, e^2 and **regular** otherwise.*

Hence we are looking for a maximum set of edge-disjoint odd T-trails in G^{12} without irregular edges.

A brief outline of our approach is as follows. In Section 5.1 we introduce **signing** on valencies that guide T-trails and ensure they have proper parities. Given a suitable signing, we prove the existence of an integer odd T-trail packing in $(G^{12}, T, \mathbf{1})$ (with possible irregular edges) of the needed value by reduction to **bidirected networks**. We also prove that a suitable signing exists.

In Section 5.2 we construct such a signing and also perform the so-called **terminal evacuation** by introducing an auxiliary terminal t' for each $t \in T$ that is connected to t with a proper number of valencies of certain signs. This transformation allows to assume that no trail in the packing contains any terminal as its intermediate vertex.

Next, Section 5.3 ensures that inner vertices are of degree at most 3.

Finally in Section 5.4 we deal with irregular edges. We show that whenever both valencies e^1 and e^2 of some edge e are used by some odd T-trail W in the packing, either W could be simplified (preserving its parity) or e is in fact **redundant**, i.e. G can be reduced by dropping e. This reduction preserves the needed properties of G; hence one can recompute a packing and iterate. These iterations continue until there are no more remaining irregular edges.

5.1 Signed graphs

We also use the framework of **signed graphs** whose edges are endowed with signs "+" and "−". Intuitively, a signed valence graph introduces a convenient family of T-trails defined by the requirement of alternation which makes parity of the trail uniquely determined by the signs of the first and the last valence.

▶ **Definition 17.** *A **signing** is an arbitrary function $M : E(G^{12}) \to \{+, -\}$. Graph G^{12} together with some signing M forms a **signed valence graph** (G^{12}, M). In presence of terminal set $T \subseteq V(G)$, a **signed valence network** $(G^{12}, M, T, \mathbf{1})$ appears. A T-trail W in (G^{12}, M) is called **alternating** if signs of valencies alternate along W.*

▶ **Definition 18.** *Signing M is called **inner balanced** if for any $v \in V(G) - T$, the number of positive edges incident to v equals the number of negative edges incident to v.*

We shall need the notion of **bidirected graphs**, which generalize digraphs and admit three possible kinds of edges: a usual **directed edge** (ingoing for one endpoint and outgoing for another), a **positive edge** (which is ingoing for both its endpoints) and a **negative edge** (which is outgoing for both its endpoints). The definition of a bidirected walk or a bidirected trail is similar to Definition 1 with the only difference that for any internal vertex v_i exactly one of e_i, e_{i+1} is ingoing to v_i and another is outgoing from v_i. Refer to [10, Ch. 36] for details.

The notion of inner Eulerianness is extended to bidirected graphs as follows: a bidirected graph G is **inner Eulerian** with respect to terminal set T if for any $v \in V(G) - T$ the number of edges ingoing to v is equal to the number of edges outgoing from v. Similarly to the undirected case, triple (G, T, cap) consisting of bidirected graph G, terminal set T and capacity function cap is called a **bidirected network**.

We rely on two theorems of a similar kind, one of which is due to Cherkassky [2] and Lovász [7], and another is due to Babenko and Karzanov [1, Th. 1.1]:

▶ **Theorem 19** (Min-max formula for T-trail packings in inner Eulerian undirected graphs [2, 7])**.** *Let $(G, T, \mathbf{1})$ be an inner Eulerian network. Then the value of a maximum packing of T-trails equals $\frac{1}{2}\sum_{t\in T}\lambda(\{t\}, T - \{t\})$. Such a packing can be chosen integer and can be constructed in polynomial time.*

▶ **Theorem 20** (Min-max formula for T-trail packings in inner Eulerian bidirected graphs [1])**.** *Let $(G, T, \mathbf{1})$ be an inner Eulerian bidirected network. Then the value of a maximum packing of bidirected T-trails equals $\frac{1}{2}\sum_{t\in T}\lambda(\{t\}, T - \{t\})$. Such a packing can be chosen integer and can be constructed in polynomial time.*

Note that the value of a maximum packing in the latter theorem does not depend on actual directions of edges. As we mentioned before, signings encode a certain family of odd T-walks. The following theorem describes why it is important for us.

▶ **Theorem 21.** *Let $(G^{12}, M, T, \mathbf{1})$ be a signed valence network with an inner balanced signing M. Let $\mathcal{P}$ be a maximum packing of odd T-trails in $(G^{12}, T, \mathbf{1})$, and $\mathcal{S}$ be a maximum packing of alternating T-trails in $(G^{12}, M, T, \mathbf{1})$. Then $\|\mathcal{P}\| = \|\mathcal{S}\|$. Additionally, $\mathcal{P}$ and $\mathcal{S}$ can be chosen integer and can be constructed in polynomial time.*

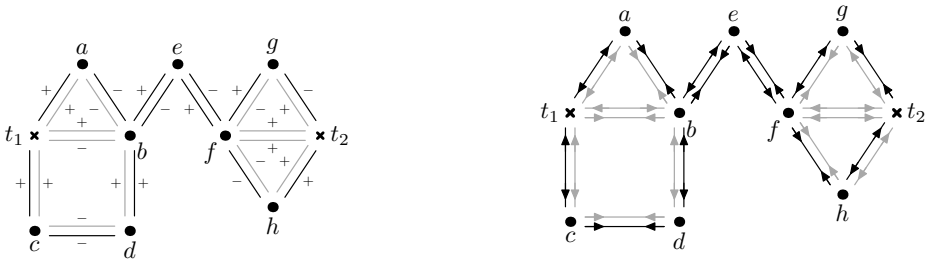
Proof. Construct an auxiliary bidirected graph $\overleftrightarrow{G^{12}}$ corresponding to the signed valence graph (G^{12}, M) as follows: edges in $\overleftrightarrow{G^{12}}$ correspond to valences in $E(G^{12})$; an edge is positive if the sign of the valence is "+" and negative otherwise; M being inner balanced implies that $\overleftrightarrow{G^{12}}$ is inner Eulerian, therefore Theorem 20 is applicable to $(\overleftrightarrow{G^{12}}, T, \mathbf{1})$. Also note that bidirected T-trails in $\overleftrightarrow{G^{12}}$ correspond to alternating T-trails in (G^{12}, M). Refer to Figures 3a and 3b for an example.

Note that G^{12} is automatically inner Eulerian due to each vertex in $V(G^{12})$ being adjacent to an even number of valencies, therefore Theorem 19 is applicable to $(G^{12}, T, \mathbf{1})$.

It follows that maximum packing $\mathcal{P}$ of odd T-trails in $(G^{12}, T, \mathbf{1})$ and maximum packing $\overleftrightarrow{\mathcal{P}}$ of bidirected T-trails in $(\overleftrightarrow{G^{12}}, T, \mathbf{1})$ are of the same value $\frac{1}{2}\sum_{t\in T}\lambda(\{t\}, T - \{t\})$. The latter packing can be chosen integer and can be constructed in polynomial time, and then transformed into a maximum integer packing $\mathcal{S}$ of alternating T-trails in signed valence network $(G^{12}, M, T, \mathbf{1})$. ◀

▶ **Definition 22.** *A signed valence network $(G^{12}, M, T, \mathbf{1})$ with inner balanced signing M is **(p, q)-tight** if 1) there exists an integer T-trail packing in $(G^{12}, T, \mathbf{1})$ of value $p + q$ and 2) the number of "−" valencies adjacent to terminals T is q.*

▶ **Lemma 23.** *Given a signed valence network $(G^{12}, M, T, \mathbf{1})$ with a (p, q)-tight inner balanced signing M, there exists an integer packing $\mathcal{P} + \mathcal{Q}$ in $(G^{12}, M, T, \mathbf{1})$ of value $p + q$, where $\mathcal{P}$ consists of at least p odd alternating T-trails and $\mathcal{Q}$ consists of at most q even alternating T-trails. Moreover, T-trails in $\mathcal{P} + \mathcal{Q}$ can be chosen so as to avoid passing through terminals T as intermediate vertices.*



(a) Packing of two $t_1 - t_2$ trails in $(G^{12}, M, T, \mathbf{1})$, one of which is even and another is odd.

(b) Packing of two bidirected $t_1 - t_2$ trails in $(\overleftrightarrow{G^{12}}, T, \mathbf{1})$, one of which is even and another is odd.

Figure 3 Correspondence between signed valence graph with inner balanced signing $(G^{12}, M, T, \mathbf{1})$ and inner Eulerian bidirected graph $(\overleftrightarrow{G^{12}}, T, \mathbf{1})$. Terminals are crosses, other vertices are black dots.

Proof. The first tightness property implies existence of an integer T-trail packing in $(G^{12}, T, \mathbf{1})$ of value $p + q$. Then, by Theorem 21 we get a packing of $p + q$ alternating T-trails in $(G^{12}, M, T, \mathbf{1})$. Break this packing into two parts $\mathcal{P}$ and $\mathcal{Q}$, where $\mathcal{P}$ consists of odd T-trails and $\mathcal{Q}$ consists of even T-trails.

The second tightness property implies $\|\mathcal{Q}\| \leq q$ as each trail in $\mathcal{Q}$ has a $-$ valence incident to a terminal, therefore $\|\mathcal{P}\| \geq p$, as needed.

W.l.o.g. all these T-trails do not contain terminals as intermediate vertices (for otherwise, if some alternating T-trail W visits $t \in T$ as its intermediate vertex, then W can be split into two subtrails W_1, W_2 at t; among W_1, W_2 at least one, say W_1 is a valid alternating T-trail; replace W with W_1 and repeat). ◀

5.2 Initial signing and terminal evacuation

In this section we present an algorithm for constructing a tight inner balanced signing M. This is done with the help of network $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ from Section 3. Note that since $cap = \mathbf{2}$, we have $\widetilde{cap} = \mathbf{1}$. Degrees of vertices in $\widetilde{G}$ coincide with degrees of their pre-images in G, therefore $\widetilde{G}$ is also inner Eulerian.

Consider a maximum multiflow $\mathcal{F}$ in $(\widetilde{G}, \widetilde{T}, \mathbf{1})$ with commodity graph H_T. Theorem 10 ensures that $\mathcal{F}$ can be chosen integer, i.e. $\mathcal{F}$ is a collection of edge-disjoint $\widetilde{T}$-trails (endowed with weight 1) in $\widetilde{G}$ connecting vertex pairs of form $t_1 - t_2'$ for distinct $t_1, t_2 \in T$. Each of these T-trails is odd, therefore their pre-images are odd T-trails in G^{12} (see Figure 4). Denote the packing of these odd T-trails in $(G^{12}, T, \mathbf{1})$ (taken with weight 1) as $\mathcal{P}$. Let $p := \|\mathcal{P}\|$. The proof of Theorem 9 implies:

▶ **Corollary 24.** *$\mathcal{P}$ is a maximum odd T-trail packing in $(G^{12}, T, \mathbf{1})$.*

Consider the subgraph $\widetilde{Z}$ of $\widetilde{G}$ consisting of edges not appearing in T-trails of $\mathcal{F}$. Since any vertex $v \in V(\widetilde{G}) - \widetilde{T}$ has even degree in $\widetilde{G}$, $\widetilde{Z}$ is also inner Eulerian with respect to terminals $\widetilde{T}$. Therefore $\widetilde{Z}$ decomposes into two families of edge-disjoint trails: a collection of cyclic trails and a collection of $\widetilde{T}$-trails.

The former ones correspond to even cyclic trails in G^{12} (due to biparticity of $\widetilde{G}$); denote the packing (with unit weights) of these even cyclic trails in $(G^{12}, T, \mathbf{1})$ as $\mathcal{E}$.

The latter ones may be further subdivided into two categories: (i) $t_1 - t_2$ or $t_1' - t_2'$ trails for distinct $t_1, t_2 \in T$; and (ii) $t - t'$ trails for $t \in T$. (Note that $t_1 - t_2'$ trails for $t_1 \neq t_2$ cannot appear due to maximality of $\mathcal{F}$.) The first category corresponds to even T-trails in

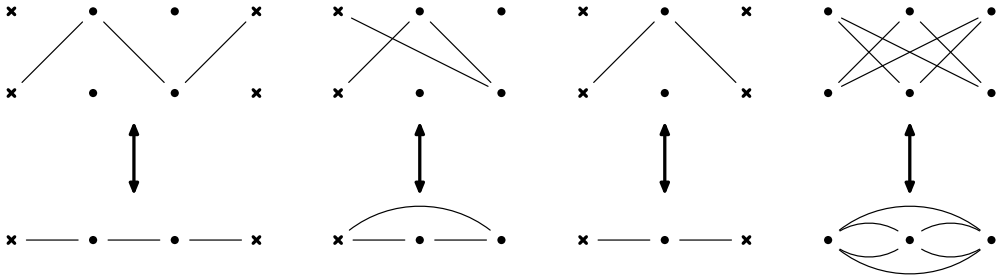


Figure 4 Four possible trail-like components of $\widetilde{G}$ and their images in G^{12}: an odd T-trail, an odd cyclic trail passing through terminal, an even T-trail and an even cyclic trail. Terminals are crosses, other vertices are black dots.

$(G^{12}, T, \mathbf{1})$. The second category corresponds to odd cyclic trails passing through terminals in $(G^{12}, T, \mathbf{1})$. Denote the packings in $(G^{12}, T, \mathbf{1})$ (with unit weights) corresponding to these two categories as $\mathcal{Q}$ and $\mathcal{R}$ respectively, and let $q := \|\mathcal{Q}\|$ and $r := \|\mathcal{R}\|$.

Note that $\mathcal{P} + \mathcal{Q} + \mathcal{R} + \mathcal{E}$ is an integer packing of (possibly cyclic) trails in $(G^{12}, T, \mathbf{1})$ that traverses each edge in G^{12} exactly once.

Starting from this moment we forget about graph $\widetilde{G}$ and release the notation $(\cdot)'$ of its meaning of symmetry in $\widetilde{G}$.

Perform **terminal evacuation** as follows. For each terminal $t \in T$ introduce a new terminal t' connected to t by a certain number of edges. Namely, extend each $t_1 - t_2$ trail (t_1 and t_2 may coincide) in $\mathcal{P} + \mathcal{Q} + \mathcal{R}$ with new $t_1' - t_1$ and $t_2 - t_2'$ valencies in G^{12}, obtaining a new valence graph G'^{12} and new integer packings $\mathcal{P}', \mathcal{Q}', \mathcal{R}'$ in $(G'^{12}, T', \mathbf{1})$.

Note that originally each terminal $t \in T$ had an even number of adjacent edges in G^{12}, therefore it serves as an endpoint for an even number of trails in $\mathcal{P} + \mathcal{Q} + \mathcal{R}$ (counting endpoints of $\mathcal{R}$ twice). Hence, for any $t \in T$ the number of added $t' - t$ valencies is even, therefore the underlying graph G' is well-defined and can be constructed by adding half the number of $t' - t$ valencies. Note that odd (resp. even) T'-trails in G'^{12} correspond to odd (resp. even) T-trails in G^{12}.

Now construct signing M' for G'^{12} by: turning trails in $\mathcal{P}'$ and $\mathcal{R}'$ into odd alternating trails starting and ending with + valencies; turning trails in $\mathcal{Q}'$ into even alternating trails (in any of two possible ways); turning each (cyclic) trail in $\mathcal{E}$ alternating (in any of two possible ways). Clearly, such M' is inner balanced w.r.t. T'. Also M' is (p, q)-tight. Indeed, $\mathcal{P}' + \mathcal{Q}'$ is a T'-trail packing of value $p + q$. Finally, "−" valencies adjacent to terminals T' correspond to T'-trails in $\mathcal{Q}'$, hence there are exactly q of them. Hence we proved the following theorem.

▶ **Theorem 25.** *Given network $(G, T, \mathbf{2})$ with inner Eulerian G such that the maximum value of an odd T-walk packing in $(G, T, \mathbf{2})$ is p, it is possible to construct in polynomial time a signed valence network $(G'^{12}, M', T', \mathbf{1})$ with an inner balanced signing M' such that:*

- *M' is (p, q)-tight for some q;*
- *any packing $\mathcal{P}'$ of odd T'-trails in $(G', T', \mathbf{2})$ can be transformed into a packing $\mathcal{P}$ of odd T-trails in $(G, T, \mathbf{2})$ of the same value in polynomial time.*

5.3 Subcubization

In this section we prove that it is sufficient to solve the problem only for graphs with degree of non-terminal vertices not exceeding 3, which simplifies the subsequent case splitting.

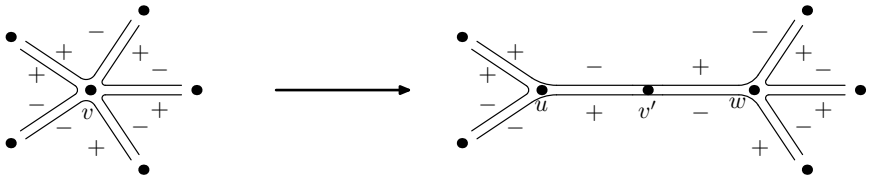


Figure 5 Subcubization at v.

▶ **Definition 26.** *Valence network* $(G^{12}, T, \mathbf{1})$ *is called* ***inner subcubic****, if* $\deg v \le 3$ *for any* $v \in V(G) - T$.

Define the **supercubicity** of G to be

$$s(G) := \sum_{v \in V(G)-T} \max\{0, \deg v - 3\}.$$

Obviously, $s(G) = 0$ for inner subcubic networks.

Let $(G^{12}, M, T, \mathbf{1})$ be a signed valence network with a (p, q)-tight inner balanced signing M. Apply Lemma 23 to construct an integer packing $\mathcal{P} + \mathcal{Q}$, where $\mathcal{P}$ (resp. $\mathcal{Q}$) consists of at least p (resp. at most q) odd (resp. even) alternating T-trails.

Consider an inner vertex v of degree $d \ge 4$. Denote edges incident to v in G as $\delta_G(v) = \{e_1, \dots, e_d\}$. Whenever some trail W in $\mathcal{P} + \mathcal{Q}$ passes through v, it contains a pair of consequent valencies corresponding to some edges $\{e_i, e_j\}$ in $E(G)$; call (e_i, e_j) for $i \le j$ an (ordered) **transit pair**. (Note that this ordering of e_i and e_j is not related to the order in which these edges are passed by W.) Clearly, whenever an alternating trail passes through a transit pair, it takes valencies of opposite signs.

Valencies corresponding to edges in $\delta_G(v)$ not traversed by any of W_i could also be (arbitrarily) divided into pairs of opposite signs (due to signs balance). Fix some division; it generates (by replacing valencies with their preimages in G) more pairs (e_i, e_j) for $i \le j$ that we also regard as transit. Totally we get exactly d transit pairs.

▶ **Lemma 27.** *One can partition the set of incident edges* $\delta_G(v)$ *into two subsets* $L \sqcup R$ *such that* $|L|, |R| \ge 2$ *and there are at most two transit pairs* (e_i, e_j) *(call them* ***split transit pairs****) such that* e_i, e_j *belong to distinct subsets, i.e.* $e_i \in L$, $e_j \in R$ *or* $e_i \in R$, $e_j \in L$.

Proof. Suppose there exists a transit pair (e_i, e_j) for $i < j$. Define $L := \{e_i, e_j\}$ and $R := \delta_G(v) - L$. Each split transit pair must use another valence of e_i or e_j, hence there could be at most two such pairs.

On the other hand, if all transit pairs are of the form (e_i, e_i), then an arbitrary partition $L \sqcup R$ with $|L|, |R| \ge 2$ will do. ◀

Construct a new valence network $(G', M', T' = T, \mathbf{1})$ (Figure 5) by replacing vertex v with three vertices u, v', w and two edges uv', $v'w$, and also replacing v with u in all edges from L and replacing v with w in all edges from R. There are either 0 or 2 split transit pairs; if there are two of them, extend these trails by inserting valencies of uv' and $v'w$ with suitable signs so that signing M' is inner balanced. If there are no transit pairs, simply make both uv' and $v'w$ have one positive and one negative valence. Thus, we also obtain a new packing $\mathcal{P}' + \mathcal{Q}'$ in $(G'^{12}, M', T, \mathbf{1})$, where $\mathcal{P}'$ (resp. $\mathcal{Q}'$) contains at least p (resp. at most q) odd (resp. even) alternating T-trails not passing through terminals as intermediate vertices.

▶ **Lemma 28.** $s(G') = s(G) - 1$ *for the resulting* G'.

Proof. First of all, $\deg v' = 2 < 3$, so we do not need to consider v' when calculating the change of supercubicity. Then, $\deg u = 1 + |L| \geq 3$ and $\deg w = 1 + |R| \geq 3$; also $\deg u + \deg w = 2 + |L| + |R| = d + 2$ and finally $\deg v = d$. We conclude:

$$s(G') - s(G) = \max\{0, \deg u - 3\} + \max\{0, \deg w - 3\} - \max\{0, \deg v - 3\} =$$
$$(\deg u - 3) + (\deg w - 3) - (\deg v - 3) = (\deg u + \deg w) - \deg v - 3 = (d+2) - d - 3 = -1\,.$$

◀

Repeat these transformations until there are no more inner vertices with degree more than 3. We obtain an inner subcubic signed valence network $(G'^{12}, M', T' = T, \mathbf{1})$ with an inner balanced signing M'. Note that any T'-trail W' in $(G'^{12}, T', \mathbf{1})$ may easily be transformed into a T-trail W in $(G^{12}, T, \mathbf{1})$ of the same parity by performing all actions in the reverse order and removing added parts of W', if there are any.

▶ **Lemma 29.** *The resulting signing M' is (p, q)-tight.*

Proof. The total number of "−" valencies adjacent to terminals does not change during subcubization. Also, packing $\mathcal{P}' + \mathcal{Q}$' has the same value as $\mathcal{P} + \mathcal{Q}$, i.e. $p + q$. ◀

Hence we proved the following theorem.

▶ **Theorem 30.** *If $(G^{12}, M, T, \mathbf{1})$ is signed valence network with a (p, q)-tight inner balanced signing M, it is possible to construct a signed valence network $(G'^{12}, M', T' = T, \mathbf{1})$ with a (p, q)-tight inner balanced signing M' such that:*

- *G' is inner subcubic;*
- *any packing $\mathcal{P}'$ of odd T'-trails in $(G', T', \mathbf{2})$ may be transformed into packing $\mathcal{P}$ of odd T-trails in $(G, T, \mathbf{2})$ of the same value in polynomial time.*

5.4 Regularization

Let $(G^{12}, M, T, \mathbf{1})$ be an inner subcubic signed valence network with a (p, q)-tight inner balanced signing M. Construct an integer packing $\mathcal{P} + \mathcal{Q}$ of at least p odd alternating T-trails (denoted by $\mathcal{P}$) and at most q even alternating T-trails (denoted by $\mathcal{Q}$) using Lemma 23.

Suppose there is edge xy in $E(G)$ that is irregular for some T-trail W in $\mathcal{P} + \mathcal{Q}$, i.e. W traverses both of xy's valencies in G^{12}. Denote the fragment of W between two occurrences of valencies of xy (but not including them) by C.

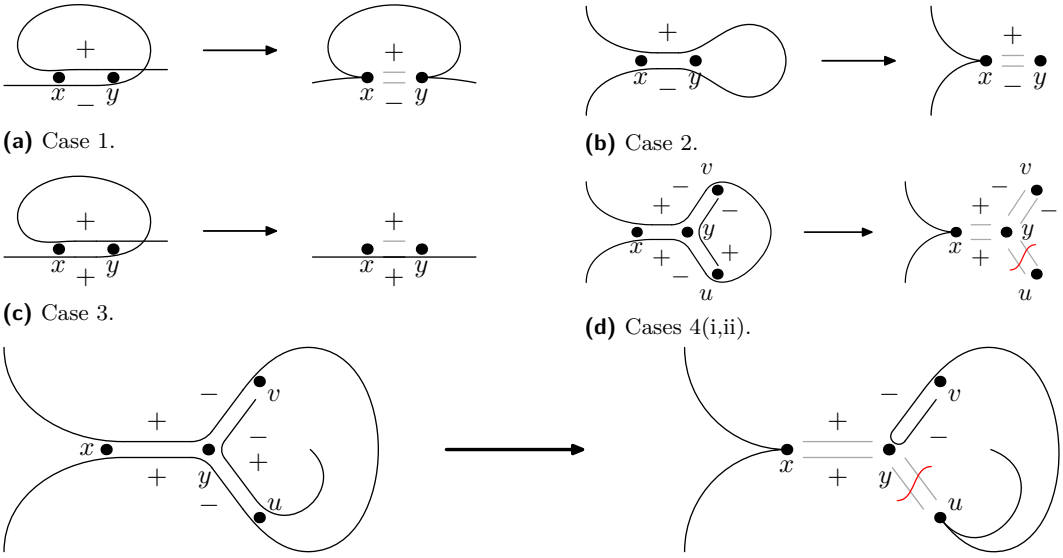
Note that xy is not adjacent to any terminal since all T-trails in $\mathcal{P} + \mathcal{Q}$ are assumed to avoid passing through terminals as intermediate vertices. Consider cases as follows:

Case 1 (Figure 6a): valencies of xy have opposite signs and W traverses them in the same direction. Simplify W by dropping occurrences of both of these valencies.

Case 2 (Figure 6b): valencies of xy have opposite signs and W traverses them in the opposite directions. Simplify W by dropping occurrences of both of these valencies together with C.

Case 3 (Figure 6c): valencies of xy have same signs and W traverses them in the same direction. Simplify W by dropping one of the occurrences of these valencies together with C.

Case 4 (Figures 6d and 6e): valencies of xy have same signs and W traverses them in the opposite directions. Assume that xy is chosen such that C is the shortest possible. W.l.o.g. y belongs to C. Finally assume that both valencies of xy are +; the remaining case is done analogously.



(a) Case 1.

(b) Case 2.

(c) Case 3.

(d) Cases 4(i,ii).

(e) Cases 4(iii,iv).

Figure 6 Regularization cases.

▶ **Lemma 31.** *In Case 4,* $\deg y = 3$ *and* C *starts with a negative valence of some edge* yu *and terminates with a negative valence of some edge* vy *with* $u \neq v$.

Proof. If $\deg y = 1$, the inner Eulerianess of y is contradicted as both of its adjacent valencies are positive.

If $\deg y = 2$, two valencies of the remaining adjacent edge are both negative and W must follow both of them in order to be alternating. This contradicts the choice of xy with the shortest C.

Finally $\deg y = 3$ and C starts with some valence of yu and terminates with some valence of vy, both of which are negative. If $u = v$, this would again contradict the choice of xy with the shortest C. ◀

Consider two remaining valencies of yu and yv. For signing to be balanced at y, one of them must be + and another must be −. Therefore, one of yu and yv has both a + and a − valence; assume it is yu, the other case is done analogously. Call yu **redundant** and obtain a new signed valence network $(G'^{12}, M', T' = T, \mathbf{1})$ by removing yu in G'.

▶ **Lemma 32.** *New signing* M' *is inner balanced and* (p, q)*-tight.*

Proof. Signing M' is inner balanced since we remove two valencies of the same edge of opposite signs. Also, the removed edge is not adjacent to a terminal, therefore the total number of "−" valencies adjacent to terminals is preserved.

Let us prove that a packing of T-trails of value at least $p + q$ still remains. Namely, we alter $\mathcal{P} + \mathcal{Q}$ so that none of its T-trails passes through valencies of the removed edge yu.

W.l.o.g. let e^1 be the valence of $e = yu$ that is the initial or the final valence of C. Alter W by removing both valencies of xy and C, obtaining a (non-alternating) subtrail W' avoiding e^1. Note that the remaining valence e^2 may either: (i) not belong to any trail in $\mathcal{P} + \mathcal{Q}$; (ii) belong to C; (iii) belong to the same trail W outside of C; (iv) belong to another trail in $\mathcal{P} + \mathcal{Q}$.

In subcases (i,ii) (Figure 6d) e^2 is no longer used by any trail in $\mathcal{P}+\mathcal{Q}$; replace W with W'. In subcases (iii,iv) (Figure 6e), consider trail containing e^2 and replace e^2 in it with the $y-u$ fragment of C that is different from e^1 (note that C is not used by W' anymore). ◀

Repeat the procedure until no more irregular edges exist. In Cases 1–3 the signed valence graph does not change, but the total length of odd T-trails in the packing decreases. Therefore, this step may be iterated until either there are no irregular edges or Case 4 happens and we obtain a new signed graph (G'^{12}, M'). In other words, $(|E(G)|, L)$, where L is the total length of T-trails in $\mathcal{P}$, decreases lexicographically in each case. Thus the total number of iterations is polynomial. Let us summarize the result of this section by the following theorem.

▶ **Theorem 33.** *If $(G^{12}, M, T, \mathbf{1})$ is an inner subcubic signed valence network with a (p,q)-tight inner balanced signing M, then it is possible to construct an integer packing of odd T-trails in $(G, T, \mathbf{2})$ of value at least p in polynomial time.*

5.5 Concluding the proof

Proof of Theorem 14. Let $(G, T, \mathbf{2})$ be a inner Eulerian network and let p be the value of a maximum odd T-walk packing in it. Apply Theorem 25 to construct a signed valence network $(G'^{12}, M', T', \mathbf{1})$ with a (p,q)-tight inner balanced signing M' (for some q). By Theorem 30 the latter network can be replaced by a subcubic signed valence network $(G''^{12}, M'', T'', \mathbf{1})$ with a (p,q)-tight inner balanced signing M''. Now Theorem 33 implies the existence of packing $\mathcal{P}''$ of odd T''-trails of value p in $(G'', T'', \mathbf{2})$.

Finally reverse the changes applied to network: $\mathcal{P}''$ gives rise to packing $\mathcal{P}'$ of odd T'-trails of the same value p in $(G', T', \mathbf{2})$ (by Theorem 30); in its turn, $\mathcal{P}'$ generates packing $\mathcal{P}$ of odd T-trails of value p in $(G, T, \mathbf{2})$ (by Theorem 25), as needed.

Note that all of the above steps take polynomial time. ◀

References

1. Maxim A. Babenko and Alexander V. Karzanov. Free multiflows in bidirected and skew-symmetric graphs. *Discret. Appl. Math.*, 155(13):1715–1730, 2007.
2. B. V. Cherkassky. A solution of a problem on multicommodity flows in a network. *Ekonomika i Matematicheskie Metody*, 13(1):143–151, 1977.
3. M. Chudnovsky, J. Geelen, and W. Cunningham. An algorithm for packing non-zero A-paths in group-labelled graphs. *Combinatorica*, 28(2):145–161, 2008.
4. Ross Churchley, Bojan Mohar, and Hehui Wu. Weak duality for packing edge-disjoint odd (u, v)-trails. 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 2086–2094. SIAM, 2016.
5. Sharat Ibrahimpur and Chaitanya Swamy. Min-max theorems for packing and covering odd (u, v)-trails. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 279–291. Springer, 2017.
6. Alexander Karzanov. On a class of maximum multicommodity flow problems with integer optimal solutions. *American Mathematical Society Translations, Ser. 2*, 158:81–99, February 1994. doi:10.1090/trans2/158/09.
7. L. Lovász. On some connectivity properties of Eulerian graphs. *Acta Math. Akad. Sci. Hung.*, 28:129–138, 1976.

8 W. Mader. Über die maximalzahl kantendisjunkter H-wege. *Archiv der Mathematik (Basel)*, 31:382–402, 1978.

9 G. Pap. Packing non-returning A-paths algorithmically. *Discrete Mathematics*, 308(8):1472–1488, 2008.

10 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

11 Alexander Schrijver and Paul D. Seymour. Packing odd paths. *J. Comb. Theory, Ser. B*, 62(2):280–288, 1994.

12 Yutaro Yamaguchi. *Combinatorial Optimization on Group-Labeled Graphs*. PhD thesis, Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Japan, 2016.

A Proof of Theorem 12

We are going to use the following min-max relation for multiflows due to Karzanov under the same commodity graph constraints as in Theorem 10; see [6]:

▶ **Theorem 34.** *Let (G,T,cap) be a network and H be a commodity graph obeying the conditions of Theorem 10. Consider a partition $\mathcal{X} := \{X_1, X_2, \dots, X_k\}$ of T into disjoint nonempty sets $X_1, X_2, \dots, X_k$; call $\mathcal{X}$ **proper** if each X_i is an independent set of H. Define the **capacity** of $\mathcal{X}$ as $cap(\mathcal{X}) := \frac{1}{2}\sum_{i=1}^{k} \lambda(X_i, T-X_i)$; also let Y_i $(X_i \subseteq Y_i \subseteq V(G)-(T-X_i))$ be the corresponding minimum cut between X_i and $T-X_i$ in G.*

Let $\mathcal{F}$ range over multiflows in network (G,T,cap) with commodity graph H, and $\mathcal{X}$ range over proper partitions of T; then

$$\max_{\mathcal{F}} \|\mathcal{F}\| = \min_{\mathcal{X}} cap(\mathcal{X}).$$

Moreover, the minimum $\mathcal{X}$ can be chosen such that Y_i are pairwise disjoint.

Let us rewrite the capacity of an odd T-walk barrier as follows.

▶ **Definition 35.** *Let H be a (not necessarily induced) subgraph of G. Define function $S[H] : E(G) \to \{0, \frac{1}{2}, 1\}$ called a **slice of H** by $S[H](e) := 1$ for $e \in U(H)$, $S[H](e) := \frac{1}{2}$ for $e \in I(H)$, and 0 otherwise.*

A similar notion of slices (differing by a factor of 2) earlier appeared in [11]. Now for an odd T-walk barrier $cap(B) = cap \cdot S[B]$, where $\cdot$ stands for the scalar product.

▶ **Lemma 36.** *Let H be a (not necessarily induced) subgraph of G and $\mathcal{C}$ be the family of connected components of H. Then $S[H] = \sum_{C \in \mathcal{C}} S[C]$ (regarded as functions on $E(G)$).*

Proof. If $S[H](xy) = 1$, either x and y belong to the same connected component C_1, in which case $S[C_1](xy) = 1$ and $S[C_2](xy) = 0$ for any other $C_2 \in \mathcal{C}$, $C_2 \neq C_1$, or to two distinct connected components $C_1, C_2 \in \mathcal{C}$, in which case $S[C_1](xy) + S[C_2](xy) = \frac{1}{2} + \frac{1}{2} = 1$ and $S[C_3](xy) = 0$ for any other $C_3 \in \mathcal{C}$, $C_3 \neq C_1, C_2$.

If $S[H](xy) = \frac{1}{2}$, let C_1 be the connected component containing one of xy's endpoints; in this case $S[C_1](xy) = \frac{1}{2}$ and $S[C_2](xy) = 0$ for $C_2 \in \mathcal{C}$, $C_2 \neq C_1$.

Otherwise $S[H](xy) = 0$ and $S[C_1](xy) = 0$ for any $C_1 \in \mathcal{C}$. ◀

▶ **Lemma 37.** *Let $\mathcal{X}$ range over proper partitions of $\widetilde{T}$ and B be fixed odd T-walk barrier, then*

$$\min_{\mathcal{X}} cap(\mathcal{X}) \leq cap(B).$$

Proof. Consider some barrier B and let $\mathcal{C}$ denote the family of connected components of B. We construct a proper partition $\mathcal{X}$ of $\widetilde{T}$ and a corresponding family of cuts separating X and $\widetilde{T} - X$ for any $X \in \mathcal{X}$. Moreover, the capacities of these cuts are bounded by corresponding summands in $\sum_{C \in \mathcal{C}} cap \cdot S[C]$.

Call a component $C \in \mathcal{C}$ **redundant** if it contains no terminals from T and **singular** if it contains a single terminal. Otherwise C must be **bipartite** (regarded as a graph) and all terminals must belong to the same part of bipartition so that there is no odd T-walk within this component. Construct a proper partition as follows.

If C is singular containing just terminal $t \in T$, enclose t with its symmetric vertex t' with set $X := \{t, t'\}$, which is a maximal anticlique in H_T. Note that $Y := C \cup C'$ is a cut between X and $\widetilde{T} - X$; edges of $\delta(Y)$ in $\widetilde{G}$ correspond to edges of $I(C)$ in G. Thus

$$\frac{1}{2}\widetilde{cap}(\delta(Y)) = \frac{1}{2}\sum_{xy \in \delta(Y)} \widetilde{cap}(xy) = \frac{1}{2}\sum_{xy \in I(C)} (\widetilde{cap}(xy') + \widetilde{cap}(x'y)) = \\ = \frac{1}{2}\sum_{xy \in I(C)} cap(xy) = \frac{1}{2} cap(I(C)) \le cap \cdot S[C]. \quad (1)$$

If C is bipartite containing multiple terminals $t_1, \ldots, t_k$, introduce sets $X := \{t_1, \ldots, t_k\}$ and $X' := \{t'_1, \ldots, t'_k\}$, which are subsets of maximal anticliques T and T' in H_T, respectively. Let L and R be the bi-partition parts of C such that $t_1, \ldots, t_k \in L$. Note that $Y := L \sqcup R'$ is a cut between X and $\widetilde{T} - X$ and, symmetrically, $Y' = L' \sqcup R$ is a cut between X' and $\widetilde{T} - X'$.

There are two kinds of edges in $\delta(Y) \cup \delta(Y')$ in $\widetilde{G}$: the ones connecting Y or Y' with $V(G) - (Y \sqcup Y')$, and the ones connecting Y with Y'. Again, the former ones correspond to edges in $I(C)$, while the latter ones are edges connecting L with L' or R with R'; therefore their pre-images belong to $\gamma(L)$ or $\gamma(R)$, hence they belong to $U(C)$. Using these observations, we get the following:

$$\frac{1}{2}\left(\widetilde{cap}(\delta(Y)) + \widetilde{cap}(\delta(Y'))\right) = \frac{1}{2}\sum_{xy \in \delta(Y \sqcup Y')} \widetilde{cap}(xy) + \sum_{x \in Y,\, y \in Y'} \widetilde{cap}(xy) \le \\ \le \frac{1}{2}\sum_{xy \in I(C)} (\widetilde{cap}(xy') + \widetilde{cap}(x'y)) + \sum_{xy \in U(C)} (\widetilde{cap}(xy') + \widetilde{cap}(x'y)) = \\ = \frac{1}{2}\sum_{xy \in I(C)} cap(xy) + \sum_{xy \in U(C)} cap(xy) = \frac{1}{2} cap(I(C)) + cap(U(C)) = cap \cdot S[C]. \quad (2)$$

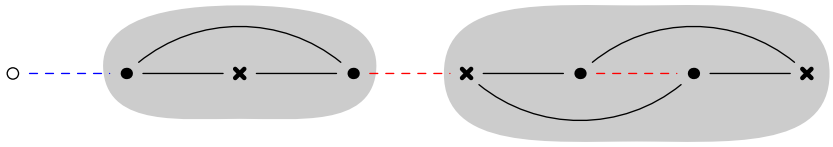
Finally, if C is redundant, it does not produce any set for our proper partition. Note that each vertex $v \in \widetilde{T}$ belongs to exactly one of the formed sets in our partition.

Summing Equation (1) and Equation (2) over all singular and bipartite components, we get $cap(\mathcal{X}) \le cap \cdot S[B] = cap(B)$, which completes the proof. ◀

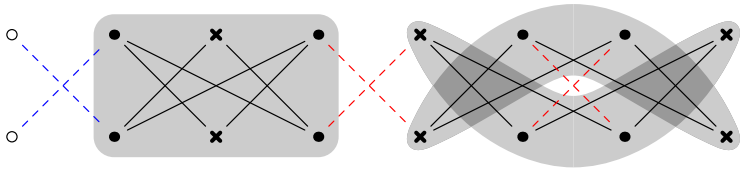
▶ **Lemma 38.** *There exists a proper partition $\mathcal{X}$ with minimum $cap(\mathcal{X})$ such that no two $X_1, X_2 \in \mathcal{X}$ are subsets of the same maximal anticlique in H_T.*

Proof. Construct another proper partition $\mathcal{Z}$ by replacing X_1 and X_2 with $X_1 \sqcup X_2$. The change in capacity is $cap(\mathcal{Z}) - cap(\mathcal{X}) = \lambda(X_1 \sqcup X_2, \widetilde{T} - (X_1 \sqcup X_2)) - \lambda(X_1, \widetilde{T} - X_1) - \lambda(X_2, \widetilde{T} - X_2)$.

Let Y_i be a minimum cut between X_i and $\widetilde{T} - X_i$ for $i = 1, 2$. Submodularity of cut capacities [10, Sec. 44.1a] implies $\lambda(X_1, \widetilde{T} - X_1) + \lambda(X_2, \widetilde{T} - X_2) = cap(\delta(Y_1)) + cap(\delta(Y_2)) \ge cap(\delta(Y_1 \cap Y_2)) + cap(\delta(Y_1 \cup Y_2)) \ge 0 + \lambda(X_1 \sqcup X_2, \widetilde{T} - (X_1 \sqcup X_2))$. Therefore, $cap(\mathcal{Z}) \le cap(\mathcal{X})$. Repeat merging parts until done. ◀



(a) Odd T-walk barrier B in (G, T, c): left connected component is singular; right connected component is non-singular; blue and red edges are $I(B)$ and $U(B)$, respectively.



(b) Proper partition $\mathcal{X}$ in $(\widetilde{G}, \widetilde{T}, \widetilde{c})$: left set is singular; two right sets are non-singular and symmetric; blue and red edges are accounted in $cap(\mathcal{X})$ with weight $\frac{1}{2}$ and 1, respectively.

Figure 7 Odd T-walk barrier B and its corresponding proper partition $\mathcal{X}$.

From the previous lemma trivially follows the following corollary.

▶ **Corollary 39.** *There exists a proper partition $\mathcal{X}$ with minimum $cap(\mathcal{X})$ such that all of its sets are of the form $\{t, t'\}$ for $t \in T$ (call them **singular**) except for, possibly, two symmetric sets $X, X' \in \mathcal{X}$ (call them **non-singular**).*

▶ **Lemma 40.** *There exists a proper partition $\mathcal{X}$ with minimum $cap(\mathcal{X})$ such that the associated minimum cuts between X and $\widetilde{T} - X$ for $X \in \mathcal{X}$ obey the following properties:*

- *if X is singular, the corresponding cut Y is self-symmetric, i.e. $Y' = Y$;*
- *if X and X' are non-singular, their corresponding cuts are also symmetric to each other;*
- *all the above cuts are disjoint.*

Proof. Using Theorem 34, we may choose $\mathcal{X}$ such that the corresponding cuts Y, Y' are disjoint.

If Y is a minimum cut between $\{t, t'\}$ and $\widetilde{T} - \{t, t'\}$ for some $t \in T$, then so is Y'. Since by submodularity $cap(\delta(Y)) + cap(\delta(Y')) \geq cap(\delta(Y \cap Y')) + cap(\delta(Y \cup Y'))$, it follows that $Y \cap Y'$ is also a minimum cut between $\{t, t'\}$ and $\widetilde{T} - \{t, t'\}$; also $Y \cup Y'$ is self-symmetric by construction. Replace Y with $Y \cap Y'$.

The second part of the statement is similar. If X and X' are non-singular and Y_1 (resp. Y_2) is a minimum cut between X and $\widetilde{T} - X$ (resp. X' and $\widetilde{T} - X'$), then Y_2' and (resp. Y_1') is also a minimum cut for the same vertex sets. Using a similar argument, replace Y_1 and Y_2 with $Y_1 \cap Y_2'$ and $Y_2 \cap Y_1'$, which are also minimum cuts symmetric to each other.

The steps above replace cuts with their subsets; therefore the last property is preserved. ◀

▶ **Lemma 41.** *Let $\mathcal{X}$ range over proper partitions of T and B range over odd T-walk barriers, then*

$$\min_{\mathcal{X}} cap(\mathcal{X}) \geq \min_{B} cap(B).$$

Proof. Pick $\mathcal{X}$ with the minimum capacity $cap(\mathcal{X})$ satisfying the properties from the previous lemma.

If $X = \{t, t'\} \in \mathcal{X}$ is singular and Y is the corresponding cut, consider the pre-image $C \subseteq V(G)$ of Y. Add the subgraph of G induced by C to barrier B. Note that $U(C) = \varnothing$. Now

$$cap \cdot S[C] = \frac{1}{2} cap(I(C)) = \frac{1}{2} \sum_{xy \in I(C)} cap(xy) =$$
$$= \frac{1}{2} \sum_{xy \in I(C)} (\widetilde{cap}(xy') + \widetilde{cap}(x'y)) = \frac{1}{2} \widetilde{cap}(\delta(Y)). \quad (3)$$

If $X, X' \in \mathcal{X}$ are non-singular, consider their corresponding cuts Y and Y'. Let $Y = L \sqcup R'$ for $L, R \subseteq V(G)$ (which implies $Y' = L' \sqcup R$). Add the bipartite subgraph D of G induced by L and R to barrier B. Note that $U(D)$ is the pre-image of edges xy' with $x \in Y$ and $y' \in Y'$. Therefore

$$cap \cdot S[D] = cap(U(D)) + \frac{1}{2} cap(I(D)) = \sum_{xy \in U(D)} cap(xy) + \frac{1}{2} \sum_{xy \in I(D)} cap(xy) =$$
$$= 2 \cdot \frac{1}{2} \sum_{xy \in U(D)} (\widetilde{cap}(xy') + \widetilde{cap}(x'y)) + \frac{1}{2} \sum_{xy \in I(D)} (\widetilde{cap}(xy') + \widetilde{cap}(x'y)) =$$
$$= \frac{1}{2} (\widetilde{cap}(\delta(Y)) + \widetilde{cap}(\delta(Y'))). \quad (4)$$

◀

Proof of Theorem 12. From Theorem 9 we know that $\max_{\mathcal{P}} \|\mathcal{P}\| = \max_{\mathcal{Q}} \|\mathcal{Q}\|$, where $\mathcal{P}$ ranges over odd T-walk packings and $\mathcal{Q}$ ranges over multiflows in $(\widetilde{G}, \widetilde{T}, \widetilde{cap})$ with commodity graph H_T. Then, from Theorem 34 it follows that $\max_{\mathcal{Q}} \|\mathcal{Q}\| = \min_{\mathcal{X}} cap(\mathcal{X})$, where $\mathcal{X}$ ranges over proper partitions of T in H_T. Finally, from Lemma 37 and Lemma 41 it follows that $\min_{\mathcal{X}} cap(\mathcal{X}) = \min_B cap(B)$. ◀

Improved Weighted Matching in the Sliding Window Model

Cezar-Mihail Alexandru ✉
Department of Computer Science, University of Bristol, UK

Pavel Dvořák ✉
Tata Institute of Fundamental Research, Mumbai, India
Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

Christian Konrad ✉
Department of Computer Science, University of Bristol, UK

Kheeran K. Naidu ✉
Department of Computer Science, University of Bristol, UK

Abstract

We consider the Maximum-weight Matching (MWM) problem in the streaming sliding window model of computation. In this model, the input consists of a sequence of weighted edges on a given vertex set V of size n. The objective is to maintain an approximation of a maximum-weight matching in the graph spanned by the L most recent edges, for some integer L, using as little space as possible. Prior to our work, the state-of-the-art results were a $(3.5+\varepsilon)$-approximation algorithm for MWM by Biabani et al. [ISAAC'21] and a $(3+\varepsilon)$-approximation for (unweighted) Maximum Matching (MM) by Crouch et al. [ESA'13]. Both algorithms use space $\tilde{O}(n)$.

We give the following results:

1. We give a $(2+\varepsilon)$-approximation algorithm for MWM with space $\tilde{O}(\sqrt{nL})$. Under the reasonable assumption that the graphs spanned by the edges in each sliding window are simple, our algorithm uses space $\tilde{O}(n\sqrt{n})$.
2. In the $\tilde{O}(n)$ space regime, we give a $(3+\varepsilon)$-approximation algorithm for MWM, thereby closing the gap between the best-known approximation ratio for MWM and MM.

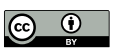
Similar to Biabani et al.'s MWM algorithm, both our algorithms execute multiple instances of the $(2+\varepsilon)$-approximation $\tilde{O}(n)$-space streaming algorithm for MWM by Paz and Schwartzman [SODA'17] on different portions of the stream. Our improvements are obtained by selecting these substreams differently. Furthermore, our $(2+\varepsilon)$-approximation algorithm runs the Paz-Schwartzman algorithm in *reverse direction* over some parts of the stream, and in *forward direction* over other parts, which allows for an improved approximation guarantee at the cost of increased space requirements.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Streaming models

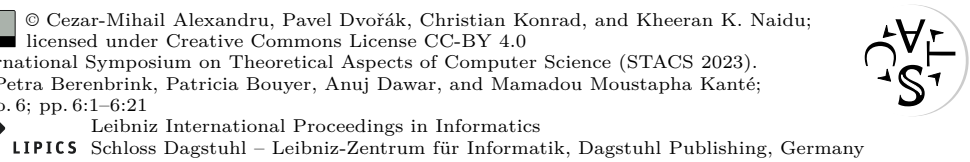
Keywords and phrases Sliding window algorithms, Streaming algorithms, Maximum-weight matching

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.6

Funding *Cezar-Mihail Alexandru*: Supported by EPSRC DTP studentship EP/T517872/1.
Pavel Dvořák: Supported by EPSRC New Investigator Award EP/V010611/1 and by Czech Science Foundation GAČR grant #22-14872O.
Christian Konrad: Supported by EPSRC New Investigator Award EP/V010611/1.
Kheeran K. Naidu: Supported by EPSRC DTP studentship EP/T517872/1.



© Cezar-Mihail Alexandru, Pavel Dvořák, Christian Konrad, and Kheeran K. Naidu;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

The *data streaming model* is a well-established computational model that provides a framework for studying massive data set algorithms. The defining features of the model are restricted access to the input data and sublinear space. A data streaming algorithm processes its input sequentially in a single pass while maintaining only a small summary of the input in memory.

In this paper, we study the Maximum-weight Matching (MWM) problem in the *(streaming) sliding window* model. In this variant of the streaming model, the input consists of a potentially infinite sequence $e_1, e_2, \ldots$ of weighted edges on an underlying vertex set V of size n. The objective is to maintain a matching of large weight in the graph spanned by the L most recent edges, for some integer L, using as little space as possible. In more detail, after having processed the current edge e_i, for every i, the objective is to report an approximation of a maximum-weight matching in the graph spanned by the current sliding window $E_i := \{e_j \ : \ \max\{i - L + 1, 1\} \leq j \leq i\}$. Many of the known sliding window algorithms for graph problems operate within *semi-streaming space* [12], i.e., within space $O(n \operatorname{polylog} n) = \tilde{O}(n)$. In this paper, we will work both with the semi-streaming space regime and also consider algorithms that use more space.

While sliding window algorithms have been studied for two decades [10], sliding window algorithms for graph problems were first considered by Crouch et al. [7] in 2013. Amongst other results, they showed that there is a $(3 + \varepsilon)$-approximation semi-streaming sliding window algorithm for unweighted Maximum Matching (MM) and a 9.027-approximation semi-streaming sliding window algorithm for MWM. While no improved results are known for MM, Crouch and Stubbs [8] subsequently improved upon the result for MWM and gave a $(6 + \varepsilon)$-approximation semi-streaming algorithm, and, very recently, Biabani et al. [4] gave a $(3.5 + \varepsilon)$-approximation in the semi-streaming space regime. The state-of-the-art results for MM and MWM in the semi-streaming sliding window model therefore do not yet line up.

Our Results

In this paper, we give two sliding window algorithms for MWM that both improve upon the state-of-the-art approximation guarantee of $3.5 + \varepsilon$.

As our first result, we give a substantial improvement and obtain an approximation factor of $2 + \varepsilon$ at the expense of increased space requirements:

▶ **Theorem 1** (simplified version). *There is a deterministic $(2 + \varepsilon)$-approximation sliding window algorithm for Maximum-weight Matching that uses space $\tilde{O}(\sqrt{nL})$ (with dependency on ε and logarithms suppressed), for any $\varepsilon > 0$.*

Some remarks are in order. First, we observe that going beyond the approximation factor of 2, even using space $O(n^{1.999})$, would answer a long-standing open problem in graph streaming research, namely, whether there is a one-pass $(2 - \Omega(1))$-approximation streaming algorithm for MM with space $O(n^{1.999})$. We thus cannot expect to obtain further improvements in the approximation guarantee with current techniques. Second, the space requirements of our algorithm depend on the sliding window length L. This is in contrast to the $(3.5 + \varepsilon)$-approximation algorithm by Biabani et al. [4] and our second algorithm described below. Under the natural assumption that the graphs described by all sliding windows are simple, we have $L = O(n^2)$, which yields a space bound of $\tilde{O}(n\sqrt{n})$.

As our second result, we close the gap between MM and MWM in the semi-streaming space regime. To this end, we give a semi-streaming sliding window algorithm for MWM that matches the approximation guarantee of the best-known sliding window algorithm for MM.

Table 1 Known sliding window algorithms for MM and MWM.

	Approximation Factor	Space	Reference
MM	$3+\varepsilon$	$\tilde{O}(n)$	Crouch et al. [7]
MWM	9.027	$\tilde{O}(n)$	Crouch et al. [7]
	$6+\varepsilon$	$\tilde{O}(n)$	Crouch and Stubbs [8]
	$3.5+\varepsilon$	$\tilde{O}(n)$	Biabani et al. [4]
	$3+\varepsilon$	$\tilde{O}(n)$	This paper (Theorem 2)
	$2+\varepsilon$	$\tilde{O}(\sqrt{nL})$	This paper (Theorem 1)

▶ **Theorem 2** (simplified version). *There is a deterministic semi-streaming sliding window algorithm for Maximum-weight Matching with approximation factor $3+\varepsilon$, for any $\varepsilon > 0$.*

Table 1 summarizes all results known for MM and MWM in the sliding window model.

Techniques

Both our algorithms make use of the one-pass $(2+\varepsilon)$-approximation streaming algorithm for MWM by Paz and Schwartzman [19]. Since we make use of the inner workings of the algorithm, we will discuss this algorithm first.

Paz and Schwartzman's MWM Algorithm. Paz and Schwartzman's original algorithm [19] uses space $O\big(\frac{1}{\varepsilon} \cdot n \log^2 n\big)$ and is based on the *local ratio technique* (see [3] for further details on this technique). Ghaffari and Wajc [13] gave a simplified version and improved the space complexity to the (optimal in n) bound $O\big(\frac{\log(1/\varepsilon)}{\varepsilon} \cdot n \log n\big)$.

The Paz and Schwartzman algorithm with Ghaffari and Wajc's improvement works as follows. For every vertex $v \in V$, it maintains a potential $\varphi(v)$ that is initialized with 0, and uses a stack data structure Stack. When an edge $e = \{u, v\}$ arrives in the stream, e is pushed onto Stack if its weight $w(e)$ exceeds the sum of the potentials of its incident vertices by a factor of at least $(1+\varepsilon)$, i.e., $w(e) \geq (1+\varepsilon)(\varphi(u)+\varphi(v))$. The discrepancy between $w(e)$ and $\varphi(u)+\varphi(v)$ is denoted the *reduced weight* of e and is abbreviated by $w'(e) := w(e) - (\varphi(u) + \varphi(v))$. Then, the potentials $\varphi(u)$ and $\varphi(v)$ are updated as $\varphi(u) = \varphi(u) + w'(e)$ and $\varphi(v) = \varphi(v) + w'(e)$. Last, if either u or v is adjacent to at least $\frac{3\log(1/\varepsilon)}{\varepsilon} + 1$ edges in Stack then the oldest (and thus lightest) edge incident to the vertex is removed from Stack, thereby limiting the number of edges on Stack. After having processed all the edges in the stream, the output matching $\hat{M}$ is computed in a post-processing step. The edges in Stack are popped one by one and greedily inserted into $\hat{M}$ if possible, i.e., as long as $\hat{M}$ remains a matching. We denote the Paz and Schwartzman algorithm by $\mathcal{ALG}^{\varepsilon}_{PS}$. See Section 2 for a formal description.

$(2+\varepsilon)$-approximation Algorithm with Space $\tilde{O}(\sqrt{nL})$. Our $(2+\varepsilon)$-approximation algorithm processes the input in blocks of size $s = \tilde{\Theta}(\sqrt{nL})$. Consider one such block B_j, i.e., a substream of s consecutive edges. The key idea of our algorithm is to run multiple instances of the Paz-Schwartzman algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$ on B_j, however, in *reverse direction*. We start with a single instance $\mathcal{I}_1$. At various moments during the processing of B_j, we fork the current instance $\mathcal{I}_i$ to obtain an additional instance $\mathcal{I}_{i+1}$, and then only continue to feed further edges into $\mathcal{I}_{i+1}$; thus, in any moment of processing the block B_j, we feed the edge to only one instance of the Paz-Schwartzman algorithm. The fork happens when the sum of reduced weights $W'(\mathcal{I}_i)$ of the edges on Stack in $\mathcal{I}_i$ exceeds the sum of reduced weights of the previous instance by a $1+\varepsilon$ factor, i.e., $W'(\mathcal{I}_i) > (1+\varepsilon) \cdot W'(\mathcal{I}_{i-1})$. As a result, we

obtain instances of Paz-Schwartzman that processed suffixes of different lengths of block B_j (remember that we process B_j in the reverse direction), and adjacent instances have a similar sum of reduced weights (up to a $1+\varepsilon$ factor). As we will point out in Section 2, the sum of reduced weights in an instance of Paz-Schwartzman is strongly related to the weight of a maximum-weight matching among the edges observed thus far, and we heavily exploit this property in our proofs.

In each block B_j, besides preparing the instances of Paz-Schwartzman as described above, we also feed the edges of B_j (in the forward direction) into those instances of Paz-Schwartzman that were prepared during previous blocks $B_{j'}$, with $j' < j$, and that are still *alive*, i.e., have only been fed edges from the current sliding window. As such, each instance of Paz-Schwartzman is executed on a portion of the stream in the reverse direction, followed by all the subsequent edges from more recent blocks in the forward direction until the current edge. The output produced when processing the current edge is the output of the oldest alive instance of Paz-Schwartzman.

Consider two adjacent instances $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$ of Paz-Schwartzman prepared in the same block, where $\mathcal{I}_i$ has processed only a subset of the edges of $\mathcal{I}_{i+1}$ and their sums of reduced weights W' are such that $W'(\mathcal{I}_{i+1}) \approx (1+\varepsilon)W'(\mathcal{I}_i)$. The key benefit of executing Paz-Schwartzman in the reverse direction as opposed to forward is that the edges processed by $\mathcal{I}_{i+1}$ but not by $\mathcal{I}_i$ contribute to the sum of reduced weights only with an ε-fraction of $W'(\mathcal{I}_i)$ (since $W'(\mathcal{I}_{i+1}) - W'(\mathcal{I}_i) \approx \varepsilon W'(\mathcal{I}_i)$). When $\mathcal{I}_i$ is the oldest alive instance and thus constitutes the output of our algorithm, we only *miss* an ε-fraction in terms of reduced weights of the edges in the sliding window that $\mathcal{I}_i$ has not considered. We remark that this property could not be established if we run Paz-Schwartzman in the forward direction. This property together with the fact that the sum of reduced weights is related to the weight of a maximum-weight matching allows us to establish the approximation factor of our algorithm.

Since only the L most recent edges are relevant, our algorithm considers at most $\frac{L}{s} = \tilde{\Theta}(\sqrt{L/n})$ blocks simultaneously. Each block consists of $\tilde{O}(1)$ instances of Paz-Schwartzman. Since each of these instances requires space $\tilde{O}(n)$, we obtain the final space bound of $\tilde{O}(n) \cdot \frac{L}{s} = \tilde{O}(\sqrt{nL})$.

$(3+\varepsilon)$-approximation Semi-streaming Algorithm. Our $(3+\varepsilon)$-approximation algorithm follows similar arguments as the $(3.5+\varepsilon)$-approximation algorithm by Biabani et al. [4]. We will therefore first explain the techniques behind Biabani et al.'s algorithm and then discuss our new ideas which yield the improved approximation guarantee.

Biabani et al.'s algorithm combines the *smooth histogram technique* for sliding window algorithms by Braverman and Ostrovsky [6] with the Paz and Schwartzman algorithm. Braverman and Ostrovsky showed that if a function f fulfills certain smoothness criteria[1] then a sliding window algorithm for approximating f can be obtained from a traditional (non-sliding window) streaming algorithm for f at the expense of only a logarithmic increase in the space requirements (as long as the approximation factor of the streaming algorithm is constant), and a slight increase in the approximation factor. In the context of MWM, the smoothness criteria are captured by Biabani et al. [4] via the notion of *lookahead* algorithm.

[1] Informally, a function $f : 2^X \to \mathbb{R}$ is considered to be *smooth* if it satisfies the following: If $f(A)$ is close to $f(B)$ for $A, B \subseteq X$, for a suitable notion of closeness, then the values $f(A \cup C)$ and $f(B \cup C)$ are close for all $C \subseteq X$.

▶ **Definition 3** ((f, α, β)-lookahead algorithm [4]). *Let $\beta \in (0, 1)$ and $\alpha > 0$ be real numbers. Let X be a ground set, S a stream of items of X, and let $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. We say that a streaming algorithm $\mathcal{ALG}$ is a* (f, α, β)-lookahead algorithm *if, for any partitioning of S into three substreams A, B, C with $\mathcal{ALG}(B) \geq (1 - \beta) \cdot \mathcal{ALG}(AB)$, the following holds: $f(ABC) \leq \alpha \cdot \mathcal{ALG}(BC)$.*

In this paper, the stream AB denotes the concatenation of streams A and B (as it is used in the previous definition). We observe that the previous definition holds for real-valued non-decreasing functions. In the context of MWM, the weight of a maximum-weight matching rather than the matching itself fulfills these conditions. We will therefore consider the problem of determining the weight of a maximum-weight matching instead, and, in order to be able to output an actual matching as required in MWM, we will rely on the fact that the underlying algorithm which we will consider also maintains the actual matching itself. Furthermore, we will write $\mathsf{MWM}(S)$ to denote the weight of a maximum-weight matching in stream S.

Biabani et al. [4] showed that if there is a $(\mathsf{MWM}, \alpha, \beta)$-lookahead algorithm that uses space s then there exists a sliding-window algorithm with approximation ratio α and space $O\big(\frac{1}{\beta} \cdot s \log \sigma\big)$, where $\sigma = \frac{n}{2} \cdot w_{\max}/w_{\min}$ and $w_{\max}$ and $w_{\min}$ are the maximum and minimum weights of an edge in the input stream, respectively. Observe that, under the usual assumption that $w_{\max}/w_{\min}$ is polynomial in n, we have $\log \sigma = O(\log n)$.

The main part of their analysis is to show that a monotonic version of the Paz and Schwartzman algorithm, denoted $\mathcal{ALG}_{mon}$, constitutes a $(\mathsf{MWM}, (3.5 + \varepsilon), \beta)$-lookahead algorithm, for small values of ε and $\beta \leq \varepsilon/9$. Combined, this yields a $(3.5+\varepsilon)$-approximation semi-streaming sliding window algorithm for MWM.

We first note (see Appendix A for details) that the analysis of Biabani et al. is best possible in that the Paz and Schwartzman algorithm and its monotonic version are no better than $(\mathsf{MWM}, 3.5, \beta)$-lookahead algorithms. The smooth histogram technique applied to lookahead algorithms as defined in Definition 3 thus cannot give an improved approximation guarantee when Paz and Schwartzman's algorithm is used as the underlying algorithm.

To illustrate our improvement, we first provide insight into the structure of Biabani et al.'s analysis. In order to prove that $\mathcal{ALG}_{mon}$ is a $(\mathsf{MWM}, 3.5 + \varepsilon, \beta)$-lookahead algorithm, Biabani et al. relate $\mathsf{MWM}(ABC)$ to the output of $\mathcal{ALG}_{mon}$ on various substreams of ABC:

$$\begin{aligned} \mathsf{MWM}(ABC) \quad &\leq \quad 2(1+\varepsilon) \cdot \big(\mathcal{ALG}_{mon}(AB) + \mathcal{ALG}_{mon}(BC)\big) \\ &\quad - \frac{1}{2(1+\varepsilon)} \cdot \mathcal{ALG}_{mon}(B) \ . \end{aligned} \tag{1}$$

They subsequently use the smoothness assumption from Definition 3 and a monotonicity property of $\mathcal{ALG}_{mon}$ to relate $\mathcal{ALG}_{mon}(AB)$ and $\mathcal{ALG}_{mon}(B)$ to $\mathcal{ALG}_{mon}(BC)$. This ultimately yields the desired bound $\mathsf{MWM}(ABC) \leq (3.5 + \varepsilon) \cdot \mathcal{ALG}_{mon}(BC)$.

To obtain our improvement, we observe that a similar inequality to Inequality 1 can be obtained by considering *sums of reduced weights* of the respective runs of $\mathcal{ALG}_{mon}$ instead of the weights of the output matchings of $\mathcal{ALG}_{mon}$ on the different substreams. This idea is motivated by the fact that the sum of reduced weights is a lower bound on the weight of the matching produced by the algorithm, which can therefore give a more precise analysis. However, when departing from such an inequality involving sums of reduced weights, we unfortunately cannot immediately complete our analysis since, unlike when considering the outputs of $\mathcal{ALG}_{mon}$ directly, we do not have a sufficient smoothness property regarding sums of reduced weights at our disposal that would allow us to bound these quantities.

Our key idea is as follows. To establish the necessary smoothness properties, we employ the smooth histogram technique directly on sums of reduced weights rather than on the size of the output matching itself. To be consistent with the literature and to illustrate the

increment over Biabani et al.'s work, we encapsulate this idea via an alternative definition of lookahead algorithms, denoted *refined lookahead algorithms* (see Definition 9 for details), which enables us to incorporate the required smoothness property of sums of reduced weights into the definition. We then prove that, similar to lookahead algorithms, refined lookahead algorithms can still be turned into sliding window algorithms with a similar small increase in the space complexity. Last, we finish our argument by proving that $\mathcal{ALG}^{\varepsilon}_{PS}$ is a refined lookahead algorithm with an approximation factor of $3+\varepsilon$, which establishes our result.

Further Related Work

The sliding window model can be regarded as a streaming insertion-deletion model with highly structured deletions since, for each incoming edge, the oldest edge in the current window is deleted. Interestingly, the complexities of MM and MWM in the sliding window model are much closer to those in the insertion-only model, where no deletions are allowed, as opposed to the insertion-deletion model, where arbitrary deletions are allowed. In the insertion-only model, the currently best one-pass algorithm known for MM is the Greedy matching algorithm, which produces a 2-approximation and uses semi-streaming space $\tilde{O}(n)$. It is known that computing a $(1+\ln 2)$-approximation requires strictly more space than $\tilde{O}(n)$ [15], see also the previous lower bounds [14, 16]. It remains a key open problem to close this gap. Regarding MWM, a series of works [12, 18, 21, 11, 8, 19, 13] culminated in the Paz and Schwartzman algorithm, which closes the gap between MWM and MM from an algorithmic perspective in the insertion-only model. In the insertion-deletion model, where arbitrary previously inserted edges can be deleted again, it is known that space $\Theta(n^2/\alpha^3)$ is necessary and sufficient for computing an α-approximation to MM, see [2] for the algorithm and [9] for a matching lower bound (see also the previous works [17, 1]). MWM reduces easily to MM in the insertion-deletion model, by, for example, grouping edges of similar weights into groups and running the MM algorithm a logarithmic number of times in parallel at the expense of only a marginal increase in the approximation factor.

The sliding window model is inspired by the problem of inferring statistics of data occurring within a certain time frame over a continuous stream of data (e.g., maintaining the number of distinct users who have accessed a social media page in the last 24 hours). The model was introduced by Datar et al. [10], and Crouch et al. [7] were the first to consider graph problems in the sliding window model. Among others, they showed that testing Connectivity and Bipartiteness, and constructing $(1+\varepsilon)$-sparsifiers can be done in the sliding window model using semi-streaming space. Furthermore, as previously mentioned, they also gave the first sliding window algorithms for MM and MWM.

The smooth histogram technique used in our work was introduced by Braverman and Ostrovsky [6] and can be regarded as an improvement of the exponential histogram technique [10] for smooth functions. This technique has successfully been applied to a wide range of problems, including the computation of coresets [20] and for clustering problems [5].

Outline

We first give notation and a discussion of Paz and Schwartzman's algorithm including its properties in Section 2. The $(2+\varepsilon)$-approximation is presented in Section 3. The semi-streaming $(3+\varepsilon)$-approximation via the refined lookahead algorithms is then given in Section 4. Finally, we conclude with open questions in Section 5.

2 Preliminaries

In this section, we start with some important notation and a formal description of the improved version of Paz and Schwartzman's algorithm by Ghaffari and Wajc (see Algorithm 1). This is followed by some key insights about the algorithm.

Let S be an input stream representing an edge-weighted graph $G = (V, E, w)$ with a weight function $w : E \to \mathbb{R}^+$. We assume that each edge, including its weight, can be stored in a single word of memory; as such, all our space bounds are in terms of words of memory. For any subset of edges $F \subseteq E$, let $w(F) = \sum_{e \in F} w(e)$ be the sum of their weights. Then, for any maximum-weight matching in G, denoted by $M^*(S)$, we have that $\mathsf{MWM}(S) = w\big(M^*(S)\big)$.

Algorithm 1 $\mathcal{ALG}^{\varepsilon}_{PS}$ (Paz and Schwartzman's algorithm with Ghaffari and Wajc's improvements.)

Input: A stream S of weighted edges

Initialization:

1: $\mathsf{Stack} \leftarrow$ an empty stack
2: **for** every vertex $v \in V$ **do** $\varphi(v) \leftarrow 0$

Streaming:

3: **while** a new edge $e = \{u, v\}$ of the stream S is revealed **do**
4: **if** $w(e) < (1 + \varepsilon) \cdot \big(\varphi(u) + \varphi(v)\big)$ **then** $w'(e) \leftarrow 0$
5: **else**
6: $w'(e) \leftarrow w(e) - \big(\varphi(u) + \varphi(v)\big)$ ▷ $w'(e)$ is the *reduced weight of* e
7: $\varphi(u) \leftarrow \varphi(u) + w'(e)$; $\varphi(v) \leftarrow \varphi(v) + w'(e)$ ▷ update potentials
8: $\mathsf{Stack}.\mathsf{Push}(e)$
9: **for** $x \in \{u, v\}$ **do** ▷ optimizing space
10: **if** x is adjacent to $> \frac{3\log(1/\varepsilon)}{\varepsilon} + 1$ edges in Stack **then**
11: Remove the oldest edge adjacent to x from Stack

Postprocessing:

12: Let $\hat{M}$ be an empty matching
13: **while** Stack is not empty **do**
14: $e \leftarrow \mathsf{Stack}.\mathsf{Pop}()$
15: **if** $\hat{M} \cup \{e\}$ is a matching **then** $\hat{M} \leftarrow \hat{M} \cup \{e\}$
16: **return** $\hat{M}$ ▷ a GREEDY matching of the edges in Stack

$\mathcal{ALG}^{\varepsilon}_{PS}$ (Algorithm 1) uses the notions of reduced weights and vertex potentials. These are respectively represented by the functions $w'_S : E \to \mathbb{R}^+_0$ and $\varphi_S : V \to \mathbb{R}^+_0$ when the algorithm is executed on a stream S. The sum of all reduced weights is denoted by $W'_S = \sum_{e \in S} w'_S(e)$. For any edge in the stream, its reduced weight is non-negative and is unchanged by the processing of any subsequent edges. In particular, for a stream AB and any edge $e \in A$ (i.e., the edge e is present in the stream A), we have $w'_A(e) = w'_{AB}(e) \geq 0$. Hence, the sum of the reduced weights is a non-decreasing function, i.e., $W'_A \leq W'_{AB}$. The output matching of $\mathcal{ALG}^{\varepsilon}_{PS}$ on stream S is denoted by $\hat{M}(S)$.

Ghaffari and Wajc's analysis of the algorithm reveals the following key observations and results which we later use in our proofs.

▶ **Observation 4** (Ghaffari and Wajc [13]). *At any moment there are* $O\left(\frac{\log(1/\varepsilon)}{\varepsilon} \cdot n\right)$ *edges stored in* Stack *during the execution of* $\mathcal{ALG}^{\varepsilon}_{PS}$.

▶ **Proposition 5** (Ghaffari and Wajc [13]). *For any edge $e = \{u, v\}$ in a stream S, after the execution of $\mathcal{ALG}^{\varepsilon}_{PS}$, its weight is bounded as $w(e) \leq (1+\varepsilon) \cdot \big(\varphi_S(u) + \varphi_S(v)\big)$.*

▶ **Proposition 6** (Ghaffari and Wajc [13]). *Let $\varepsilon > 0$ and S be a stream of edges. Then, the following inequalities hold:*

$$w\big(M^*(S)\big) \geq W'_S,$$
$$w\big(\hat{M}(S)\big) \geq \frac{1}{1+4\varepsilon} \cdot W'_S = \frac{1}{2(1+4\varepsilon)} \sum_{v \in V} \varphi_S(v) \geq \frac{1}{2(1+4\varepsilon)(1+\varepsilon)} \cdot w\big(M^*(S)\big).$$

Note that Proposition 6 uses the important fact that $W'_S = \frac{1}{2}\sum_{v\in V} \varphi_S(v)$ as the potential of a vertex v is actually the sum of reduced weights of edges incident to v. Furthermore, its last inequality is due to Proposition 5 since each vertex in a matching is incident to at most one edge. Indeed, Proposition 6 shows that $\mathcal{ALG}^{\varepsilon}_{PS}$ is a $(2+\varepsilon)$-approximation streaming algorithm for MWM, and, by Observation 4, $\mathcal{ALG}^{\varepsilon}_{PS}$ uses space $O(\frac{\log(1/\varepsilon)}{\varepsilon} \cdot n)$ (in words).

3 $(2+\varepsilon)$-approximation Sliding Window Algorithm

In this section, we give a $(2+\varepsilon)$-approximation sliding window algorithm for MWM with space $\tilde{O}(\sqrt{nL})$, where L is the length of the sliding window.

Algorithm 2 MWM Sliding Window Algorithm.

Input: A stream S with a sliding window of length L
$\mathcal{A}$: $\mathcal{ALG}^{\varepsilon}_{PS}$ with sum of reduced weights W' and output matching $\hat{M}$.

Initialization:

1: Stack $\leftarrow$ an empty stack
2: $k \leftarrow 0$ ▷ Number of blocks
3: Parameter $s \leftarrow \left\lfloor \frac{\sqrt{n \cdot L \cdot \log 1/\varepsilon \cdot \log \sigma}}{\varepsilon} \right\rfloor$ for $\sigma = \frac{n}{2} \cdot w_{\max}/w_{\min}$.

Streaming:

4: **while** a new item e of the stream S is revealed **do**
5: Feed e to all existing instances of $\mathcal{A}$
6: Delete all instances of $\mathcal{A}$ which have processed more than L edges
7: Stack.Push(e)
8: **if** $|\text{Stack}| \geq s$ **then** ▷ Create new instances of $\mathcal{A}$
9: $k \leftarrow k+1$
10: Let $\mathcal{I}^k_1$ be a new instances of $\mathcal{A}$
11: Let $W'_{\text{prev}} \leftarrow 0, i \leftarrow 1$
12: **while** Stack is not empty **do** ▷ Process all edges in reverse order of arrival
13: $e' \leftarrow$ Stack.Pop and feed e' to $\mathcal{I}^k_i$
14: **if** $W'(\mathcal{I}^k_i) > (1+\varepsilon) \cdot W'_{\text{prev}}$ **then** ▷ $W'(\mathcal{I}^k_i)$ exceeds $(1+\varepsilon) \cdot W'(\mathcal{I}^k_{i-1})$
15: Create a new instance $\mathcal{I}^k_{i+1}$ as a copy of $\mathcal{I}^k_i$
16: $W'_{\text{prev}} \leftarrow W'(\mathcal{I}^k_i), i \leftarrow i+1$
17: **if** any instance of $\mathcal{A}$ exists **then**
18: **report** output matching of the instance that has processed the most edges
19: **else report** the maximum-weight matching of the edges in Stack

For brevity of notation, denote by $\mathcal{A}$ the Paz and Schwartzman algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$, which our algorithm (see Algorithm 2 for a listing) maintains several instances of. When the current edge e of the stream arrives, the algorithm feeds e to all existing instances of $\mathcal{A}$, then deletes any instance that has processed more than L edges, i.e., the ones that could return edges outside the sliding window. The edge e is subsequently pushed onto Stack.

When Stack has accumulated s edges, Algorithm 2 uses it to create several instances of $\mathcal{A}$: It first creates a new instance $\mathcal{I}_1$ of $\mathcal{A}$, then starts to pop the edges from Stack, processing the edges in reverse order of their arrival. When an edge e is popped it is fed into the last created instance $\mathcal{I}_i$ (initially $\mathcal{I}_1$). At any given moment, the algorithm stores the sum of reduced weights $W'(\mathcal{I}_{i-1})$ of the previous instance (initially set to 0). If the sum of reduced weights $W'(\mathcal{I}_i)$ of the latest instance $\mathcal{I}_i$ exceeds $(1+\varepsilon)\cdot W'(\mathcal{I}_{i-1})$, then a new instance $\mathcal{I}_{i+1}$ is created as a copy of $\mathcal{I}_i$. This procedure is repeated until Stack is empty again.

After processing edge e, the algorithm reports the matching computed by the instance of $\mathcal{A}$ which has processed the most edges of the current sliding window. If no instances have been created yet, then it reports an exact solution on the edges stored in Stack.

Overall, Algorithm 2 maintains multiple runs of $\mathcal{A}$, each fed with different suffixes of the sliding window. It uses Stack to implicitly partition the stream S into blocks $B_1, B_2, \ldots$ of s edges each, thus processing it block by block. Each block B_j is then processed, crucially in reverse order of arrival, feeding each edge into an initially empty instance $\mathcal{I}^j_1$ of $\mathcal{A}$. Then, copies $\mathcal{I}^j_i$ are created whenever the sum of reduced weights exceed a $(1+\varepsilon)$ factor of the previous copy. Once the block B_j has been processed entirely, the subsequent edges of the stream are fed to the instances $\mathcal{I}^j_1, \mathcal{I}^j_2, \ldots, \mathcal{I}^j_\ell$ in the natural arrival order. Note that the algorithm constructs the instances such that $\mathcal{I}^j_1$ only processes a single edge of the block B_j and $\mathcal{I}^j_\ell$ processes the entire block.

Intuitively, Algorithm 2 ensures that, as edges of the block start to fall outside of the sliding window, the oldest remaining instance is still a good approximation of the solution on the entire sliding window, i.e., consecutive runs of $\mathcal{A}$ are not too different in terms of output. Moreover, immediately after processing block B_j, it holds that $W'(\mathcal{I}^j_i) > (1+\varepsilon)\cdot W'(\mathcal{I}^j_{i-1})$ for all $1 < i \leq \ell$. Therefore, there are only logarithmically many runs of $\mathcal{A}$ per block.

In the following proofs, we use a notion $S(\mathcal{I})$ to denote a substream that is processed by the instance $\mathcal{I}$ of $\mathcal{A}$.

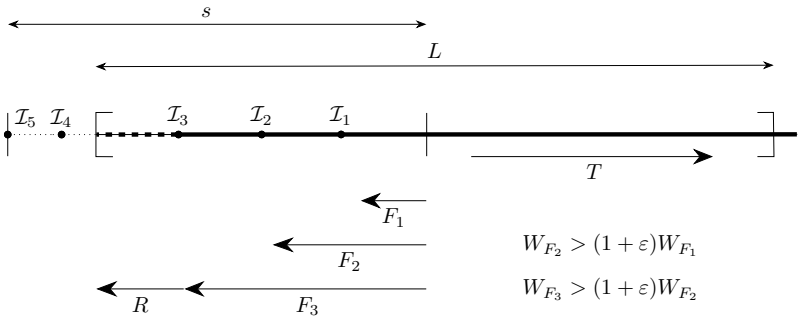


Figure 1 A schematic of a block of the algorithm. The notation here coincides with the notation used in the proof of Theorem 7. The window of length L is marked between the square brackets. There are five instances $\mathcal{I}_1, \ldots, \mathcal{I}_5$ created for the block of length s. The instance $\mathcal{I}_i$ processed the stream $\overleftarrow{F_i}T$. Note that $\mathcal{I}_4$ and $\mathcal{I}_5$ are already expired, thus they were deleted. The algorithm outputs the result of the instance $\mathcal{I}_3$ meaning the stream $\overleftarrow{F_3}T$. The proof of Theorem 7 will show that the omission of the remainder $\overleftarrow{R}$ does not compromise the output matching too much.

▶ **Theorem 7.** *There is a deterministic streaming sliding window algorithm for* Maximum-weight Matching *with an approximation factor* $2+\varepsilon$ *that uses* $O\left(\frac{\sqrt{n \cdot L \cdot \log 1/\varepsilon \cdot \log \sigma}}{\varepsilon}\right)$ *words of memory for any* $\varepsilon > 0$ *and* $\sigma = \frac{n}{2} \cdot w_{max}/w_{min}$.

Proof. We will prove that Algorithm 2 satisfies the assertion of the theorem. Let B_j be the oldest block of the stream which is still partially contained in the current sliding window E, i.e., E contains at least one edge of B_j and no edge of B_{j-1}. Let $\mathcal{I}_1^j, \ldots, \mathcal{I}_\ell^j$ be the instances created during the processing of block B_j. Note that each instance $\mathcal{I}_i^j$ processes the edges of B_j in reverse order. Thus, we consider B_j as a stream of edges ordered in reverse to the order in which they arrived. For clarity, we denote this as $\overleftarrow{B_j}$ and similarly for all relevant substreams of $\overleftarrow{B_j}$. Let $\overleftarrow{F_i}$ be the substream of $\overleftarrow{B_j}$ fed into the instance $\mathcal{I}_i^j$, i.e., $\overleftarrow{F_i} = S(\mathcal{I}_i^j) \cap \overleftarrow{B_j}$. Note that $\overleftarrow{F_1} \subseteq \cdots \subseteq \overleftarrow{F_\ell} = \overleftarrow{B_j}$.

Approximation. Let T be the stream of edges that arrive after the stream $\overleftarrow{B_j}$, i.e., $E \subseteq \overleftarrow{B_j}T$. First suppose that $E = \overleftarrow{B_j}T$. Then, Algorithm 2 returns the matching computed by the oldest instance $\mathcal{I}_\ell^j$ which has processed the whole stream $\overleftarrow{F_\ell}T$, i.e., all edges of E (as $\overleftarrow{F_\ell} = \overleftarrow{B_j}$). Thus, it returns a $(2+\varepsilon)$-approximation of the optimal solution.

Now, suppose that $E \subset \overleftarrow{B_j}T$. Let $\overleftarrow{F_i}T \subseteq E \subset \overleftarrow{F}_{i+1}T$. Note that such an i exists as $E \cap \overleftarrow{B_j} \neq \emptyset$ and $|\overleftarrow{F_1}| = 1$. Algorithm 2 returns a matching computed by the instance $\mathcal{I}_i^j$ that processed the stream $S(\mathcal{I}_i^j) = \overleftarrow{F}T$ for $\overleftarrow{F} = \overleftarrow{F_i}$. Let $\overleftarrow{R}$ be the substream of $\overleftarrow{F}_{i+1} \setminus \overleftarrow{F}$ such that E contains exactly the edges of the stream $\overleftarrow{F}\overleftarrow{R}T$. See Figure 1, for an illustration of the substreams processed by various instances $\mathcal{I}_i$.

Since $E \subset \overleftarrow{F}_{i+1}T$, it holds by construction of Algorithm 2 that $W'_{\overleftarrow{F}\overleftarrow{R}} \leq (1+\varepsilon) \cdot W'_{\overleftarrow{F}}$, where $W'_{\overleftarrow{F}\overleftarrow{R}}$ and $W'_{\overleftarrow{F}}$ are the sums of reduced weights computed by $\mathcal{A}$ on streams $\overleftarrow{F}\overleftarrow{R}$ and $\overleftarrow{F}$, respectively. For any vertex v, let $\Delta(v) := \varphi_{\overleftarrow{F}\overleftarrow{R}}(v) - \varphi_{\overleftarrow{F}}(v)$. Recall that φ is an increasing function by the construction of the algorithm, thus $\Delta(v) \geq 0$. Then, by the proportionality between the sum of reduced weights and the sum of potentials ($\sum_e w'(e) = 2\sum_v \varphi(v)$, see Proposition 6), we have the following upper bound:

$$\sum_{v \in V} \Delta(v) = \sum_{v \in V} \varphi_{\overleftarrow{F}\overleftarrow{R}}(v) - \varphi_{\overleftarrow{F}}(v) \leq \sum_{v \in V} (1+\varepsilon) \cdot \varphi_{\overleftarrow{F}}(v) - \varphi_{\overleftarrow{F}}(v) = \varepsilon \cdot \sum_{v \in V} \varphi_{\overleftarrow{F}}(v) \ .$$

We now claim that if we assign, for every $v \in V$, a weight $c(v) := (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}T}(v) + \Delta(v)\big)$, then we have a valid (weighted) vertex cover in the graph consisting of all edges in $\overleftarrow{F}\overleftarrow{R}T$, i.e., for each edge $e = \{u, v\} \in \overleftarrow{F}\overleftarrow{R}T$, it holds that $c(e) := c(u) + c(v) \geq w(e)$. Consider two cases. If $e \in \overleftarrow{F}T$, then we have $c(e) \geq (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}T}(v) + \varphi_{\overleftarrow{F}T}(u)\big) \geq w(e)$. Otherwise, $e \in \overleftarrow{R}$ and we have

$$\begin{aligned} c(e) &= (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}T}(v) + \varphi_{\overleftarrow{F}T}(u) + \varphi_{\overleftarrow{F}\overleftarrow{R}}(v) - \varphi_{\overleftarrow{F}}(v) + \varphi_{\overleftarrow{F}\overleftarrow{R}}(u) - \varphi_{\overleftarrow{F}}(u)\big) \\ &\geq (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}\overleftarrow{R}}(v) + \varphi_{\overleftarrow{F}\overleftarrow{R}}(u)\big) && (\overleftarrow{F} \text{ is a substream of } \overleftarrow{F}T) \\ &\geq w(e) \ . && \text{(by Proposition 5)} \end{aligned}$$

Thus, we get a valid vertex cover as required. Now, we can use this to show that the returned matching $\hat{M}(\overleftarrow{F}T)$ computed by $\mathcal{A}$ on the stream $\overleftarrow{F}T$ is a $(2+\varepsilon)$-approximation of the maximum weighted matching M^* of the sliding window E:

$$
\begin{aligned}
w(M^*) &= \sum_{e \in M^*} w(e) \leq \sum_{v \in V} c(v) && \text{(each vertex is incident to at most one edge in } M^*) \\
&= (1+\varepsilon) \sum_{v \in V} \big(\varphi_{\overleftarrow{F}T}(v) + \Delta(v)\big) \\
&\leq (1+\varepsilon) \sum_{v \in V} \big(\varphi_{\overleftarrow{F}T}(v) + \varepsilon \varphi_{\overleftarrow{F}}(v)\big) \\
&\leq (1+\varepsilon)^2 \sum_{v \in V} \varphi_{\overleftarrow{F}T}(v) && \text{(since } \varphi \text{ is monotonic)} \\
&\leq (1+3\varepsilon) \sum_{v \in V} \varphi_{\overleftarrow{F}T}(v) && \text{(since } \varepsilon^2 < \varepsilon \text{ for } 0 < \varepsilon < 1) \\
&\leq 2(1+3\varepsilon)(1+4\varepsilon) \cdot w\big(\hat{M}(\overleftarrow{F}T)\big) && \text{(by Proposition 6)} \\
&\leq (2+38\varepsilon) \cdot w\big(\hat{M}(\overleftarrow{F}T)\big) \ .
\end{aligned}
$$

Space. The sliding window E can be covered by $O\left(\frac{L}{s}\right)$ many blocks, as s is the block size. First, we bound the number of instances ℓ created for a block $\overleftarrow{B}_j$. Recall that edges from the block B_j processed by the instance $\mathcal{I}_i^j$ are the edges in $\overleftarrow{F}_i$, and $\overleftarrow{F}_1 \subseteq \cdots \subseteq \overleftarrow{F}_\ell = \overleftarrow{B}_j$. Furthermore, $\overleftarrow{F}_1 = \{e'\}, W'_{\overleftarrow{F}_1} = w(e') \geq w_{\min}$, and $W'_{\overleftarrow{F}_{i+1}} > (1+\varepsilon) \cdot W'_{\overleftarrow{F}_i}$ for all $i < \ell$. Thus,

$$
(1+\varepsilon)^{\ell-1} \cdot w_{\min} \leq (1+\varepsilon)^{\ell-1} \cdot W'_{\overleftarrow{F}_1} < W'_{\overleftarrow{F}_\ell} \leq w\big(\hat{M}(\overleftarrow{F}_\ell)\big) \leq \frac{n}{2} \cdot w_{\max}.
$$

By rearranging, we get $\ell = O(\log_{1+\varepsilon} \sigma) = O\left(\frac{1}{\varepsilon} \cdot \log \sigma\right)$. By Observation 4, each instance of $\mathcal{A}$ stores $O\left(\frac{n \log(1/\varepsilon)}{\varepsilon}\right)$ edges. Thus, at any moment, all existing instances of $\mathcal{A}$ store $O\left(\frac{L}{s} \cdot \frac{\log \sigma}{\varepsilon} \cdot \frac{n \log(1/\varepsilon)}{\varepsilon}\right)$ many edges.

Note that we additionally need to store the edges of at most one block (stored in Stack), i.e., at most s edges. Overall, we need to store at most $O\left(\frac{L}{s} \cdot \frac{\log \sigma}{\varepsilon} \cdot \frac{n \log(1/\varepsilon)}{\varepsilon} + s\right)$ edges. Setting s to $\left\lfloor \frac{\sqrt{n \cdot L \cdot \log 1/\varepsilon \cdot \log \sigma}}{\varepsilon} \right\rfloor$ gives us the final space bound in words of memory. ◀

▶ Remark. Assuming that ε is constant and that σ is polynomial in n, we obtain an algorithm that uses $\tilde{O}(\sqrt{nL})$ space. This is $o(n^2)$ space as long as $L = \tilde{o}(n^3)$. If, additionally, the input graph of each window is simple, we have that $L = O(n^2)$ (a simple graph always has $O(n^2)$ edges) and a space bound of $O\left(n\sqrt{n} \cdot \frac{\sqrt{\log 1/\varepsilon \cdot \log \sigma}}{\varepsilon}\right)$, which simplifies to $\tilde{O}\left(n\sqrt{n}\right)$.

We can easily adapt the algorithm to the (unweighted) MM problem. More specifically, the Paz-Schwartzman algorithm becomes the Greedy matching algorithm, while the sum of reduced weights simply becomes the size of the Greedy matching obtained. While the approximation factor remains $2+\varepsilon$, the matchings of the instances now store $O(n)$ edges instead of $O\left(\frac{n \log(1/\varepsilon)}{\varepsilon}\right)$. Also, $\sigma = \frac{n}{2}$. Then, by setting s to $\left\lfloor \sqrt{\frac{n \cdot L \cdot \log n}{\varepsilon}} \right\rfloor$, we obtain a better memory bound for the algorithm. This adaptation yields the following result:

▶ **Theorem 8.** *There is a deterministic streaming sliding window algorithm for MM with an approximation factor of $(2+\varepsilon)$ that uses $O\left(\sqrt{\frac{n \cdot L \cdot \log n}{\varepsilon}}\right)$ words of memory for any $\varepsilon > 0$.*

4 $(3+\varepsilon)$-approximation Sliding Window Algorithm

In this section, we give a $(3+\varepsilon)$-approximation semi-streaming sliding window algorithm by applying the smooth histogram technique [6] in a similar manner as Biabani et al. [4]. We start with our definition of a *refined lookahead algorithm* which we use to describe a sliding window algorithm. Then, we show that $\mathcal{ALG}^{\varepsilon}_{PS}$ is a refined lookahead algorithm; thus, obtaining the sliding window algorithm for MWM.

▶ **Definition 9** ($(f,\alpha_1,\alpha_2,\beta)$-refined lookahead algorithm). *Let $\beta \in (0,1)$, $\alpha_1,\alpha_2 \geq 1$ and, for a ground set X, let $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. We say a streaming algorithm $\mathcal{ALG}$ with two outputs O_1, O_2 is a* $(f,\alpha_1,\alpha_2,\beta)$-refined lookahead *algorithm if the following holds for any stream S of items of the set X:*

1. *$O_1(S) \leq f(S) \leq \alpha_1 \cdot O_1(S)$, i.e., the first output is an α_1-approximation of f.*
2. *For any partitioning of S into three disjoint sub-streams A, B, and C with $O_1(B) \geq (1-\beta)\cdot O_1(AB)$, we have $O_2(BC) \leq f(ABC) \leq \alpha_2 \cdot O_2(BC)$, i.e., if the first output on the substream AB is similar to the first output on the substream B then the second output on the substream BC is an α_2-approximation of f on the whole stream $S = ABC$.*

Observe that if $O_1 = O_2$ and $\alpha_1 = \alpha_2 = \alpha$ then we retrieve the standard definition of a (f,α,β)-lookahead algorithm as given by Biabani et al. (see Definition 3). Our refined lookahead algorithm is also similarly turned into a sliding window algorithm. In essence, the algorithm simulates runs of a traditional streaming algorithm on suffixes of the current sliding window. It maintains runs on suffixes such that the value of O_1 of any two consecutive runs are not too different, while the value of O_1 of any non-consecutive runs are sufficiently different so as to ensure that at most a logarithmic number of runs are required at any point of time. The second output O_2 is a solution which, given the smoothness assumptions of the runs, is always guaranteed to be an α_2-approximation of the next oldest run. Details of the algorithm and the proof of the following theorem are provided in Appendix B.

▶ **Theorem 10.** *Let $0 < \beta < 1$ and $\alpha_1,\alpha_2 \geq 1$, S be a stream of items from a set X, and $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. Suppose there exists a $(f,\alpha_1,\alpha_2,\beta)$-refined lookahead algorithm that uses at most s words of memory. Then, there is a sliding window algorithm that maintains an α_2-approximation of f using $O\big(\frac{1}{\beta}\cdot s\log(\alpha_1\sigma)\big)$ words of memory for $\sigma = f(S)/f_{\min}$ where $f_{\min} = \min\{f(e) : e \in S\}$.*

We will now apply Definition 9 to algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$. To this end, we consider the first output O_1 as the sum of reduced weights W'_S, the second output O_2 as the weight of the returned matching $w\big(\hat{M}(S)\big)$, and function f as the weight of a maximum-weight matching $\mathsf{MWM}(S)$. In fact, we prove in Theorem 11 that this indeed yields a $\big(\mathsf{MWM},(2+2\varepsilon),(3+20\varepsilon),\beta\big)$-refined lookahead algorithm. Hence, the algorithm given by Theorem 10 with $\mathcal{ALG}^{\varepsilon}_{PS}$ is a $(3+\varepsilon)$-approximation semi-streaming sliding window algorithms for MWM.

▶ **Theorem 11.** *Let $0 < \varepsilon \leq \frac{1}{10}$ and $0 < \beta \leq \frac{\varepsilon}{9}$. The algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$ is a $\big(MWM,(2+2\varepsilon),(3+20\varepsilon),\beta\big)$-refined lookahead algorithm.*

To prove Theorem 11, we follow the approach of Biabani et al. [4]. Let an input stream S be partitioned into three substreams ABC. They split the maximum matching of the stream $M^* = M^*(ABC)$ into two parts M^*_{AB} and M^*_C where $M^*_{AB} := M^* \cap AB$ is the restriction of M^* to the edges in AB, analogously for the substream C. Biabani et al. then bound the weights of these two parts separately. To this end, they use the notion of a *critical subgraph*.

▶ **Definition 12** (Critical Subgraph [4]). *Consider a graph G specified by a stream S of edges. Let A, B, C be disjoint substreams of S such that $S = ABC$. Then, the* critical subgraph *of G with respect to the maximum matching $M^*(ABC)$ and the substreams A, B, C is the subgraph $H = (V_H, E_H)$ such that*

- $E_H := \{e \in B \mid e \text{ is adjacent to two edges in } M^*_C\}$.
- $V_H := V(E_H)$, *i.e., V_H is the set of endpoints of the edges in E_H.*

Biabani et al. use the critical subgraph to bound the weights of M^*_{AB} and M^*_C in terms of the weight of the matching returned by the algorithm $w\big(\hat{M}\big)$ (Lemmas 13 and 14 in their work [4]). In our analysis, in particular, in Lemmas 15 and 16, we use the same ideas to bound the weights of M^*_{AB} and M^*_C in terms of sums of reduced weights computed by the algorithm instead.

Before stating and proving Lemmas 15 and 16, we present the following auxiliary lemma already proved by Biabani et al. in the exact formulation as we need it. We highlight that their proof holds for any run of $\mathcal{ALG}^{\varepsilon}_{PS}$ on an arbitrary stream.

▶ **Lemma 13** (Biabani et al. [4], Lemma 15). *For any stream AB,*

$$(1+\varepsilon) \cdot \sum_{v \in V_H} \varphi_{AB}(v) \geq \sum_{e \in E_H} w'_B(e) \ .$$

For the statement of the next auxiliary lemma, we need the following notion. Let S be a stream of edges. For an edge $e \in S$, we define the set $P_S(e)$ as the set of edges incident to e (including e) arriving no later than e, i.e, $P_S(e) = \{e' \in S \mid e' \cap e \neq \emptyset, t_{e'} \leq t_e\}$, where, for any edge f, t_f is the arrival time of edge f. Biabani et al. [4] showed that the weight of any edge e can be bounded by the sum of the reduced weights of the edges in $P_S(e)$ (up to a $(1+\varepsilon)$ factor).

▶ **Lemma 14** (Biabani et al. [4], Lemma 5). *For each edge $e \in S$,*

$$w(e) \leq (1+\varepsilon) \sum_{e' \in P_S(e)} w'_S(e).$$

With that, we can finally prove our analogous lemmas of Biabani et al.'s Lemmas 13 and 14 [4] which bound $w\big(M^*_{AB}\big)$ and $w\big(M^*_C\big)$, respectively.

▶ **Lemma 15** (Analogue of Lemma 13, [4]). *For any stream ABC,*

$$w\big(M^*_{AB}\big) \leq 2(1+\varepsilon) \cdot W'_{AB} - \sum_{e \in E_H} w'_B(e).$$

Proof. By definition, we have $w\big(M^*_{AB}\big) = \sum_{e \in M^*_{AB}} w(e)$. Let $e = \{u, v\} \in M^*_{AB}$. Note that the vertices u and v are not in V_H. Thus, we can bound the sum as follows.

$$\begin{aligned} w\big(M^*_{AB}\big) &\leq (1+\varepsilon) \sum_{v \in V \setminus V_H} \varphi_{AB}(v) && \text{by Proposition 5} \\ &= (1+\varepsilon) \left(\sum_{v \in V} \varphi_{AB}(v) - \sum_{v \in V_H} \varphi_{AB}(v) \right) \end{aligned}$$

By Proposition 6 and by Lemma 13, we have

$$\sum_{v \in V} \varphi_{AB}(v) = 2W'_{AB} \qquad \text{and} \qquad (1+\varepsilon) \sum_{v \in V_H} \varphi_{AB}(v) \geq \sum_{e \in E_H} w'_B(e).$$

Thus, we can conclude that

$$w\big(M^*_{AB}\big) \leq 2(1+\varepsilon) \cdot W'_{AB} - \sum_{e \in E_H} w'_B(e). \qquad \blacktriangleleft$$

▶ **Lemma 16** (Analogue of Lemma 14, [4]). *For any stream ABC,*

$$w\big(M^*_C\big) \leq 2(1+\varepsilon)\cdot W'_{BC} - (1+\varepsilon)\sum_{e\in B\setminus E_H} w'_B(e).$$

Proof. First, when considering a run of the algorithm on BC, by Lemma 14, we obtain

$$w\big(M^*_C\big) = \sum_{e\in M^*_C} w(e) \leq (1+\varepsilon)\sum_{e\in M^*_C}\sum_{e'\in P(e)} w'_{BC}(e')\ .$$

Observe that any edge $e\in BC$ is incident to at most two edges of M^*_C, and the edges of $B\setminus E_H$ are incident to at most one edge of M^*_C. Hence, we can rewrite the previous double sum as follows:

$$\begin{aligned}\sum_{e\in M^*_C}\sum_{e'\in P(e)} w'_{BC}(e') &\leq 2\cdot\sum_{e\in BC} w'_{BC}(e) - \sum_{e\in B\setminus E_H} w'_{BC}(e)\\ &= 2\cdot W'_{BC} - \sum_{e\in B\setminus E_H} w'_{BC}(e)\ ,\end{aligned}$$

which implies the result. ◀

Now, we are ready to prove Theorem 11, i.e., $\mathcal{ALG}^{\varepsilon}_{PS}$ is a $\big(\mathsf{MWM}, (2+2\varepsilon), (3+20\varepsilon), \beta\big)$-refined lookahead algorithm for suitable parameters ε and β.

Proof of Theorem 11. We recall that we consider a version of $\mathcal{ALG}^{\varepsilon}_{PS}$ such that the first output is the sum of reduced weight W' and the second output is the weight of the computed matching $w\big(\hat{M}\big)$. First, by Proposition 6, we get that for any stream S it holds that $W'_S \leq w\big(M^*(S)\big) \leq 2(1+\varepsilon)\cdot W'_S$. Thus, it remains to prove that for any stream ABC, given that $W'_B \geq (1-\beta)\cdot W'_{AB}$, the maximum matching $M^* = M^*(ABC)$ is such that $w\big(M^*\big) \leq (3+20\varepsilon)\cdot w\big(\hat{M}(BC)\big)$.

$$\begin{aligned}
w\big(M^*\big) &\leq 2(1+\varepsilon)\cdot W'_{AB} + 2(1+\varepsilon)\cdot W'_{BC} - W'_B && \text{by Lemmas 15 and 16}\\
&\leq \frac{2(1+\varepsilon)}{1-\beta}\cdot W'_B + 2(1+\varepsilon)\cdot W'_{BC} - W'_B && \text{by } W'_B \geq (1-\beta)\cdot W'_{AB}\\
&\leq (1+3\varepsilon)\cdot W'_B + 2(1+\varepsilon)\cdot W'_{BC} && \text{since } \beta \leq \frac{\varepsilon}{9}\\
&\leq (3+5\varepsilon)\cdot W'_{BC} && \text{by } W' \text{ being non-decreasing}\\
&\leq (3+5\varepsilon)(1+4\varepsilon)\cdot w\big(\hat{M}(BC)\big) && \text{by Proposition 6}\\
&\leq (3+20\varepsilon)\cdot w\big(\hat{M}(BC)\big) && \text{since } \varepsilon \leq \frac{1}{10}
\end{aligned}$$ ◀

Theorems 10 and 11 together then imply our main result.

▶ **Theorem 2.** *There is a deterministic streaming sliding window algorithm for Maximum-weight Matching with an approximation factor $3+\varepsilon$ that uses $O\left(\frac{\log(1/\varepsilon)}{\varepsilon^2}\cdot n\log\sigma\right)$ words of memory, for any $0<\varepsilon\leq 0.1$ and $\sigma = \frac{n}{2}\cdot w_{max}/w_{min}$.*

▶ Remark. Our $(3+\varepsilon)$-approximation algorithm for MWM yields the $(3+\varepsilon)$-approximation algorithm for MM by Crouch et al. [7] when $\mathcal{ALG}^{\varepsilon}_{PS}$ is replaced with the Greedy matching algorithm (the sum of reduced weights becomes the size of the matching). The hard instance of their algorithm also holds for our algorithm.

5 Conclusion

In this paper, we gave two algorithms for MWM in the sliding window model. Our first algorithm has an approximation factor of $2+\varepsilon$ and uses space $\tilde{O}(\sqrt{nL})$, and our second algorithm has an approximation factor of $3+\varepsilon$ and uses semi-streaming space. The approximation factor of our semi-streaming algorithm matches the approximation factor of the best semi-streaming sliding window algorithm known for (unweighted) MM [7].

Regarding the semi-streaming space regime, since further improvements in the approximation factor would imply improvements for (unweighted) MM, the most natural direction for future research is to make further progress on the unweighted version of the problem first. Is there a 2.99-approximation semi-streaming space sliding window algorithm for MM?

While the known lower bounds for MM for one-pass streaming algorithms in the insertion-only model also apply to the sliding window model, no stronger lower bounds for the sliding window model are known. Can we prove a lower bound on the approximation factor of sliding window algorithms for MM that use semi-streaming space and are stronger than what is currently known for the insertion-only model, i.e., stronger than $1+\ln(2)$ [15]?

References

1 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. 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 1345–1364. SIAM, 2016. doi:10.1137/1.9781611974331.ch93.

2 Sepehr Assadi and Vihan Shah. An asymptotically optimal algorithm for maximum matching in dynamic streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.9.

3 Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approxmation algrithms in memoriam: Shimon even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463, 2004. doi:10.1145/1041680.1041683.

4 Leyla Biabani, Mark de Berg, and Morteza Monemizadeh. Maximum-Weight Matching in Sliding Windows and Beyond. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, 2021.

5 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering problems on sliding windows. In *SODA*, 2016.

6 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *FOCS 2007*, 2007.

7 Michael S. Crouch, Andrew McGregor, and Daniel M. Stubbs. Dynamic graphs in the sliding-window model. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms – ESA 2013 – 21st Annual European Symposium, Sophia Antipolis, France, September 2–4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013. doi:10.1007/978-3-642-40450-4_29.

8 Michael S. Crouch and Daniel Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *APPROX-RANDOM*, 2014.

9 Jacques Dark and Christian Konrad. Optimal lower bounds for matching and vertex cover in dynamic graph streams. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CCC.2020.30.

10 Mayur Datar, A. Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31:1794–1813, 2002.

11 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *ArXiv*, abs/0907.0305, 2011.

12 Joan Feigenbaum, Sampath Kannan, Andrew Mcgregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348:207–216, 2005.

13 Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming $(2 + \epsilon)$-approximate matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 13:1–13:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.13.

14 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.

15 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 1874–1893. SIAM, 2021. doi:10.1137/1.9781611976465.112.

16 Mikhail Kapralov. Better bounds for matchings in the streaming model. In *SODA*, 2013.

17 Christian Konrad. Maximum matching in turnstile streams. 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 840–852. Springer, 2015. doi:10.1007/978-3-662-48350-3_70.

18 Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005. doi:10.1007/11538462_15.

19 Ami Paz and Gregory Schwartzman. A $(2 + \epsilon)$-approximation for maximum weight matching in the semi-streaming model. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2153–2161. SIAM, 2017. doi:10.1137/1.9781611974782.140.

20 Yanhao Wang, Yuchen Li, and Kian-Lee Tan. Coresets for minimum enclosing balls over sliding windows. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

21 Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012. doi:10.1007/s00453-010-9438-5.

A Hard Instance for Paz and Schwartzman's Algorithm

In this section, we show that the Paz and Schwartzman's algorithm and its monotonic version are no better than $(\mathsf{MWM}, 3.5, \beta)$-lookahead algorithms. The definition of a lookahead algorithm given by Biabani et al. (Definition 3) together with the Paz and Schwartzman's algorithm thus cannot be used to improve upon the approximation factor of 3.5.

Recall that a lookahead algorithm relies on the smoothness of the algorithm's output. More formally, an (f, α, β)-lookahead algorithm $\mathcal{ALG}$ satisfies the condition that for any stream ABC, if $\mathcal{ALG}(B) \geq (1-\beta)\cdot\mathcal{ALG}(AB)$ then $f(ABC) \leq \alpha\cdot\mathcal{ALG}(BC)$ (see Definition 3).

In other words, if the algorithm $\mathcal{ALG}$ outputs similar results on the streams B and AB then the algorithm's output on BC is required to be an α-approximation of the objective value $f(ABC)$ of the whole stream ABC.

We will present a graph G whose edges are divided into three substreams A, B and C such that $\mathcal{ALG}^{\varepsilon}_{PS}$ outputs matchings of the same weight on substreams AB and B, while the outputted matching on substream BC is roughly a 3.5-approximation of a maximum-weight matching of the entire stream ABC. The graph G is such that even if we modified $\mathcal{ALG}^{\varepsilon}_{PS}$ to return maximum-weight matchings among the edges stored in Stack then the same properties still hold. Thus, the hard instance is also hard for the monotonic version of the algorithm. The graph G is depicted in Figure 2.

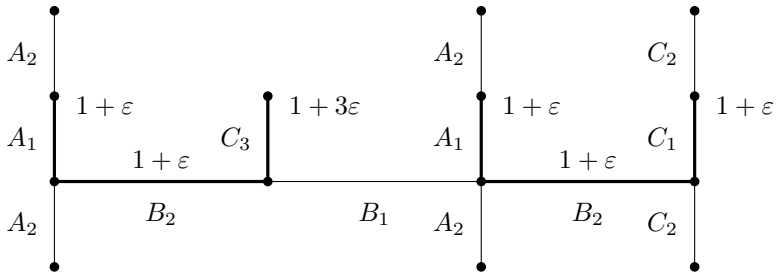


Figure 2 The edges of the graph G are divided into substreams A, B and C. The order of the edges within the substreams is indicated by subscripts (the order of the edges with the same subscript is not important). The thin edges have unit weight and the thick edges have the indicated larger weights.

Matchings computed on AB and B. First, we analyze $\mathcal{ALG}^{\varepsilon}_{PS}$ separately on the substreams A and B. See Figure 3 for the values of the reduced weights and potentials computed by the algorithm.

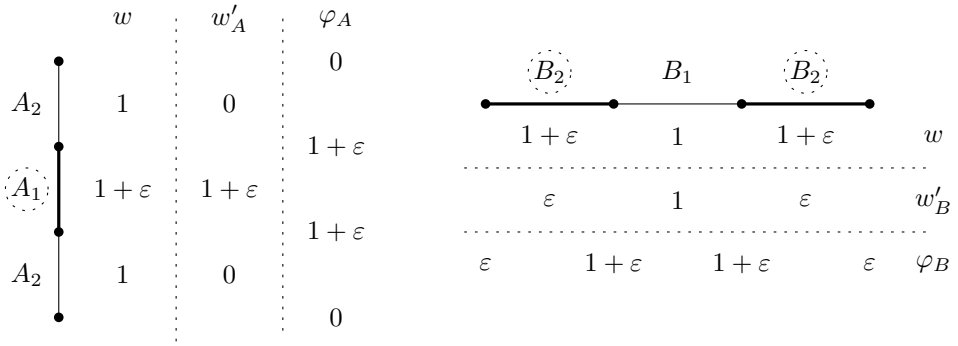


Figure 3 Reduced weights and potentials computed by $\mathcal{ALG}^{\varepsilon}_{PS}$ when run separately on substreams A and B. Recall that substream A consists of two paths, while only one of them is depicted here. The edges outputted by the runs of the algorithm are marked by dotted circles.

Observe that the substream A consists of two disjoint paths of length three. While only one of them is shown in Figure 3, the algorithm computes the same reduced weights and potentials for both paths.

We now analyze the execution of the algorithm on substream AB. To this end, consider the moment when the substream A has been fully processed and substream B begins. Observe that each edge of B is now incident to a single vertex with potential $1+\varepsilon$. Thus, by the

construction of the algorithm, none of the edges of B are pushed onto Stack. These edges therefore have reduced weights zero and cannot be outputted by the algorithm. Furthermore, when run on AB, the algorithm outputs the two edges in A_1, i.e., $w\big(\hat{M}(AB)\big) = 2 + 2\varepsilon$, which are the only two edges pushed onto Stack.

As established in Figure 3, when the algorithm runs only on the substream B, it outputs the two edges in B_2, i.e., $w\big(\hat{M}(B)\big) = 2 + 2\varepsilon$. Hence, we have that $w\big(\hat{M}(B)\big) = w\big(\hat{M}(AB)\big)$. It follows that the stream ABC satisfies the condition $w\big(\hat{M}(B)\big) \geq (1-\beta) \cdot w\big(\hat{M}(AB)\big)$, for any value of $\beta \geq 0$, as required by the definition of a lookahead algorithm.

Matching computed on BC. Now, we analyze the execution of the algorithm on the substream BC. At the time when the substream C begins, the reduced weights of edges in B and the current potentials of the incident vertices are the same as when the algorithm is run only on the substream B – see Figure 3 for these values. See Figure 4, for the reduced weights of the edges in C when we run the algorithm on the substream BC.

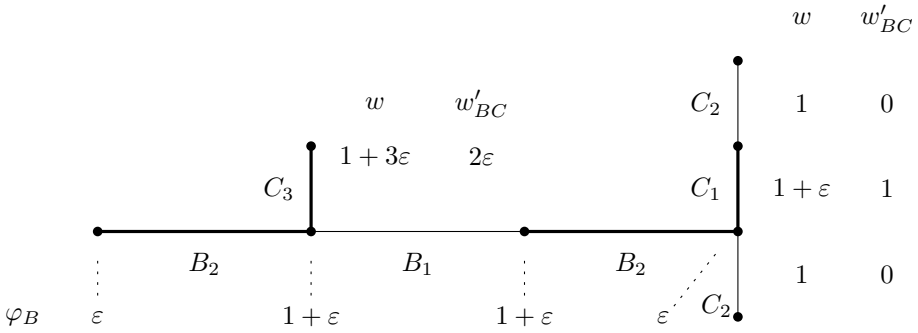


Figure 4 The reduced weights of the edges in C after the execution on the substream BC and the potentials of the vertices incident to the edges in B at the time when the substream B is processed.

By the end of the execution, only the two edges in C_1 and C_3 are pushed onto Stack since the edges in C_2 have reduced weights zero. The algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$ outputs a greedy matching of the edges pushed onto Stack (in the reverse order they arrived). In particular, it outputs the edges in C_1 and C_3 and they block all edges in B. Hence, $w\big(\hat{M}(BC)\big) = 2 + 4\varepsilon$. Observe further that these edges constitute a maximum-weight matching among the edges pushed onto Stack.

Maximum-weight Matching and Approximation Factor. First, observe that the unique maximum-weight matching in G consists of all the edges that have an endpoint of degree 1 (the edges in A_2, C_2, and C_3) and is thus of weight $7 + 3\varepsilon$. Since $w\big(\hat{M}(BC)\big) = 2 + 4\varepsilon$, we conclude that it is not possible for $\mathcal{ALG}^{\varepsilon}_{PS}$ to yield a $(\mathsf{MWM}, 3.5 - \Delta, \beta)$-lookahead algorithm, for any constant $\Delta > 0$ and suitable parameter β.

B More on Refined Lookahead Algorithms

In this section, we will prove Theorem 10. To this end, for convenience, we restate the definition of refined lookahead algorithms first.

▶ **Definition 9** ($(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm). *Let $\beta \in (0,1)$, $\alpha_1, \alpha_2 \geq 1$ and, for a ground set X, let $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. We say a streaming algorithm $\mathcal{ALG}$ with two outputs O_1, O_2 is a* $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead *algorithm if the following holds for any stream S of items of the set X:*

1. $O_1(S) \leq f(S) \leq \alpha_1 \cdot O_1(S)$, *i.e., the first output is an* α_1*-approximation of* f*.*
2. *For any partitioning of* S *into three disjoint sub-streams* A*,* B*, and* C *with* $O_1(B) \geq (1-\beta) \cdot O_1(AB)$*, we have* $O_2(BC) \leq f(ABC) \leq \alpha_2 \cdot O_2(BC)$*, i.e., if the first output on the substream* AB *is similar to the first output on the substream* B *then the second output on the substream* BC *is an* α_2*-approximation of* f *on the whole stream* $S = ABC$*.*

Algorithm 3 Lookahead Sliding Window Algorithm.

Input: A stream S with a sliding window of length L
$\mathcal{A}$: a $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm with outputs O_1 and O_2

Initialization:

1: Let $k \leftarrow 0$ be the number of instances

Streaming:

2: **while** a new item e of the stream S is revealed **do**
3: Create an instance $\mathcal{I}_{k+1}$ of $\mathcal{A}$
4: Feed e into all existing instances $\mathcal{I}_1, \ldots, \mathcal{I}_{k+1}$
5: $i \leftarrow 1$
6: **while** $i < k$ **do** ▷ Deleting instances with similar value of O_1
7: Let $j > i$ be the largest index for which $O_1(\mathcal{I}_j) \geq (1-\beta) \cdot O_1(\mathcal{I}_i)$
8: **if** no such j exists **then** $j \leftarrow i+1$
9: Delete instances $\mathcal{I}_r$ for each $i < r < j$
10: $i \leftarrow j$
11: Let $\mathcal{I}_{>1}$ be the next existing instance after $\mathcal{I}_1$ ▷ $\mathcal{I}_1$ was not deleted
12: **if** $\mathcal{I}_{>1}$ does not exist **then** continue to line 15
13: **if** $|S(\mathcal{I}_{>1})| \geq L$ **then** ▷ $|S(\mathcal{I}_{>1})|$ is the number of items fed into $\mathcal{I}_{>1}$
14: Delete $\mathcal{I}_1$
15: Renumber the instances and let k be the number of remaining ones
16: **if** $|S(\mathcal{I}_1)| = L$ **then report** $O_2(\mathcal{I}_1)$
17: **else report** $O_2(\mathcal{I}_2)$

Let e be the current item of the stream being processed by Algorithm 3 and let E be the current sliding window consisting of the L most recently processed items (including e). While processing e, the algorithm first creates a new instance $\mathcal{I}_{k+1}$ (called a bucket in Biabani et al. [4]) of $\mathcal{A}$. Then, e is fed into all existing instances $\mathcal{I}_1, \ldots, \mathcal{I}_{k+1}$. Next, starting from the oldest instance $\mathcal{I}_1$, only its newest similar instance, determined by O_1 (Item 2 of Definition 9), is kept and every other instance in between is deleted. Whether a newest similar instance exists or not, the process is then repeated with the next oldest remaining instance until reaching the newest instance. Note that the oldest and newest instances, $\mathcal{I}_1$ and $\mathcal{I}_{k+1}$ respectively, are never deleted by this process. However, if the number of items fed into the second oldest remaining instance $\mathcal{I}_{>1}$ is at least L, i.e., the current sliding window E is fully contained in the stream $S(\mathcal{I}_{>1})$ of edges processed by $\mathcal{I}_{>1}$, then $\mathcal{I}_1$ is deleted. The instances are then renumbered to $\mathcal{I}_1, \ldots, \mathcal{I}_k$, from the oldest one to the newest, such that k is the number of remaining instances. At this stage, the sliding window E is sandwiched between streams $S(\mathcal{I}_1)$ and $S(\mathcal{I}_2)$. Finally, after processing the item, if the current sliding window contains exactly the edges processed by $\mathcal{I}_1$, then the algorithm reports the second output O_2 of the instance $\mathcal{I}_1$ as the solution, otherwise it reports $O_2(\mathcal{I}_2)$.

In essence, the instances of $\mathcal{A}$ created by Algorithm 3 simulate runs of a traditional streaming algorithm on suffixes of the current sliding window. Note that the oldest run always contains all items of the sliding window and potentially some additional ones. The

idea is to maintain runs on suffixes such that the value of O_1 of any two consecutive runs are not too different, while the value of O_1 of any non-consecutive runs are sufficiently different so as to ensure that at most a logarithmic number of instances of $\mathcal{A}$ is used at any point of time.

This idea is exactly captured when $\mathcal{A}$, with two outputs O_1 and O_2, is a $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm (which applies the smooth histogram technique by Braverman and Ostrovsky [6]). The first output O_1 is used to determine how often a run on a suffix should be maintained, which depends on the smoothness criteria given by Item 2 of Definition 9. The second output O_2 is a solution which, given the smoothness assumptions of the runs, is always guaranteed to be an α_2-approximation of the next oldest run. We highlight that the smoothness assumptions are only guaranteed to hold for consecutive runs whose suffixes differ by more than one item. Then, for a stream S of items from a set X and a non-decreasing function $f : 2^X \to \mathbb{R}^+$, the number of runs is at most logarithmic in n as long as $\sigma_f(S) = f(S)/f_{\min}$, where $f_{\min} = \min\{f(e) : e \in S\}$, is polynomial in n. We prove this formally in Theorem 10.

▶ **Theorem 10.** *Let $0 < \beta < 1$ and $\alpha_1, \alpha_2 \geq 1$, S be a stream of items from a set X, and $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. Suppose there exists a $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm that uses at most s words of memory. Then, there is a sliding window algorithm that maintains an α_2-approximation of f using $O\big(\frac{1}{\beta} \cdot s \log(\alpha_1 \sigma)\big)$ words of memory for $\sigma = f(S)/f_{\min}$ where $f_{\min} = \min\{f(e) : e \in S\}$.*

Proof. We prove that Algorithm 3 satisfies the assertion of the theorem. Let $\mathcal{A}$ be the used $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm with the outputs O_1 and O_2.

Approximation. Let E be the sliding window at any instance of the algorithm, i.e., the set of the L most recently processed items. The algorithm ensures that E is sandwiched between streams of items fed to $\mathcal{I}_1$ and $\mathcal{I}_2$, i.e., $S_2 \subseteq E \subseteq S_1$ for $S_i = S(\mathcal{I}_i), i \in \{1, 2\}$. We are now in one of two cases, either the items of S_1 and S_2 differ by exactly one item or more than one item.

In the former case, the algorithm asserts that $|S_2| < L$, otherwise S_1 would have been deleted, and therefore the items of S_1 are exactly those of the sliding window E, i.e., $|S_1| = L$. The reported solution is then always $O_2(S_1) = O_2(E)$ which by Item 2 of Definition 9 (consider the case when $E = ABC = BC$) is trivially an α_2-approximation of $f(E)$.

In the latter case, the algorithm would have, at some point, deleted instances which caused $\mathcal{I}_1$ and $\mathcal{I}_2$ to become consecutive instances (Line 9 of Algorithm 3). Consider the time t^* when they first became adjacent. Let S_1^* and S_2^* be the streams processed by $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively, in the time t^*. The algorithm asserts that $O_1(S_2^*) \geq (1 - \beta) \cdot O_1(S_1^*)$. Let C be the remaining items fed into the instances such that $S_1 = S_1^* C$ and $S_2 = S_2^* C$. Then, by Item 2 of Definition 9 and f being non-decreasing,

$$O_2(S_2) \leq f(S_2) \leq f(E) \leq f(S_1) \leq \alpha_2 \cdot O_2(S_2).$$

Hence, we have that, $O_2(S_2)$, is an α_2-approximation of $f(E)$. Now, if $|S_1| \neq L$ the solution reported is $O_2(S_2)$, otherwise $|S_1| = L$ and the solution reported is $O_2(S_1) = O_2(E)$. We conclude that in either case an α_2-approximation of $f(E)$ is reported.

Space. Let k be the maximum number of instances stored by the algorithm after processing an item. After the process of deleting and renumbering the instances, the algorithm ensures that $O_1(\mathcal{I}_{i+2}) < (1 - \beta) \cdot O_1(\mathcal{I}_i)$ holds for any instances $\mathcal{I}_i$ and $\mathcal{I}_{i+2}$. Thus for the largest odd number k' not exceeding k,

$$(1 + \beta)^{\frac{k'-1}{2}} O_1(\mathcal{I}_{k'}) < O_1(\mathcal{I}_1).$$

Recall that $\frac{f(S(\mathcal{I}_1))}{f(S(\mathcal{I}_{k'}))} \leq \sigma$. Then, by Item 1 of Definition 9, we have that $\frac{O_1(\mathcal{I}_1)}{O_1(\mathcal{I}_{k'})} \leq \alpha_1\sigma$. It follows that

$$\frac{k'-1}{2} < \log_{1+\beta}(\alpha_1\sigma) \qquad \text{and} \qquad k' = O\left(\frac{1}{\beta} \cdot \log(\alpha_1\sigma)\right).$$

This implies the result since there are only ever $k+1 \leq k'+2$ instances of $\mathcal{A}$, each of which uses at most s words of memory. ◀

A motivating example of the refined lookahead definition is exactly the Paz-Schwartzman algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$ with the first output O_1 as the sum of reduced weights W'_S, the second output O_2 as the weight of the returned matching $w\big(\hat{M}(S)\big)$, and function f as the weight of a maximum-weight matching $\mathsf{MWM}(S)$. Now, consider the graph given in Appendix A (see Figure 2). We have that $w\big(\hat{M}(AB)\big) = w\big(\hat{M}(B)\big) = 2+2\varepsilon$, $W'_{AB} = 2+2\varepsilon$ and $W'_B = 1+2\varepsilon$. We showed in Appendix A that this is indeed a hard instance for (standard) lookahead algorithms when the weight of the matching computed is used as the smoothness constraint (recall that $(1-\beta)\cdot w\big(\hat{M}(AB)\big) \leq w\big(\hat{M}(B)\big)$ is then required in a hard instance, which is the case here). On the other hand, refined lookahead algorithms allow us to use the sum of reduced weights as the smoothness constraint. Since $(1-\beta)\cdot W'_{AB} \not\leq W'_B$, for small enough β, the instance therefore is not hard for refined lookahead algorithms.

Approximate Sampling and Counting of Graphs with Near-Regular Degree Intervals

Georgios Amanatidis
University of Essex, Colchester, UK

Pieter Kleer
Tilburg University, The Netherlands

Abstract

The approximate uniform sampling of graphs with a given degree sequence is a well-known, extensively studied problem in theoretical computer science and has significant applications, e.g., in the analysis of social networks. In this work we study a generalization of the problem, where *degree intervals* are specified instead of a single degree sequence. We are interested in sampling and counting graphs whose degree sequences satisfy the corresponding degree interval constraints. A natural scenario where this problem arises is in hypothesis testing on networks that are only partially observed. We provide the first *fully polynomial almost uniform sampler (FPAUS)* as well as the first *fully polynomial randomized approximation scheme (FPRAS)* for sampling and counting, respectively, graphs with near-regular degree intervals, i.e., graphs in which every node has a degree from an interval not too far away from a given $r \in \mathbb{N}$. In order to design our FPAUS, we rely on various state-of-the-art tools from Markov chain theory and combinatorics. In particular, by carefully using Markov chain decomposition and comparison arguments, we reduce part of our problem to the recent breakthrough of Anari, Liu, Oveis Gharan, and Vinzant (2019) on sampling a base of a matroid under a strongly log-concave probability distribution, and we provide the first non-trivial algorithmic application of a breakthrough asymptotic enumeration formula of Liebenau and Wormald (2017). As a more direct approach, we also study a natural Markov chain recently introduced by Rechner, Strowick and Müller-Hannemann (2018), based on three local operations – switches, hinge flips, and additions/deletions of an edge. We obtain the first theoretical results for this Markov chain, showing it is rapidly mixing for the case of near-regular degree intervals of size at most one.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains

Keywords and phrases graph sampling, degree interval, degree sequence, Markov Chain Monte Carlo method, switch Markov chain

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.7

Related Version *Full Version*: https://arxiv.org/abs/2110.09068

Acknowledgements Part of this work has been carried out while Pieter Kleer was a postdoctoral fellow at the Max Planck Institute for Informatics in Saarbrücken, Germany.

1 Introduction

The (approximate) uniform sampling and counting of graphs with given degrees has received a lot of attention during the last few decades, see, e.g., [1, 5, 6, 8, 10, 14–18, 20, 22, 24–29, 34, 36, 37, 39, 42–44, 50]. Given a degree sequence $\boldsymbol{d} = (d_1, \ldots, d_n)$, the goal of approximate uniform sampling is to design a randomized algorithm that outputs a labelled simple undirected graph G with degree sequence $\boldsymbol{d}$, according to a distribution that is close to the uniform distribution over the set of all graphs with this degree sequence. Such an algorithm is called an *approximate (uniform) sampler*. Approximate samplers find applications in fields such as complex network analysis, where they serve as null models for hypothesis testing. Consider, e.g., a social network with edges representing friendships or relationships. One might see a very high number of edges between a certain group of nodes and, based on this,


© Georgios Amanatidis and Pieter Kleer;
licensed under Creative Commons License CC-BY 4.0


conjecture that these nodes form a *community* of friends or colleagues. In order to test this hypothesis, one would like to be able to generate graphs with *similar characteristics* as the observed network and, based on these generated samples, decide how likely it is that there is a high number of edges between that particular group of nodes by chance alone. Here the characteristic of interest is the degree sequence of the observed network [47]. For determining how many samples are sufficient in order to test the hypothesis, we also need to be able to count the number of graphs with the given degree sequence.

In practice, it is not always possible to have exact knowledge of the degree sequence of an observed network, due to erroneous measurements. In order to overcome this, there is a need for more robust null models. One such model was proposed by Rechner, Strowick and Müller-Hannemann [48]. Instead of a given degree sequence $\boldsymbol{d}$, the null models now consist of all graphs with given *degree interval* constraints $[\ell_i, u_i]$, for $i \in [n] = \{1, \ldots, n\}$. In this case we say that a graph G has degrees in the interval $[\boldsymbol{\ell}, \boldsymbol{u}]$ with $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_n)$ and $\boldsymbol{u} = (u_1, \ldots, u_n)$. The algorithmic task at hand then becomes to develop algorithms for sampling and counting graphs from the set $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ of all graphs satisfying the interval constraints. An intuitive two-step approach for solving this problem is to first sample *according to the correct proportional distribution* a degree sequence $\boldsymbol{d} = (d_1, \ldots, d_n)$ from the set of all degree sequences satisfying the interval constraints $\ell_i \leq d_i \leq u_i$, for $i \in [n]$, and then sample uniformly at random a graphical realization from the set $\mathcal{G}(\boldsymbol{d})$ of all graphs with degree sequence $\boldsymbol{d}$. A crucial difficulty that arises here is that the probability with which each degree sequence $\boldsymbol{d}$ needs to be sampled in the first step is not obvious. This probability should be proportional to the number $|\mathcal{G}(\boldsymbol{d})|$ which is not known in general.

To make the problem more concrete, we give a brief example in the context of the social network application that we started out with. Suppose we have a partially observed network. For a given node i, we let ℓ_i be the number of observed edges adjacent to i, δ_i the number of missing observations and, thus, $u_i = n - 1 - (\ell_i + \delta_i)$ the number of observed non-edges (i.e., pairs $\{i, j\}$ for which we know there is no edge between nodes i and j). There are now two extreme cases: either all missing observations are non-edges, meaning that node i has degree ℓ_i, or all missing observations are indeed edges, meaning that node i has degree u_i. Hence, we are interested in sampling (and counting) graphs for which each node i has a degree in the interval $[\ell_i, u_i]$, for every $i \in [n]$. In this and other similar settings, these problems seem to be natural and elegant generalizations of the classic graph sampling and counting problems.

Towards sampling graphs with given degree intervals, Rechner et al. [48] introduced a Markov chain based on three simple operations: *switches*, *hinge flips* and *additions/deletions*. The chain in each step selects one of these operations uniformly at random and performs it, if possible. We call this chain the *degree interval Markov chain*. The operations are shown in Figure 1 and a formal definition is given in Section 2. These three operations are the ones described by Coolen et al. [15] as the most commonly used operations in Markov Chain Monte Carlo algorithms for the generation of simple undirected graphs *in practice*. This serves as additional motivation for rigorously studying Markov chains based on these operations. We will also be interested in the *switch-hinge flip Markov chain* that only uses the switch and hinge flip operations. The hinge flip and switch operations are of particular interest both in theory and in practice as they preserve the number of edges and the degree sequence of a graph, respectively.

Our contributions. In this work, we give the first efficient approximate sampler and approximate counter for graphs with so-called *near-regular* degree intervals. Near-regularity here refers to the fact that all graphs have degrees which are close to a common value up to a

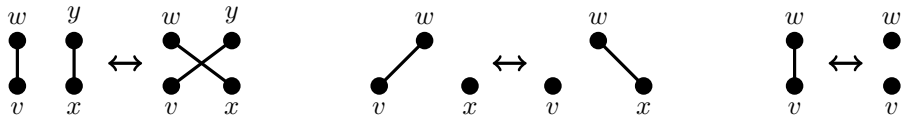


Figure 1 Left to right: switch on v, w, x, y; hinge flip on v, w, x; edge addition/deletion on v, w.

sublinear margin. To be more precise, we show that there is a *fully polynomial almost uniform sampler (FPAUS)* and a *fully polynomial randomized approximation scheme (FPRAS)* (for formal definitions see Section 2), in case the degree intervals are close to a common value $r = r(n) \in \mathbb{N}$, i.e., if $[\ell_i, u_i] \subseteq [r - r^\alpha, r + r^\alpha]$ for some $0 < \alpha < \frac{1}{2}$. The parameter $\alpha > 0$ models the maximum length of the degree intervals that we allow; this length of $2r^\alpha$ should be relatively small compared to r.[1] We also need a minor technical assumption on the value of r in order to avoid some (arguably not very interesting) boundary cases. The main result of this work is Theorem 1 below.

For vectors $\boldsymbol{a} = (a_1, \ldots, a_n), \boldsymbol{b} = (b_1, \ldots, b_n) \in \mathbb{R}^n$, we write $\boldsymbol{a} \leq \boldsymbol{b}$ if $a_i \leq b_i$ for all $i \in [n]$. Given $\boldsymbol{\ell}, \boldsymbol{u} \in \mathbb{N}^n$, by $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ we denote the set of all graphs G whose degree sequence $\boldsymbol{d}(G)$ satisfies $\boldsymbol{\ell} \leq \boldsymbol{d}(G) \leq \boldsymbol{u}$.

▶ **Theorem 1.** *Let $0 < \alpha < 1/2$ and $0 < \sigma < 1$ be fixed. Let $r = r(n)$ with $2 \leq r \leq (1 - \sigma)n$. If for every node $i \in [n]$ it holds that $[\ell_i, u_i] \subseteq [r - r^\alpha, r + r^\alpha]$, then there is an FPAUS for the approximate uniform sampling of graphs from $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ and an FPRAS for approximating $|\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})|$.*

For given degree intervals $[\boldsymbol{\ell}, \boldsymbol{u}]$ and $m \in \mathbb{N}$, we write $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ for the set of graphs G whose degree sequence $\boldsymbol{d}(G)$ satisfies $\boldsymbol{\ell} \leq \boldsymbol{d}(G) \leq \boldsymbol{u}$ *and* $\sum_i d_i = 2m$. By using reductions between approximate sampling and approximate counting [2, Appendix C] we get that to prove Theorem 1 it suffices to show the existence of an FPAUS for sampling from $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$. To this end, we show that the switch-hinge flip Markov chain is rapidly mixing under the conditions of Theorem 1. This result is summarized in Theorem 2.

▶ **Theorem 2.** *Let α, σ, and r be as in Theorem 1. If $[\ell_i, u_i] \subseteq [r - r^\alpha, r + r^\alpha]$, for all $i \in [n]$, and $2m \in \left[\sum_i \ell_i, \sum_i u_i\right]$, then the switch-hinge flip Markov chain is rapidly mixing on $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$.*

A more direct approach for sampling from $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ than the one behind Theorem 1 would be to use the degree interval Markov chain. An interesting open question is whether this chain is rapidly mixing under the assumptions in Theorem 1 (or under weaker assumptions). As a first step into this direction, we show rapid mixing when all the degree intervals have size at most one, i.e., when $u_i - 1 \leq \ell_i \leq u_i$.

▶ **Theorem 3.** *Let α, σ, and r be as in Theorem 1. If $[\ell_i, u_i] \subseteq [r - r^\alpha, r + r^\alpha]$ and $u_i - 1 \leq \ell_i \leq u_i$, for all $i \in [n]$, then the degree interval Markov chain is rapidly mixing on $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$.*

The technical novelty of our work lies in the highly nontrivial combination of state-of-the-art tools from Markov chain theory and combinatorics. An overview of our proof approach is given in Section 3. It relies on Markov chain decomposition and comparison techniques of

[1] One should note that an assumption of this kind is to be expected. Otherwise, we would be also solving the problem of (approximately) uniformly sampling a graph with *any* given degree sequence, which is a long-standing open problem.

Martin and Randall [41], rapid mixing results for the switch Markov chain by Amanatidis and Kleer [1], the breakthrough work of Anari et al. [4] on strongly log-concave probability distributions, and the work of Liebenau and Wormald [39] regarding asymptotic enumeration formulas for the number of near-regular graphs.

▶ Remark 4. Our theorems – and all the building blocks used in their proofs – are shown to be true for all $n \geq n_0$, where $n_0 \in \mathbb{N}$ is a constant that depends on the other constant parameters involved. It is straightforward that for $n < n_0$ our results are always true.

Related work. For the problem of sampling graphs with a given degree sequence, there is an extensive literature, particularly on Markov Chain Monte Carlo (MCMC) methods. Jerrum and Sinclair [34] provide an approximate uniform sampler and an approximate counter for *P-stable* degree sequences, for which the number of graphical realizations of a given degree sequence does not vary too much under small perturbations of the sequence. A first step beyond P-stability was recently made by Erdős et al. [21]. Jerrum, Sinclair and Vigoda [35] provide an approximate sampler (and counter) for arbitrary bipartite degree sequences by reducing the problem to sampling perfect matchings in an appropriate graph representation of the given instance. The work of Bézakova, Bhatnagar and Vigoda [7] provides a more direct approach. There are also various non-MCMC methods available in the literature, see, e.g., [5, 27, 28, 37, 43, 50]. One MCMC approach that has received considerable attention is the *switch Markov chain*, based on the switch operation in Figure 1. This is a simpler, more direct approach than reducing the problem to sampling a perfect matching from a large auxiliary graph. The chain was first analyzed by Kannan, Tetali and Vempala [36], and has been extensively studied, see, e.g., [1, 16, 22, 44]. The state of the art on its mixing time is the work of Erdős et al. [22], who show that the chain is rapidly mixing for all P-stable degree sequences. Very recently, Erdős, Mezei and Miklós [23] generalized our Theorem 3 to intervals of length 1 centered around P-stable degree sequences. We consider the fact that their meticulous direct approach does not go beyond length 1 as another indication of the difficulty of directly arguing about the degree interval Markov chain.

The decomposition theorem of Martin and Randall [41] we use (Theorem 8), based on the decomposition method of Madras and Randall [40], also appeared in an unpublished manuscript by Caracciolo, Pelissetto and Sokal [13]. Erdős et al. [25] use a related decomposition approach for sampling *balanced joint degree matrix* realizations. Rechner et al. [48] introduce the degree interval Markov chain for the *bipartite* version of the problem of sampling graphs with given degree intervals and show its irreducibility for arbitrary degree intervals. The result of Liebenau and Wormald [39] builds on a long line of work on asymptotic expressions for the number of graphs with given degrees. Indicatively, Bender and Canfield [6] gave a formula for bounded degree sequences and Bollobás [10] for r-regular sequences with $r = O(\sqrt{\log(n)})$. McKay and Wormald gave expressions both for sparse sequences with maximum degree $o(n^{1/2})$ [43] and for a certain dense regime [42].

Anari et al. [4], in a breakthrough recent work, gave the first polynomial time algorithm for approximate sampling a base of a matroid under a strongly log-concave probability distribution. The theory of strongly log-concave (or Lorentzian) polynomials dates back to the work of Gurvits [31], and was further developed by Anari, Oveis Gharan and Vinzant [3] and Brändén and Huh [12]. In another recent work, Kleer [38] made a connection between asymptotic enumeration formulas and strongly log-concave polynomials for a case of sparse bipartite graphs where only the degrees on one side of the bipartition can vary.

Outline. In Section 2 we provide all the necessary preliminaries. We then continue with a proof overview for Theorems 2 and 3 in Section 3. For readers with some familiarity regarding Markov chains and some intuition about degree sequence problems, it should be possible to go through (most of) Section 3 without delving into (the admittedly long) Section 2 first. As one of the main building blocks of the proof of Theorem 2, we show in Section 4 that the asymptotic formula of Liebenau and Wormald [39], when restricted to the degree interval regime of Theorem 1, approximately gives rise to a so-called strongly log-concave polynomial, a result which might be of independent interest. Appendix A contains most of the remaining arguments about the Markov chains used in our proofs. Any missing proofs are deferred to the full version of this paper [2].

2 Preliminaries

We need a variety of preliminaries for this work, that are all collected in this section.

2.1 M-convexity and strongly log-concave polynomials

We start with the notion of M-convexity for functions [45, 46]. Let $\nu : \mathbb{Z}^n_{\geq 0} \to \mathbb{R} \cup \{\infty\}$ be a function. The *effective domain* of ν is given by $\mathrm{dom}(\nu) = \{\boldsymbol{\alpha} \in \mathbb{N}^n : \nu(\boldsymbol{\alpha}) < \infty\}$. The function ν is called $M^\natural$*-convex* if it satisfies the *(symmetric) exchange property*: For any $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathrm{dom}(\nu)$ and any $i \in [n]$ satisfying $\alpha_i > \beta_i$, there exists a $j \in [n]$ such that $\alpha_j < \beta_j$ and $\nu(\boldsymbol{\alpha}) + \nu(\boldsymbol{\beta}) \geq \nu(\boldsymbol{\alpha} - \boldsymbol{e}_i + \boldsymbol{e}_j) + \nu(\boldsymbol{\beta} + \boldsymbol{e}_i - \boldsymbol{e}_j)$,

where $\boldsymbol{e}_k$ is defined as $\boldsymbol{e}_k(\ell) = 1$ if $k = \ell$ and $\boldsymbol{e}_k(\ell) = 0$ otherwise. The function ν is called M*-convex* if it is $M^\natural$-convex *and* there is an $d \in \mathbb{N}$ such that $\mathrm{dom}(\nu) \subseteq \{\boldsymbol{\alpha} : \sum_i \alpha_i = d\}$. A subset $C \subseteq \mathbb{Z}^n_{\geq 0}$ is called M-convex if the indicator function $\nu_C : \mathbb{Z}^n_{\geq 0} \to \mathbb{R} \cup \{\infty\}$, given by $\nu_C(\boldsymbol{\alpha}) = 1$ if $\boldsymbol{\alpha} \in C$ and $\nu_C(\boldsymbol{\alpha}) = 0$ otherwise, is M-convex.

We write $\mathbb{R}[x_1, \ldots, x_n]$ to denote the set of all polynomials in $x_1, \ldots, x_n$ with real coefficients. We consider polynomials $p \in \mathbb{R}[x_1, \ldots, x_n]$ with non-negative coefficients. For a vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n) \in \mathbb{Z}^n_{\geq 0}$, we write $\partial^{\boldsymbol{\beta}} = \prod_{i=1}^n \partial_{x_i}^{\beta_i}$ to denote the partial differential operator that differentiates a function β_i times with respect to x_i for $i = 1, \ldots, n$. For $\boldsymbol{\alpha} \in \mathbb{Z}^n_{\geq 0}$, we write $x^{\boldsymbol{\alpha}}$ to denote $\prod_{i=1}^n x_i^{\alpha_i}$. Furthermore, we write $\boldsymbol{\alpha}! = \prod_i \alpha_i!$, and for $\boldsymbol{\alpha}, \boldsymbol{\kappa} \in \mathbb{Z}^n_{\geq 0}$ with $\alpha_i \leq \kappa_i$ for all i, we write $\binom{\boldsymbol{\kappa}}{\boldsymbol{\alpha}} = \prod_{i=1}^n \binom{\kappa_i}{\alpha_i}$. For a constant $c \in \mathbb{N}$ with $c \geq \max_i \alpha_i$, we write $\binom{c}{\boldsymbol{\alpha}} = \prod_{i=1}^n \binom{c}{\alpha_i}$. Let $\boldsymbol{\kappa} \in \mathbb{Z}^n_{\geq 0}$ and the Cartesian product $K = \times_i \{0, \ldots, \kappa_i\}$. Let $w : K \to \mathbb{R}_{\geq 0}$ be a weight function. The *generating polynomial* of w is $g_{\boldsymbol{\kappa}}(x) = \sum_{\boldsymbol{\alpha} \in K} w(\boldsymbol{\alpha}) x^{\boldsymbol{\alpha}}$. The *support* of $g_{\boldsymbol{\kappa}}$ is the set $\mathrm{supp}(g_{\boldsymbol{\kappa}}) = \{\boldsymbol{\alpha} \in K : w(\boldsymbol{\alpha}) > 0\}$. The polynomial $g_{\boldsymbol{\kappa}}$ is called d-homogeneous if $|\boldsymbol{\alpha}| = \sum_i \alpha_i = d$ for all $\boldsymbol{\alpha} \in \mathrm{supp}(g_{\boldsymbol{\kappa}})$.

▶ **Definition 5** (Strong log-concavity [31]). *A polynomial $p \in \mathbb{R}[x_1, \ldots, x_n]$ with non-negative coefficients is called* log-concave *on a subset $S \subseteq \mathbb{R}^n_{\geq 0}$ if its Hessian $\nabla^2 \log(p)$ is negative semidefinite on S. A polynomial p is called* strongly log-concave (SLC) *on S if for any $\boldsymbol{\beta} \in \mathbb{N}^n$, we have that $\partial^{\boldsymbol{\beta}} p$ is log-concave.*

For convenience, the zero polynomial is defined to be SLC always. Finally, if the generating polynomial $g_{\boldsymbol{\kappa}}$ is SLC, then the probability distribution $\pi(\boldsymbol{\alpha}) \propto w(\boldsymbol{\alpha})$ is called SLC as well. We next state some properties of SLC polynomials that will be used in this work.

▶ **Proposition 6** ([12]). *If $p \in \mathbb{R}[x_1, \ldots, x_n]$ is SLC and $\gamma \in \mathbb{R}_{\geq 0}$, then γp is SLC.*

▶ **Proposition 7** (Following from [12]).[2] *Let $\nu : \mathbb{Z}^n_{\geq 0} \to \mathbb{R} \cup \{\infty\}$ with $dom(\nu) \subseteq \{0, 1, \ldots, n-1\}^n$ and let*

$$f_{\boldsymbol{\kappa}}(x) = \sum_{\boldsymbol{\alpha} \in dom(\nu)} \frac{1}{\boldsymbol{\alpha}!} e^{-\nu(\boldsymbol{\alpha})} x^{\boldsymbol{\alpha}} \quad \text{and} \quad g_{\boldsymbol{\kappa}}(x) = \sum_{\boldsymbol{\alpha} \in dom(\nu)} \binom{\gamma}{\boldsymbol{\alpha}} e^{-\nu(\boldsymbol{\alpha})} x^{\boldsymbol{\alpha}}, \tag{1}$$

be $2m$-homogeneous polynomials, where $\gamma = (n-1, \ldots, n-1)$. If ν is M-convex, then $f_{\boldsymbol{\kappa}}$ and $g_{\boldsymbol{\kappa}}$ are SLC.

2.2 Markov chains and mixing times

Let $\mathcal{M} = (\Omega, P)$ be an ergodic, time-reversible Markov chain with state space Ω, transition matrix P, and stationary distribution π. We write $P^t(x, \cdot)$ for the distribution over Ω at time step t with initial state $x \in \Omega$. The *total variation distance* of this distribution from stationarity at time t with initial state x is

$$\Delta_x(t) = \frac{1}{2} \sum_{y \in \Omega} \left| P^t(x, y) - \pi(y) \right|,$$

and the *mixing time* of $\mathcal{M}$ is

$$\tau(\epsilon) = \max_{x \in \Omega} \tau_x(\epsilon) \text{ where } \tau_x(\epsilon) = \min\{t : \Delta_x(t') \leq \epsilon \text{ for all } t' \geq t\} \text{ for } \epsilon > 0\,.$$

The chain $\mathcal{M}$ is said to be *rapidly mixing* if its mixing time is upper bounded by a polynomial in $\ln(|\Omega|/\epsilon)$.

It is well-known that the matrix P only has real eigenvalues $1 = \lambda_0 > \lambda_1 \geq \cdots \geq \lambda_{|\Omega|-1} > -1$. We may replace P by $(P + I)/2$ to make the chain *lazy*, and hence guarantee that all its eigenvalues are non-negative. In that case, by $\mathrm{Gap}(P) = 1 - \lambda_1$ we denote the spectral gap of P. In this work all Markov chains involved are lazy. It is well known that one can use the spectral gap to give an upper bound on the mixing time of Markov chain. That is, it holds that

$$\tau_x(\epsilon) \leq \frac{1}{2(1 - \lambda_1(P))} \Big(\log \pi(x)^{-1} + 2 \log \Big(\frac{1}{2\epsilon} \Big) \Big),$$

as it follows directly from Proposition 1 in [49]. Furthermore, if one has two Markov chains $\mathcal{M} = (\Omega, P)$ and $\mathcal{M}' = (\Omega, P')$ both with stationary distribution π and there are constants c_1, c_2 such that $c_1 P(x, y) \leq P'(x, y) \leq c_2 P(x, y)$ for all $x, y \in \Omega$ with $x \neq y$. Then (see, e.g., [41]) it follows that $c_1 \,\mathrm{Gap}(P) \leq \mathrm{Gap}(P') \leq c_2 \,\mathrm{Gap}(P)$.

The *state space graph* of the chain $\mathcal{M}$ is the directed graph $\mathbb{G} = \mathbb{G}(\mathcal{M})$ with node set Ω that contains the edges $(x, y) \in \Omega \times \Omega$ for which $P(x, y) > 0$ and $x \neq y$. that contains an edge $(x, y) \in \Omega \times \Omega$ if and only if $P(x, y) > 0$ and $x \neq y$ (denoted by $x \sim y$). Let $\mathcal{P} = \bigcup_{x \neq y} \mathcal{P}_{xy}$, where $\mathcal{P}_{xy}$ is the set of simple paths between x and y in the state space graph $\mathbb{G}$. A *flow* f in Ω is a function $\mathcal{P} \to [0, \infty)$ satisfying $\sum_{p \in \mathcal{P}_{xy}} f(p) = \pi(x)\pi(y)$ for all $x, y \in \Omega$, $x \neq y$. The flow f can be extended to the oriented edges $e = (z, z')$ of $\mathbb{G}$ by setting $f(e) = \sum_{p \in \mathcal{P} : e \in p} f(p)$, so that $f(e)$ is the total flow routed through $e \in E(\mathbb{G})$. Let $\mathrm{length}(f) = \max_{p \in \mathcal{P} : f(p) > 0} |p|$

[2] Our $g_{\boldsymbol{\kappa}}$ is a slight variant the corresponding function of Theorem 3.14 of [12] with $q = 1/e$. The statement for this $g_{\boldsymbol{\kappa}}$ follows from a simple transformation of $f_{\boldsymbol{\kappa}}$ that preserves strong log-concavity, namely the operator that maps $x^{\boldsymbol{\alpha}}$ to $\boldsymbol{\alpha}! \binom{\gamma}{\boldsymbol{\alpha}} x^{\boldsymbol{\alpha}}$ [11].

be the length of a longest flow-carrying path, and let $\mathrm{load}(e) = f(e)/Q(e)$ be the *load* of the edge e, where $Q(e) = \pi(x)P(x,y)$ for $e = (x,y)$. If $\mathrm{load}(f) = \max_{e \in E(\mathbb{G})} \psi(e)$ is the maximum load of the flow, it holds that $\mathrm{Gap}(P)^{-1} \leq \mathrm{load}(f)\,\mathrm{length}(f)$ (see, e.g, [49]).

We will sometimes also work (implicitly) with the so-called *modified log-Sobolev constant* $\rho = \rho(P)$. This constant can also be used to upper bound the mixing time of a Markov chain. In particular, it holds that

$$\tau_x(\epsilon) \leq \frac{1}{\rho(P)} \left(\log\log \pi(x)^{-1} + \log\left(\frac{1}{2\epsilon^2}\right)\right),$$

see, e.g., [9]. Details on $\rho(P)$ are given in [2, Appendix B].

Markov chain decomposition. We describe a Markov chain decomposition of Martin and Randall [41] that follows the decomposition framework of Madras and Randall [40]. Let $\mathcal{M} = (\Omega, P)$ be a Markov chain and $\bigcup_{i=1}^{q} \Omega_i$ be a partition of Ω for some $q \in \mathbb{N}$. We define the restriction Markov chains $\mathcal{M}_i = (\Omega_i, P_{\Omega_i})$ as follows. For $x \in \Omega_i$ we let $P_{\Omega_i}(x,y) = P(x,y)$ if $x, y \in \Omega_i$ with $x \neq y$, and $P_{\Omega_i}(x,x) = 1 - \sum_{y \in \Omega_i, y \neq x} P_{\Omega_i}(x,y)$. Furthermore, let $\partial_i(\Omega_j) = \{y \in \Omega_j : \exists x \in \Omega_i \text{ with } P(x,y) > 0\}$ be the set of elements in Ω_j that can be reached with positive probability in one transition of the chain $\mathcal{M}$ from some element in Ω_i.

Let $\mathcal{M}_{\mathrm{MH}} = ([q], P_{\mathrm{MH}})$ be (the Metropolis-Hastings variation of) the projection Markov chain on $[q] = \{1, \ldots, q\}$. That is, $P_{\mathrm{MH}}(i,j) > 0$ if and only if $\partial_i(\Omega_j) \neq \emptyset$ and, in that case,

$$P_{\mathrm{MH}}(i,j) = \frac{1}{2\Delta} \min\left\{1, \frac{\pi(\Omega_j)}{\pi(\Omega_i)}\right\}, \tag{2}$$

for $i \neq j$, where Δ is the maximum out-degree in the state space graph of $\mathcal{M}_{\mathrm{MH}}$, while $P_{\mathrm{MH}}(i,i) = 1 - \sum_{j \in [q]\setminus\{i\}} P_{\mathrm{MH}}(i,j)$. Note that $\mathcal{M}_{\mathrm{MH}}$ has stationary distribution $\pi_{\mathrm{MH}}(i) = \pi(\Omega_i)$ for $i \in \{1, \ldots, q\}$ and a holding probability of at least $1/2$. We will use the following decomposition theorem from [41].

▶ **Theorem 8** ([41], Corollary 3.3). *Suppose there exist $\beta > 0$ and $\gamma > 0$ such that $P(x,y) \geq \beta$ for all x, y that are adjacent in $\mathbb{G}(\mathcal{M})$, and $\pi(\partial_i(\Omega_j)) \geq \gamma\pi(\Omega_j)$ for all i, j that are adjacent in $\mathbb{G}(\mathcal{M}_{\mathrm{MH}})$. Then* $\mathrm{Gap}(P) \geq \beta\gamma \cdot \mathrm{Gap}(P_{\mathrm{MH}}) \cdot \min_{i=1,\ldots,q} \mathrm{Gap}(P_{\Omega_i})$.

Load-exchange Markov chain. In this work, we will need a weighted version of the *base-exchange Markov chain* studied by Anari et al. [4]. Let π be a strongly log-concave probability distribution with $\pi(\boldsymbol{\alpha}) \propto w(\boldsymbol{\alpha})$ whose support forms an M-convex set C. We define the *(unit) load-exchange Markov chain* on $C \subseteq Z^n_{\geq 0}$:

Assuming $\boldsymbol{\alpha} \in C$ is the current state of the *(unit) load-exchange Markov chain*:
- Select an element $i \in [n]$ uniformly at random.
- Pick an $\boldsymbol{\alpha}' \in C$ with $\boldsymbol{\alpha}' \geq \boldsymbol{\alpha} - \boldsymbol{e_i}$ with probability $\propto w(\boldsymbol{\alpha}')$ among all such $\boldsymbol{\alpha}'$.

Similarly to the base-exchange Markov chain [4], the above procedure defines an ergodic, time-reversible Markov chain with stationary distribution π over C given by $\pi(\boldsymbol{\alpha}) \propto w(\boldsymbol{\alpha})$. Using the notion of *polarization* for SLC polynomials [12], in combination with a simple Markov chain comparison argument (as in [2, Appendix B]), Corollary 9 can be shown. The proof (which is implicitly given in [38]), roughly speaking, uses a reduction to the case of matroids, after which a result of Cryan et al. [19] gives the desired result.

▶ **Corollary 9.** *Let $\boldsymbol{\kappa} = (n, \ldots, n)$ and suppose that the d-homogeneous polynomial $g_{\boldsymbol{\kappa}}(x) = \sum_{\boldsymbol{\alpha} \in K} w(\boldsymbol{\alpha}) x^{\boldsymbol{\alpha}} \in \mathbb{R}[x_1, \ldots, x_n]$ is SLC. Then the transition matrix P of the load-exchange Markov chain on $\operatorname{supp}(g_{\boldsymbol{\kappa}})$ satisfies $\rho(P) \geq 1/(n^2 d)$, where $\rho(P)$ is the modified log-Sobolev constant of P.*

Degree intervals and the switch-hinge flip Markov chain. A sequence of non-negative integers $\boldsymbol{d} = (d_1, \ldots, d_n)$ is called a *graphical degree sequence* if there exists a simple, undirected, labeled graph $G = (V, E)$ on nodes $V = [n]$, where node i has degree d_i, for $i \in V$. Such a graph is called a *(graphical) realization of* $\boldsymbol{d}$. By $\mathcal{G}(\boldsymbol{d})$ we denote the set of all graphical realizations of $\boldsymbol{d}$, while by $\boldsymbol{d}(G)$ we denote the degree sequence of a graph G. For given vectors $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_n)$ and $\boldsymbol{u} = (u_1, \ldots, u_n)$ of non-negative integers, we define $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u}) = \bigcup_{\boldsymbol{\ell} \leq \boldsymbol{d} \leq \boldsymbol{u}} \mathcal{G}(\boldsymbol{d})$ as the set of all graphical realizations G satisfying $\boldsymbol{\ell} \leq \boldsymbol{d}(G) \leq \boldsymbol{u}$, meaning $\ell_i \leq d_i(G) \leq u_i$ for all $i \in V$. For $m \in \mathbb{N}$, we define $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ as the set of all graphical realizations $G \in \mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ with precisely m edges, i.e., with $\sum_i d_i(G) = 2m$. Finally, we define the set of all degree sequences satisfying the degree interval constraints, and whose total sum of the degrees equals $2m$, as $\mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u}) = \{\boldsymbol{d} : \boldsymbol{\ell} \leq \boldsymbol{d} \leq u \text{ and } \sum_i d_i = 2m\}$.

A *fully polynomial almost uniform sampler (FPAUS)* for sampling graphs with given degree intervals $[\boldsymbol{\ell}, \boldsymbol{u}]$ is an algorithm that, for any $\epsilon > 0$, outputs a graph $G \in \mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ according to a distribution $\tilde{\pi}$ such that $d_{\mathrm{TV}}(\pi, \tilde{\pi}) \leq \epsilon$, where π is the uniform distribution over $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$, and runs in time polynomial in $n, \log(1/\delta)$ and $\log(1/\epsilon)$. A *fully polynomial randomized approximation scheme (FPRAS)* for the problem is an algorithm that, for every $\epsilon,\ \delta > 0$, outputs $|\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})|$ up to a multiplicative factor $(1+\epsilon)$ with probability at least $1-\delta$, in time polynomial in n, $1/\epsilon$ and $\log(1/\delta)$. Analogous definitions hold for $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ for a given m.

First we define the *switch-hinge flip Markov chain* to uniformly sample elements from $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ based on two of the local operations of Figure 1.

Assuming $G \in \mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ is the current state of the *switch-hinge flip Markov chain*:
- With probability 2/3, do nothing.
- With probability 1/6, try to perform a *switch operation*: Choose an ordered tuple of distinct nodes (v, w, x, y) uniformly at random. If $\{w, v\}, \{x, y\} \in E(G)$, and $\{y, v\}, \{x, w\} \notin E(G)$, then delete $\{w, v\}, \{x, y\}$ from $E(G)$, and add $\{y, v\}, \{x, w\}$ to $E(G)$.
- With probability 1/6, try to perform a *hinge flip operation*: Choose an ordered tuple of distinct nodes (v, w, x) uniformly at random. If $\{w, v\} \in E(G)$ and $\{w, x\} \notin E(G)$, then delete $\{w, v\}$ from and add $\{w, x\}$ to $E(G)$ if the resulting graph is in $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$.

Similarly, we define the *degree interval Markov chain* of Theorem 3, that can also perform addition/deletion operations.

Assuming $G \in \mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ is the current state of the *degree interval Markov chain*:
- With probability 1/2, do nothing.
- Otherwise:
  - With probability 1/6, try to perform a *switch operation*.
  - With probability 1/6, try to perform a *hinge flip operation*.
  - With probability 1/6, try to perform an *addition/deletion operation*: select an ordered tuple of distinct nodes (v, w) uniformly at random. If $\{v, w\} \in E(G)$, then delete it from $E(G)$ if the resulting graph is in $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$. Similarly, if $\{v, w\} \notin E(G)$, then add it to $E(G)$ if the resulting graph is in $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$.

Due to the symmetry of the transition probabilities, it is not hard to see that both chains are time-reversible with respect to the uniform distribution. Also because of the holding probability of at least $1/2$, the chains are aperiodic. Finally, by a simple counting argument, there exists polynomials $t(n), t'(n)$ such that $P_{\mathcal{G}(\boldsymbol{\ell},\boldsymbol{u})}(G,H) \geq 1/t(n)$ for all $G, H \in \mathcal{G}(\boldsymbol{\ell},\boldsymbol{u})$ with $P_{\mathcal{G}(\boldsymbol{\ell},\boldsymbol{u})}(G,H) > 0$ and $P_{\mathcal{G}_m(\boldsymbol{\ell},\boldsymbol{u})}(G,H) \geq 1/t'(n)$ for all $G, H \in \mathcal{G}_m(\boldsymbol{\ell},\boldsymbol{u})$ with $P_{\mathcal{G}_m(\boldsymbol{\ell},\boldsymbol{u})}(G,H) > 0$ respectively. The irreducibility of the chain (i.e., the fact that its state space is strongly connected) for the intervals of interest will follow implicitly from our analysis, in particular Lemmata 18 and 20.

2.3 Near-regular degree sequences

Let $r \geq 1$ be a given integer. A degree sequence $\boldsymbol{d}$ is said to be r-regular if $d_i = r$ for $i \in [n]$. For a fixed $0 \leq \alpha < 1$ we say that a degree sequence $\boldsymbol{d}$ is (α, r)-*near-regular* if $\max_i |d_i - r| \leq r^\alpha$. When we do not refer to a specific (α, r) pair, we just write about *near-regular* degree sequences.

We state some properties of near-regular degree sequences that we will use later. The most important result is Theorem 10 below. We use a slightly different formulation than that of [39].[3] For any degree sequence $\boldsymbol{d} = (d_1, \ldots, d_n)$, define

$$\xi = \sum_i d_i/n, \quad \mu = \xi/(n-1), \quad \text{and} \quad \chi = \sum_i (d_i - \xi)^2/(n-1)^2 .$$

Roughly speaking, the theorem states that if the distance between the degree sequence $\boldsymbol{d}$ and the ξ-regular sequence of the same size is not too large, then the expression in (3) is a good approximation for $|\mathcal{G}(\boldsymbol{d})|$. The absolute constant α in Theorem 1 is mostly restricted by the ϵ in Theorem 10.

▶ **Theorem 10** (Liebenau and Wormald [39]). *There exists an absolute constant $\epsilon > 0$ such that for every sequence of degree sequences $\left(\boldsymbol{d}^{(n)}\right)_{n\in\mathbb{N}}$ with ξn even, $\max_{i\in[n]} |d_i^{(n)} - \xi| = o\left(n^\epsilon \min\{\xi, n-\xi-1\}^{1/2}\right)$, and $n^2 \min\{\mu, 1-\mu\} \to \infty$, it holds that*

$$|\mathcal{G}(\boldsymbol{d})| \sim \bar{w}(\boldsymbol{d}) := \sqrt{2}\exp\left(\frac{1}{4} - \frac{\chi^2}{4\mu^2(1-\mu)^2}\right)\left(\mu^\mu(1-\mu)^{(1-\mu)}\right)^{n(n-1)/2}\prod_i \binom{n-1}{d_i^{(n)}}. \quad (3)$$

To be precise, there exists a non-negative function $\delta(n)$ with $\delta(n) \to 0$ as $n \to \infty$, so that the relative error in "$\sim$" is bounded above in absolute value by $\delta(n)$ for every such $\left(\boldsymbol{d}^{(n)}\right)_{n\in\mathbb{N}}$.

Furthermore, we also rely on the notion of strong stability introduced in [1] (and implicitly already used in [32]). A combinatorial definition of this notion is given below. It essentially states that any graph with a slightly perturbed degree sequence can easily be transformed into a graph with the desired degree sequence by flipping the edges on a short alternating path. An *alternating (u,v)-path in a graph G* is a (possibly non-simple) edge-disjoint (u,v)-path (in the corresponding complete graph) alternating between edges and non-edges of G, starting with an edge adjacent to u, and ending with a non-edge adjacent to v; recall that a non-edge is an edge contained in the complement of $E(G)$. If $u = v$ we obtain an *alternating cycle*. To facilitate the definition of strong stability, let $\mathcal{G}'(\boldsymbol{d}) = \bigcup_{\boldsymbol{d}'} \mathcal{G}(\boldsymbol{d}')$ with $\boldsymbol{d}'$ ranging over all sequences $\boldsymbol{d}'$ satisfying $\sum_i d'_i = \sum_i d_i$ and $\sum_i |d'_i - d_i| = 2$, i.e., there exist κ, λ such that $d'_\kappa = d_\kappa + 1$, $d'_\lambda = d_\lambda - 1$, and $d'_i = d_i$ otherwise.

[3] Our formulation is in line with the note after Conjecture 1.2 in [39].

▶ **Definition 11** (Strong stability). *A class $\mathcal{D}$ of degree sequences is strongly stable if there exists a constant $k \in \mathbb{N}$, such that for all $\boldsymbol{d} \in \mathcal{D}$ and all $G \in \mathcal{G}'(\boldsymbol{d})$, there is an alternating (u, v)-path in G of length at most k, where u and v are the unique nodes with $\deg_G(u) = d_u + 1$ and $\deg_G(v) = d_v - 1$.*

▶ **Proposition 12.** *Let $0 < \alpha < 1/2$ be a constant and assume that $2 \leq r(n) \leq (1 - \sigma)n$ for some constant $0 < \sigma < 1$ and $n \in \mathbb{N}$. Then there exists some $n_1 \in \mathbb{N}$ so that the class $\mathcal{F}_{(\alpha,r)}$ of (α, r)-near-regular degree sequences of length at least n_1 is strongly stable.*

The following two results hold for the class $\mathcal{F}_{(\alpha,r)}$ in Proposition 12. Lemma 13 essentially states that if an edge is present in some graphical realization, then there exists a short alternating cycle to obtain a graphical realization with the same degree sequence not containing that edge. As a result, the subset of realizations in $\mathcal{G}(\boldsymbol{d})$ containing a given edge and the set of realizations not containing it are polynomially related in size.

▶ **Lemma 13.** *Let $\boldsymbol{d} \in \mathcal{F}_{(\alpha,r)}$. Suppose that $G \in \mathcal{G}(\boldsymbol{d})$ and let $\{u, v\} \in E(G)$ (resp. $\{u, v\} \notin E(G)$). Then there exists a graph $G' \in \mathcal{G}(\boldsymbol{d})$ with $\{u, v\} \notin E(G')$ (resp. $\{u, v\} \in E(G)$) and $E(G) \triangle E(G')$ is an alternating cycle of length at most 12. Similarly, suppose that $\{u, w\}, \{u, v\} \in E(G)$. Then there exists a graph $G' \in \mathcal{G}(\boldsymbol{d})$ with $\{u, w\} \in E(G')$ and $\{u, v\} \notin E(G')$, and $E(G) \triangle E(G')$ is an alternating cycle of length at most 12.*

Furthermore, the *switch Markov chain* is rapidly mixing for the class $\mathcal{F}_{(\alpha,r)}$. This follows directly from [1] where it is shown that the switch Markov chain is rapidly mixing for all strongly stable classes of degree sequences. In particular, we will use the following result.

▶ **Corollary 14** (Follows from [1]). *Let $q(n) \geq 2$ be a given polynomial and consider the* lazy switch Markov chain *$\mathcal{M} = (\mathcal{G}(\boldsymbol{d}), P_{\mathcal{G}(\boldsymbol{d})})$ for some $\boldsymbol{d} \in \mathcal{F}_{(\alpha,r)}$ that proceeds as follows: For a given $G \in \mathcal{G}(\boldsymbol{d})$*

- *With probability $1 - 1/q(n)$ do nothing, and*
- *With probability $1/q(n)$, try to perform a switch operation.*

Then there exists a polynomial $p(n)$, such that for any $\boldsymbol{d} \in \mathcal{F}_{(\alpha,r)}$ we have $\mathrm{Gap}(P_{\mathcal{G}(\boldsymbol{d})}) \geq 1/p(n)$.

3 Proof approach overview

In this section we give a high-level overview of the proofs of Theorems 2 and 3. The idea is to decompose the degree interval Markov chain twice, using the graph operations in Figure 1. (The second step suffices to prove Theorem 2, and both steps are needed to prove Theorem 3.) We first decompose $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ based on the addition/deletion operation. Every part of the decomposition corresponds to a set $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ containing all graphs respecting the degree intervals $[\boldsymbol{\ell}, \boldsymbol{u}]$ and having exactly m edges, for some m. That is, there is a one-to-one correspondence between the possible values of m, and the parts of the decomposition. The Markov chain decomposition result of Theorem 8 tells us that if the switch hinge-flip chain is rapidly mixing for every m, and if it relatively "easy" to move between the different parts $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ by means of additions/deletions, then the degree interval chain is rapidly mixing on $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$. In the second step we carry out a similar decomposition, but now based on the hinge flip operation. That is, for a given m we decompose $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ in the sets $\mathcal{G}(\boldsymbol{d})$ for all sequences $\boldsymbol{d}$ which satisfy the interval constraints, and whose degrees sum up to $2m$. If the switch chain is rapidly mixing on every $\mathcal{G}(\boldsymbol{d})$, and we can move "easily" between the sets $\mathcal{G}(\boldsymbol{d})$ using hinge flip operations, then the switch hinge-flip Markov chain on $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ is also rapidly mixing. We continue with a formalization of these statements.

Let $\mathcal{T} = \{m_1, \dots, m_2\}$, where m_1 and m_2 are the minimum and maximum number of edges, respectively, that any $G \in \mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ could have; e.g., $m_1 = \frac{1}{2}\sum_i \ell_i$ and $m_2 = \frac{1}{2}\sum_i u_i$ in case the two summands are even.

First we partition $\mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u})$ into disjoint sets $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ for $m \in \mathcal{T}$. Recall that $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u}) = \left\{G \in \mathcal{G}(\boldsymbol{\ell}, \boldsymbol{u}) : \sum_i d_i(G) = 2m\right\}$. The restriction Markov chains $\mathcal{M}_{\mathcal{G}_m(\boldsymbol{\ell},\boldsymbol{u})}$ are essentially given by restricting the original chain to only perform switch and hinge flip operations that respect the degree intervals on graphs with precisely m edges. Applying Theorem 8 – with β and γ to be determined later – we get

$$\mathrm{Gap}(P) \geq \beta\gamma \cdot \mathrm{Gap}(P_{\mathcal{T}}) \cdot \min_{m \in \mathcal{T}} \mathrm{Gap}(P_{\mathcal{G}_m(\boldsymbol{\ell},\boldsymbol{u})}), \tag{4}$$

where $P_{\mathcal{T}}$ is the transition matrix of the Metropolis-Hastings projection chain on $\mathcal{T}$, and P the transition matrix of the degree interval Markov chain. The goal will be to show that β and γ, as well as all the spectral gaps, can be lower bounded by an inverse polynomial function of the form $1/p(n)$ for some polynomial $p(n)$. This means that $\mathrm{Gap}(P)$ is lower bounded by an inverse polynomial as well, which is equivalent to showing that the degree interval Markov chain is rapidly mixing (see Section 2.2).

Next we partition the sets $\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})$ further into sets $\mathcal{G}(\boldsymbol{d})$ for sequences $\boldsymbol{d}$ in $\mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u}) = \left\{\boldsymbol{d} : \sum_i d_i = 2m \text{ and } \boldsymbol{\ell} \leq \boldsymbol{d} \leq \boldsymbol{u}\right\}$. For this part of the decomposition we get a Metropolis-Hastings projection chain on the set $\mathcal{D}_m$. The restriction chains on $\mathcal{M}_{\mathcal{G}(\boldsymbol{d})}$ are the chains in which we essentially only apply switch operations on all graphs with degree sequence $\boldsymbol{d}$. This is precisely the switch Markov chain with some polynomially bounded holding probability (as defined in Corollary 14). Using one more time Theorem 8 for each m – again, with β_m and γ_m to be determined later – we have

$$\mathrm{Gap}(P_{\mathcal{G}_m(\boldsymbol{\ell},\boldsymbol{u})}) \geq \beta_m\gamma_m \cdot \mathrm{Gap}(P_{\mathcal{D}_m}) \cdot \min_{\boldsymbol{d} \in \mathcal{D}_m} \mathrm{Gap}(P_{\mathcal{G}(\boldsymbol{d})}), \tag{5}$$

where $P_{\mathcal{D}_m}$ is the transition matrix of the Metropolis-Hastings chain on $\mathcal{D}_m$. This time, in order to show that the switch-hinge flip Markov chain is rapidly mixing, we need to bound γ_m, β_m, and all the spectral gaps by an inverse polynomial function.

Combining (4) and (5) we now get

$$\mathrm{Gap}(P) \geq \beta\gamma \cdot \mathrm{Gap}(P_{\mathcal{T}}) \cdot \min_{m \in \mathcal{T}} \left\{\beta_m\gamma_m \cdot \mathrm{Gap}(P_{\mathcal{D}_m}) \cdot \min_{\boldsymbol{d} \in \mathcal{D}_m} \mathrm{Gap}(P_{\mathcal{G}(\boldsymbol{d})})\right\}, \tag{6}$$

and in order to show that the degree interval Markov chain is rapidly mixing, we need to show that β, γ, all β_m and γ_m, and all spectral gaps can be lower bounded by $1/q(n)$ for some polynomial $q(n)$.

While this is what we are going to do for Theorem 3, recall that for Theorem 2 (directly) and Theorem 1 (through Theorem 2 and the reductions in [2, Appendix C]), we only show that the switch-hinge flip Markov chain is rapidly mixing. In that case, it suffices to show that β_m, γ_m, and the spectral gaps involved in (5) are polynomially bounded for any given m (and the polynomial bound is independent of m), i.e., we only need to globally consider the second decomposition step. A polynomial lower bound on β and each one of the β_m follows by the very definition of the degree interval Markov chain (see also the discussion after its definition). In order to bound γ and γ_m, for all m, we use Lemma 13. Roughly speaking, we need to show that we can move rather easily between realizations of two degree sequences $\boldsymbol{d}$ and $\boldsymbol{d}'$, with $\sum_i |d_i - d'_i| = 2$. The high-level idea for γ_m is to show that it is either directly possible to perform a hinge flip in order to transition from a graph G with degree sequence $\boldsymbol{d}$ to some G' with degree sequence $\boldsymbol{d}'$, or that G is not too far away from

some other graph H with the same degree sequence $\boldsymbol{d}$ from which it is possible to directly move to some G' with degree sequence $\boldsymbol{d}'$ via a hinge flip. We take an analogous approach for bounding γ but in terms of addition/deletion operations rather than hinge flips.

The gaps of the chains $\mathcal{M}_{\mathcal{G}(\boldsymbol{d})}$ are globally bounded because of known rapid mixing results for the switch Markov chain [1] (Corollary 14). Therefore, in order to show Theorem 2, it remains to bound $\mathrm{Gap}(P_{\mathcal{D}_m})$, which we do in Appendix A.1; an outline is given in Section 3.1 below. For Theorem 3, we additionally need to bound $\mathrm{Gap}(P_{\mathcal{T}})$, which we do in Appendix A.2; a very brief outline is given in Section 3.2.

3.1 Proving Theorem 2

The main technical challenge of Theorem 2 lies in proving that the resulting Metropolis-Hastings projection chain on $\mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})$ is rapidly mixing, i.e., that $\mathrm{Gap}(P_{\mathcal{D}_m})$ can be polynomially bounded. We sometimes refer to this chain as the *hinge flip projection chain*. Note that for $\boldsymbol{d}, \boldsymbol{d}' \in \mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})$, with $||\boldsymbol{d} - \boldsymbol{d}'||_1 = 2$, it follows from (2) that

$$P_{\mathrm{MH}}(\boldsymbol{d}, \boldsymbol{d}') \geq \frac{1}{2n^2} \min\left\{1, \frac{|\mathcal{G}(\boldsymbol{d}')|}{|\mathcal{G}(\boldsymbol{d})|}\right\},$$

by taking the obvious upper bound $\Delta \leq n^2$ in (2). So, intuitively, whether or not the hinge flip projection chain is rapidly mixing depends on the quantities $|\mathcal{G}(\boldsymbol{d})|$ for $\boldsymbol{d} \in \mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})$. To this end, we first argue, using a comparison argument, that if suffices to show that the load-exchange Markov chain on $\mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})$, i.e., the Markov chain that allows us to move between degree sequences by adjusting the degree of two nodes by 1 (while keeping the degree sums fixed), is rapidly mixing for the weights $w(d) = |\mathcal{G}(\boldsymbol{d})|$. (It is not hard to see that $\mathcal{D}_m$ is in fact an M-convex set.) A Markov chain comparison argument, very informally speaking, proceeds by showing that if one Markov chain is rapidly mixing, and a second chain is very close to it (in terms of similar stationary distribution and transition probabilities), then the second chain is also rapidly mixing. In our setting, the comparison is based on the fact that both chains have the same stationary distribution π with $\pi(\boldsymbol{d}) \propto w(\boldsymbol{d})$, and the fact that their transition probabilities are polynomially related for the degree sequences that we are interested in (using the remark at the end of [2, Appendix B]).

In order to show that the load-exchange Markov chain on $\mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})$ is rapidly mixing, we would like to use Corollary 9, which states that the load-exchange Markov chain is rapidly mixing if a polynomial identified with its stationary distribution satisfies the property of strong log-concavity (SLC). To be precise, we may apply Corollary 9 if, for given $\boldsymbol{\ell}, \boldsymbol{u}$ and m, the polynomial

$$h(x) = \sum_{\boldsymbol{d} \in \mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})} w(\boldsymbol{d}) \cdot x^{\boldsymbol{d}} = \sum_{\boldsymbol{d} \in \mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})} |\mathcal{G}(\boldsymbol{d})| \cdot x^{\boldsymbol{d}}$$

is SLC. This seems hard to prove (and might not be true in general). However, it turns out that when replacing the weights $w(\boldsymbol{d})$ by by the approximations $\bar{w}(\boldsymbol{d})$ as given in the asymptotic formula (3) of Liebenau and Wormald [39], the resulting polynomial

$$\bar{h}(x) = \sum_{\boldsymbol{d} \in \mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})} \bar{w}(\boldsymbol{d}) \cdot x^{\boldsymbol{d}} \tag{7}$$

is in fact SLC, when considering the degree interval instances of Theorem 1.[4] We show this fact in Theorem 17 in Section 4 by observing that the polynomial in (7) is of the form (1) in Proposition 7, which is a general sufficient condition for a polynomial to be SLC [12].

[4] We remark at this point, that, although this is true for the regime considered in Theorem 1, this does not seem to be true for the general range in Theorem 10.

The above implies that if we run the load-exchange Markov chain with the approximations $\bar{w}(\boldsymbol{d})$, it is in fact rapidly mixing with stationary distribution $\bar{\pi}$ given by $\bar{\pi}(\boldsymbol{d}) \propto \bar{w}(\boldsymbol{d})$. Now, the approximations have the property that for some n_0 sufficiently large, it holds that for all $n \geq n_0$ and $\boldsymbol{d} \in \mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})$,

$$e^{-(19/\sigma)^2}|\mathcal{G}(\boldsymbol{d})| \leq \bar{w}(\boldsymbol{d}) \leq |\mathcal{G}(\boldsymbol{d})|\,.$$

This also implies that

$$e^{-(19/\sigma)^2}\pi(\boldsymbol{d}) \leq \bar{\pi}(\boldsymbol{d}) \leq e^{(19/\sigma)^2}\pi(\boldsymbol{d})\,.$$

One can then again use a Markov chain comparison argument to argue that the load-exchange Markov chain based on the original weights $w(\boldsymbol{d})$ is also rapidly mixing (based on the final remark in [2, Appendix B] with $\delta = 1 - e^{-(19/\sigma)^2}$). This in turn implies that the hinge flip projection chain is also rapidly mixing, which is what we wanted to show.

General framework. The approach described above for showing rapid mixing of the switch-hinge flip Markov chain might be applicable to other classes of degree interval instances. Informally speaking, the essential things that are needed are the following two things:

1. The degree sequences satisfying the interval constraints are strongly stable (see Def. 11).
2. The weights $|\mathcal{G}(\boldsymbol{d})|$ "approximately" give rise to an SLC polynomial.

The requirement of strong stability in the first point is needed for various reasons. First of all, it is a sufficient condition for the switch Markov chain (i.e., the restrictions chains in our decomposition) to be rapidly mixing [1]. Secondly, we rely on it when bounding the parameter γ in the Martin-Randall decomposition theorem (Theorem 8). Thirdly, strong stability is sufficient to argue that the transition probabilities of the load-exchange Markov chain, and the Metropolis-Hastings projection chain, are polynomially related (so that we can use a Markov chain comparison argument to compare their mixing times).

For the second point, even if the weights $|\mathcal{G}(\boldsymbol{d})|$ do not give rise to an SLC polynomial, one may still make things work. It suffices to find values $z(\boldsymbol{d})$ and polynomials q_1 and q_2 such that

$$\frac{1}{q_1(n)}|\mathcal{G}(\boldsymbol{d})| \leq z(\boldsymbol{d}) \leq q_2(n)|\mathcal{G}(\boldsymbol{d})|\,,$$

and for which

$$\bar{h}(x) = \sum_{\boldsymbol{d} \in \mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})} z(\boldsymbol{d}) \cdot x^{\boldsymbol{d}}$$

is SLC.

3.2 Proving Theorem 3

In order to prove Theorem 3, we additionally need to show that the projection chain on $\mathcal{T}$ is also rapidly mixing, i.e., that $\mathrm{Gap}(\mathcal{T})$ is polynomially bounded. In other words, we consider the Metropolis-Hastings projection Markov chain with state space $\{a, \dots, b\}$, where $a = \frac{1}{2}\sum_i \ell_i$ and $b = \frac{1}{2}\sum_i u_i$ (where we assume a and b to be integral here for the sake of simplicity), and $\pi(m) \propto |\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})|$ for $m \in \{a, \dots, b\}$. This chain will sometimes be referred to as the *addition/deletion projection chain*. A sufficient condition for this Markov chain to be rapidly mixing is that the sequence $(w_m)_{m=a,\dots,b}$ given by $w_m = |\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})|$ is log-concave,

meaning that for every m $w_{m+1}w_{m-1} \leq w_m^2$. We show log-concavity of this sequence to be true when the intervals have size at most one (corresponding to the statement of Theorem 3) by using a variation on an argument of Jerrum and Sinclair [33].

▶ Remark 15. One might wonder if the theory of strongly log-concave polynomials can also be used to prove rapid mixing for degree intervals beyond size one. For example, one might consider a polynomial of the form

$$g(x_1,\dots,x_n,y) = \sum_{m=m_1}^{m_2} \sum_{\boldsymbol{d}\in\mathcal{D}_m(\boldsymbol{\ell},\boldsymbol{u})} \binom{n-1}{2m_2-2m} \bar{z}(\boldsymbol{d}) \cdot y^{2m_2-2m} x^{\boldsymbol{d}},$$

where m_1 and m_2 are the minimum and maximum number of edges that any graph in $\mathcal{D}(\ell,u)$ can have, respectively. This is then a $2m_2$-homogeneous polynomial. The problem that now occurs though, is that the domain of this polynomial, indexed by the tuples $(d_1,\dots,d_n,2m_2-2m)$, can be shown not to be an M-convex set and, thus, g cannot be SLC.

4 SLC property in a restricted range of the Liebenau-Wormald result

Throughout this section, we consider m and n as fixed. Recall that for a given degree sequence $\boldsymbol{d} = (d_1,\dots,d_n)$ we defined $\xi = \xi(n,m) = \sum_i d_i/n = 2m/n$, $\mu = \mu(n,m) = \xi/(n-1) = 2m/(n(n-1))$ and $\chi(\boldsymbol{d}) = \sum_i (d_i-\xi)^2/(n-1)^2$. Furthermore, in (3) we defined

$$\bar{w}(\boldsymbol{d}) = \sqrt{2}\exp\left(\frac{1}{4} - s(\boldsymbol{d})^2\right)\left(\mu^\mu(1-\mu)^{(1-\mu)}\right)^{n(n-1)/2} \prod_i \binom{n-1}{d_i}, \tag{8}$$

which is approximately the number of graphs with degree sequence $\boldsymbol{d}$ in case it is near-regular. Here we have

$$s(\boldsymbol{d}) = \frac{\chi(\boldsymbol{d})}{2\mu(1-\mu)}.$$

In order to show that the weights $\bar{w}(\boldsymbol{d})$ give rise to an SLC polynomial,

$$\bar{h}(x) = \sum_{\boldsymbol{d}\in\mathcal{D}_m(\boldsymbol{\ell},\boldsymbol{u})} \bar{w}(\boldsymbol{d}) \cdot x^{\boldsymbol{d}},$$

it would be sufficient by Proposition 7 to argue that the function $s(\boldsymbol{d})^2$ is M-convex. Unfortunately, it turns out that this is not the case. Instead, we simply show that in the regime of Theorem 1, it holds that $s(\boldsymbol{d}) = O(1)$, and so we can ignore the contribution $\exp(-s(\boldsymbol{d})^2)$ in (8) at the expense of a slightly worse bound on the mixing time. The resulting approximation formula is easily seen to be SLC, which intuitively follows from a discrete form of log-concavity of the binomial coefficients; see Theorem 17.

▶ **Lemma 16.** *Under the conditions on $[\boldsymbol{\ell},\boldsymbol{u}]$ as in Theorem 1, with $0 < \sigma < 1$ and $2 \leq r = r(n) \leq (1-\sigma)n$, if n is large enough, then for any $\boldsymbol{\ell} \leq \boldsymbol{d} \leq \boldsymbol{u}$ it holds that*

$$0 \leq s(\boldsymbol{d}) \leq \frac{18n}{\sigma r^{1-2\alpha}(n-1)} \leq \frac{19}{\sigma} \tag{9}$$

Proof. By the definition of $\chi(\boldsymbol{d})$, $s(\boldsymbol{d}) \geq 0$ always holds. To see the upper bound, let $n_1 = \max\left\{\left\lceil (2/\sigma)^{\frac{1}{1-2\alpha}}\right\rceil, \lceil 18/\sigma\rceil\right\}$ and note that the quantity $s(\boldsymbol{d})$ can be rewritten as

$$s(\boldsymbol{d}) = \frac{\chi(\boldsymbol{d})}{2\mu(1-\mu)} = \frac{n^2(n-1)^2}{(n-1)^2} \frac{\sum_i (d_i-\xi)^2}{2\cdot 2m(n(n-1)-2m)}, \tag{10}$$

where $2m = \sum_i d_i$. Note that $\sum_i (d_i - \xi)^2 \leq n(2r^\alpha)^2 = 4nr^{2\alpha}$. Moreover, we can bound m using the simple facts that $r - r^\alpha \geq r/4$ and $n^\alpha \leq \sigma n/2$ for $n \geq n_1$. The latter implies that

$$r + r^\alpha \leq (1-\sigma)n + (1-\sigma)n^\alpha \leq (1-\sigma)n + \sigma n/2 = (1 - \sigma/2)n\,.$$

So, we have

$$n\frac{r}{4} \leq 2m \leq n\left(1 - \frac{\sigma}{2}\right)n\,,$$

and therefore,

$$2m(n(n-1) - 2m) \geq n\frac{r}{4}\left(1 - \left(1 - \frac{\sigma}{2}\right)\frac{n}{n-1}\right)n(n-1) \geq \frac{r\sigma n^2(n-1)}{9}\,, \tag{11}$$

where the last inequality holds because $\frac{n\sigma/2-1}{n-1} \geq \frac{4\sigma}{9}$ for $n \geq n_1$. By combining (10) and (11), we then get

$$s(\boldsymbol{d}) \leq \frac{n^2 \cdot 4nr^{2\alpha} \cdot 9}{2r\sigma n^2(n-1)} = \frac{18n}{\sigma r^{1-2\alpha}(n-1)}\,,$$

which completes the second inequality. The final inequality holds because $r \geq 2$ and $n/(n-1) \leq \frac{19}{18}$ for $n \geq n_1$. ◀

We next summarize the main result of this section, and give the remaining small technical steps of its proof. In a nutshell, it states that a simplified version of the Liebenau-Wormald formula which is within a constant factor from the original in (3) is approximately SLC in the regime of Theorem 1.

▶ **Theorem 17.** *For given $n, m \in \mathbb{N}$, $\boldsymbol{\ell}, u \in \mathbb{N}^n$ with $\boldsymbol{\ell} \leq \boldsymbol{u}$, and degree sequence $\boldsymbol{d}$ with $\sum_i d_i = 2m$ and $\boldsymbol{\ell} \leq \boldsymbol{d} \leq \boldsymbol{u}$, let*

$$\bar{z}(\boldsymbol{d}) = \sqrt{2}e^{\frac{1}{4}}\left(\mu^\mu(1-\mu)^{(1-\mu)}\right)^{n(n-1)/2}\prod_i \binom{n-1}{d_i}\,. \tag{12}$$

The resulting $2m$-homogeneous polynomial

$$\bar{f}(x) = \sum_{\boldsymbol{d} \in \mathcal{D}_m(\boldsymbol{\ell},\boldsymbol{u})} \bar{z}(\boldsymbol{d}) \cdot x^{\boldsymbol{d}}$$

is SLC.

Furthermore, there exists an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ and $m \geq n$, if the degree interval $[\boldsymbol{\ell}, \boldsymbol{u}]$ satisfies the conditions of Theorem 1, then

$$e^{-(19/\sigma)^2}|\mathcal{G}(\boldsymbol{d})| \leq \bar{z}(\boldsymbol{d}) \leq |\mathcal{G}(\boldsymbol{d})|. \tag{13}$$

for every $\boldsymbol{\ell} \leq \boldsymbol{d} \leq \boldsymbol{u}$.

Proof. We first note that the factors $\sqrt{2}$ and $\left(\mu^\mu(1-\mu)^{(1-\mu)}\right)^{n(n-1)/2}$ can all be seen as non-negative scalars as n and m are given. This means, by Proposition 6, that it suffices to show that the polynomial with coefficients

$$e^{\frac{1}{4}}\prod_i \binom{n-1}{d_i}$$

is SLC.

Comparing this to the second polynomial in Proposition 7, it follows that we can simply choose ν to be the constant function $\nu(\boldsymbol{d}) = -\frac{1}{4}$ on its effective domain $\mathcal{D}_m(\boldsymbol{\ell}, \boldsymbol{u})$. (As mentioned earlier, intuitively it is SLC because the binomial coefficients satisfy a discrete form of log-concavity.) The statement in (13) follows directly from Theorem 10 and Lemma 16. ◀

5 Discussion and future directions

We did not attempt to optimize the upper bounds on the mixing times of the Markov chains involved. Already for the *switch Markov chain* no low-degree polynomial upper bounds are known on its mixing time. For instance, the best known upper bound for r-regular graphs is $r^{23}n^8(rn\log(rn)+\log(1/\epsilon))$ [16, 17]. This is a central issue for many MCMC approaches for sampling graphs with given degrees (or degree intervals in our case). Various non-MCMC approaches to the problem, see, e.g., [5, 27, 28, 37, 43, 50], often have better running times, but only work for smaller classes of degree sequences or have weaker guarantees on the uniformity of the output than we require in our setting.

An interesting first direction for future work is determining whether the degree interval chain is rapidly mixing for more general instances. The most intriguing question from our point of view, however, is whether there is a black-box reduction implying that if the switch Markov chain is rapidly mixing for all degree sequences $\boldsymbol{d}$ satisfying $\boldsymbol{\ell} \leq \boldsymbol{d} \leq \boldsymbol{u}$, then the degree interval Markov chain is also rapidly mixing. Even more generally, can the problem of sampling graphs with given degree intervals always be reduced to the problem of sampling graphs with given degrees?

Further, one could explore other, non-MCMC, approaches for approximate sampling, especially when the degree ranges are relatively large. Can one come up with an algorithm in which resampling certain "bad events" (e.g., resampling edges adjacent to a node not satisfying its degree interval constraints) yields an exactly uniform sample, following the "partial rejection sampling" framework of Guo, Jerrum and Liu [30]? While this seems unlikely when sampling graphs with given degrees, we suspect it is possible for the problem of sampling graphs with (sufficiently large) given degree intervals.

References

1. Georgios Amanatidis and Pieter Kleer. Rapid mixing of the switch Markov chain for strongly stable degree sequences and 2-class joint degree matrices. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 966–985. SIAM, 2019.
2. Georgios Amanatidis and Pieter Kleer. Approximate sampling and counting of graphs with near-regular degree intervals. *CoRR*, abs/2110.09068, 2021. doi:10.48550/arXiv.2110.09068.
3. Nima Anari, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials, entropy, and a deterministic approximation algorithm for counting bases of matroids. In *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018*, pages 35–46. IEEE Computer Society, 2018.
4. Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. Log-concave polynomials II: high-dimensional walks and an FPRAS for counting bases of a matroid. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 1–12, 2019.
5. Mohsen Bayati, Jeong Han Kim, and Amin Saberi. A sequential algorithm for generating random graphs. *Algorithmica*, 58(4):860–910, 2010.
6. Edward A. Bender and E. Rodney Canfield. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory, Series A*, 24(3):296–307, 1978.
7. Ivona Bezáková, Nayantara Bhatnagar, and Eric Vigoda. Sampling binary contingency tables with a greedy start. *Random Structures & Algorithms*, 30(1-2):168–205, 2007.
8. Joseph Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet mathematics*, 6(4):489–522, 2011.
9. Sergey G Bobkov and Prasad Tetali. Modified logarithmic Sobolev inequalities in discrete settings. *Journal of Theoretical Probability*, 19(2):289–336, 2006.

10 Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1(4):311–316, 1980.

11 Petter Brändén. Personal communication, 2020.

12 Petter Brändén and June Huh. Lorentzian polynomials. *Annals of Mathematics*, 192(3):821–891, 2020.

13 Sergio Caracciolo, Andrea Pelissetto, and Alan D. Sokal. Two remarks on simulated tempering. *Unpublished manuscript*, 1992.

14 Corrie Jacobien Carstens and Pieter Kleer. Speeding up switch Markov chains for sampling bipartite graphs with given degree sequence. In *Proceedings of the 22nd International Conference on Randomization and Computation, RANDOM 2018*, volume 116 of *LIPIcs*, pages 36:1–36:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

15 Anthony C.C. Coolen, Alessia Annibale, and Ekaterina Roberts. *Generating random networks and graphs*. Oxford university press, 2017.

16 Colin Cooper, Martin E. Dyer, and Catherine S. Greenhill. Sampling regular graphs and a peer-to-peer network. *Combinatorics, Probability & Computing*, 16(4):557–593, 2007.

17 Colin Cooper, Martin E. Dyer, and Catherine S. Greenhill. Corrigendum: Sampling regular graphs and a peer-to-peer network. *CoRR*, abs/1203.6111, 2012. doi:10.48550/arXiv.1203.6111.

18 Colin Cooper, Martin E. Dyer, Catherine S. Greenhill, and Andrew J. Handley. The flip Markov chain for connected regular graphs. *Discrete Applied Mathematics*, 2018.

19 Mary Cryan, Heng Guo, and Giorgos Mousa. Modified log-Sobolev inequalities for strongly log-concave distributions. doi:10.48550/arXiv.1903.06081.

20 Martin E. Dyer, Catherine S. Greenhill, Pieter Kleer, James Ross, and Leen Stougie. Sampling hypergraphs with given degrees. *Discret. Math.*, 344(11):112566, 2021.

21 Péter L. Erdős, Ervin Győri, Tamaás Róbert Mezei, István Miklós, and Dániel Soltész. Half-graphs, other non-stable degree sequences, and the switch Markov chain. *The Electronic Journal of Combinatorics*, 28(3), 2021.

22 Péter L. Erdős, Catherine S. Greenhill, Tamás Róbert Mezei, István Miklós, Daniel Soltész, and Lajos Soukup. The mixing time of switch Markov chains: A unified approach. *Eur. J. Comb.*, 99:103421, 2022.

23 Péter L. Erdős, Tamás Róbert Mezei, and István Miklós. Approximate sampling of graphs with near-p-stable degree intervals. *CoRR*, abs/2204.09493, 2022. doi:10.48550/arXiv.2204.09493.

24 Péter L. Erdős, Tamás Róbert Mezei, István Miklós, and Dániel Soltész. Efficiently sampling the realizations of bounded, irregular degree sequences of bipartite and directed graphs. *PLOS ONE*, 13(8):1–20, August 2018.

25 Péter L. Erdős, István Miklós, and Zoltán Toroczkai. A decomposition based proof for fast mixing of a Markov chain over balanced realizations of a joint degree matrix. *SIAM Journal on Discrete Mathematics*, 29(1):481–499, 2015.

26 Tomás Feder, Adam Guetz, Milena Mihail, and Amin Saberi. A local switch Markov chain on given degree graphs with application in connectivity of peer-to-peer networks. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006*, pages 69–76. IEEE, 2006.

27 Pu Gao and Nicholas C. Wormald. Uniform generation of random graphs with power-law degree sequences. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1741–1758. SIAM, 2018.

28 Pu Gao and Nilocas C. Wormald. Uniform generation of random regular graphs. *SIAM Journal of Computing*, 46(4):1395–1427, 2017.

29 Catherine S. Greenhill and Matteo Sfragara. The switch Markov chain for sampling irregular graphs and digraphs. *Theor. Comput. Sci.*, 719:1–20, 2018.

30 Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovász local lemma. *Journal of the ACM (JACM)*, 66(3):18, 2019.

31 Leonid Gurvits. On multivariate Newton-like inequalities. In *Advances in combinatorial mathematics*, pages 61–78. Springer, 2009.

32 Mark Jerrum, Brendan D. McKay, and Alistair Sinclair. When is a graphical sequence stable? *Random Graphs, Vol. 2*, pages 101–116, 1992.

33 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18(6):1149–1178, 1989.

34 Mark Jerrum and Alistair Sinclair. Fast uniform generation of regular graphs. *Theoretical Computer Science*, 73(1):91–100, 1990.

35 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

36 Ravi Kannan, Prasad Tetali, and Santosh Vempala. Simple Markov-chain algorithms for generating bipartite graphs and tournaments. *Random Structures and Algorithms*, 14(4):293–308, 1999.

37 Jeong Han Kim and Van H. Vu. Generating random regular graphs. *Combinatorica*, 26(6):683–708, 2006.

38 Pieter Kleer. Sampling from the Gibbs distribution in congestion games. In *Proceedings of the 22nd ACM Conference on Economics and Computation, EC 2021*, pages 679–680. ACM, 2021.

39 Anita Liebenau and Nick Wormald. Asymptotic enumeration of graphs by degree sequence, and the degree sequence of a random graph. *CoRR*, abs/1702.08373, 2017. doi:10.48550/arXiv.1702.08373.

40 Neal Madras and Dana Randall. Markov chain decomposition for convergence rate analysis. *Ann. Appl. Probab.*, 12(2):581–606, May 2002.

41 Russell A. Martin and Dana Randall. Disjoint decomposition of Markov chains and sampling circuits in Cayley graphs. *Combinatorics, Probability & Computing*, 15(3):411–448, 2006.

42 Brendan D. McKay and Nicholas C. Wormald. Asymptotic enumeration by degree sequence of graphs of high degree. *European Journal of Combinatorics*, 11(6):565–580, 1990.

43 Brendan D. McKay and Nicholas C Wormald. Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, 11(1):52–67, 1990.

44 István Miklós, Péter L. Erdős, and Lajos Soukup. Towards random uniform sampling of bipartite graphs with given degree sequence. *Electronic Journal of Combinatorics*, 20(1), 2013.

45 Kazuo Murota. Discrete convex analysis. *Mathematical Programming*, 83(1-3):313–371, 1998.

46 Kazuo Murota. Recent developments in discrete convex analysis. In *Research Trends in Combinatorial Optimization*, pages 219–260. Springer, 2009.

47 Benjamin P. Olding and Patrick J. Wolfe. Inference for graphs and networks: Adapting classical tools to modern data. In *Data Analysis for Network Cyber-Security*, pages 1–31. World Scientific, 2014.

48 Steffen Rechner, Linda Strowick, and Matthias Müller-Hannemann. Uniform sampling of bipartite graphs with degrees in prescribed intervals. *Journal of Complex Networks*, 6(6):833–858, 2018.

49 Alistair Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Combinatorics, Probability and Computing*, 1:351–370, 1992.

50 Angelika Steger and Nicholas C. Wormald. Generating random regular graphs quickly. *Combinatroics, Probability & Computing*, 8(4):377–396, 1999.

A Decomposition of the degree interval Markov chain

In this section we give the missing details regarding the decomposition steps as outlined in Section 3.

A.1 Bounding β_m, γ_m and $\mathrm{Gap}(P_{\mathcal{D}_m})$

Throughout this section we assume that some $m \in \{m_1, \dots, m_2\}$ is fixed. Moreover, recall that we consider degree intervals of the form $[d_i, d_i + 1]$, or $[d_i, d_i]$, for $i \in [n]$. It is not hard to see that $\beta_m \geq (6n^4)^{-1}$. This rough polynomial bound follows directly from the transition probabilities of the degree interval Markov chain.

We first lower bound the γ_m in Lemma 18 below. By the definition of the hinge flip operation we have that for any $\boldsymbol{d}, \boldsymbol{d}' \in \mathcal{D}_m$, there is a strictly positive transition probability between $\boldsymbol{d}$ and $\boldsymbol{d}'$ if and only if $\sum_i |d_i - d'_i| = 2$. The proof of Lemma 18 follows from Lemma 13, where it is shown that for a graph with a given degree sequence, we can always find a graph with a slightly perturbed degree sequence that is close to the former in terms of symmetric difference (when the original sequences satisfies strong stability).

▶ **Lemma 18.** *There exists a polynomial $q_1(n)$ such that, for any feasible m and for all $\boldsymbol{d}, \boldsymbol{d}' \in \mathcal{D}_m$ with $\sum_i |d_i - d'_i| = 2$, we have $\pi_{\mathcal{D}_m}\left(\partial_{\boldsymbol{d}}\left(\mathcal{G}\left(\boldsymbol{d}'\right)\right)\right) \geq \frac{1}{q_1(n)} \pi_{\mathcal{D}_m}\left(\mathcal{G}\left(\boldsymbol{d}'\right)\right)$.*

Proof. Again assume that $n \geq n_1 = \left\lceil (2/\sigma)^{\frac{1}{1-2\alpha}} \right\rceil$. Let a and b be the unique nodes such that $d'_a = d_a + 1$ and $d'_b = d_b - 1$; note that the uniqueness of a, b follows from the condition $\sum_i |d_i - d'_i| = 2$. Let

$$\mathcal{H} = \{G \in \mathcal{G}(\boldsymbol{d}) : \exists c \in [n] \text{ such that } \{b, c\} \in E(G), \{a, c\} \notin E(G)\},$$

and note that it has the property

$$|\partial_{\boldsymbol{d}}(\mathcal{G}(\boldsymbol{d}'))| \geq \frac{1}{n}|\mathcal{H}|. \tag{14}$$

To see this, note that for a given $G \in \mathcal{H}$, we can perform the hinge flip that removes the edge $\{b, c\}$ and adds the edge $\{a, c\}$ to obtain an element in $\mathcal{G}(\boldsymbol{d}')$. Moreover, there can be at most n graphs $G \in \mathcal{H}$ that map onto a given $G' \in \partial_{\boldsymbol{d}}(\mathcal{G}(\boldsymbol{d}'))$, as there are at most n choices for c.

Moreover, using the second part of Lemma 13, we show that

$$|\mathcal{H}| \geq \frac{1}{n^{12}}|\mathcal{G}(\boldsymbol{d})|. \tag{15}$$

To see this, note that for any $G \in \mathcal{G}(\boldsymbol{d})$, we have $d_b = d'_b + 1 \geq 0$ which implies that b has at least one neighbor c in G. Now, if $\{a, c\} \notin E(G)$ we obtain an element in $\mathcal{H}$; otherwise, we can find a graph G' close to G (using Lemma 13) for which $\{a, c\} \notin E(G)$ while still $\{b, c\} \in E(G)$. As there are at most n^{12} graphs $G \in \mathcal{G}(\boldsymbol{d})$ that map to the same $G' \in \mathcal{H}$, the inequality (15) follows. Moreover, we also have $n^{10}|\mathcal{G}(\boldsymbol{d})| \geq |\mathcal{G}(\boldsymbol{d}')|$ which follows directly from Definition 11 and (the proof of) Proposition 12 (see [2, Appendix A]).

Combining the last observation with (14) and (15) then yields

$$|\partial_{\boldsymbol{d}}(\mathcal{G}(\boldsymbol{d}'))| \geq \frac{1}{q_1(n)}|\mathcal{G}(\boldsymbol{d}')|,$$

for $q_1(n) = n^{23}$. Dividing both sides by $\sum_{\boldsymbol{d} \in \mathcal{D}_m} |\mathcal{G}(\boldsymbol{d})|$, then gives the desired result. ◀

It remains to bound $\mathrm{Gap}(P_{\mathcal{D}_m})$. As explained in Section 3.1, the first step is to carry out a comparison argument with the load-exchange Markov chain with weights $w(\boldsymbol{d}) = |\mathcal{G}(\boldsymbol{d})|$ (so that it will be sufficient to study the mixing time of the latter). Remember that both the hinge flip projection chain, as well as the load-exchange chain have stationary distribution $\pi(\boldsymbol{d}) \propto w(\boldsymbol{d})$.

In what follows we write $\mathcal{M}_{\mathcal{D}_m} = (\mathcal{D}_m, P)$ for the (Metropolis-Hastings) hinge flip projection chain, and $\mathcal{M}'_{\mathcal{D}_m} = (\mathcal{D}_m, P')$ for the load-exchange chain on $\mathcal{D}_m$.

▶ **Lemma 19.** *There exists a polynomial $p(n)$ such that*

$$p(n)\,\mathrm{Gap}(P_{\mathcal{D}_m}) \geq \mathrm{Gap}(P'_{\mathcal{D}_m})\,.$$

for any $m \in \{m_1, \ldots, m_2\}$.

Proof (sketch). It suffices to show that there exists polynomials p_1 and p_2 such that, whenever $\boldsymbol{d}, \boldsymbol{f} \in \mathcal{D}_m$ satisfy $||\boldsymbol{d} - \boldsymbol{f}||_1 = 2$, then

$$\frac{1}{p_1(n)} \leq P(\boldsymbol{d}, \boldsymbol{f}), P'(\boldsymbol{d}, \boldsymbol{f}) \leq \frac{1}{p_2(n)}\,. \tag{16}$$

This then implies that the transition probabilities $P(\boldsymbol{d}, \boldsymbol{f})$ and $P'(\boldsymbol{d}, \boldsymbol{f})$ are themselves polynomially related. In turn, this implies the existence of the desired polynomial $p(n)$ as both chains have the same stationary distribution and therefore their spectral gaps are polynomially related (see [2, Appendix B]).

The existence of the polynomials in (16) follows from the fact that all near-regular degree sequences are strongly stable. First of all, it holds that the term $|\mathcal{G}(\boldsymbol{d}')|/|\mathcal{G}(\boldsymbol{d})|$ in the Metropolis-Hastings hinge flip projection chain can always be upper and lower bounded by a polynomial because of strong stability. Furthermore, in the load-exchange Markov chain we pick (in the second step) a new degree sequence $\boldsymbol{d}'$ proportional to $w(\boldsymbol{d}')$ over all possible choices of $\boldsymbol{d}'$ with $||\boldsymbol{d} - \boldsymbol{d}'||_1 = 2$ that respect the degree interval bounds. For a given $\boldsymbol{d}$, let $N(\boldsymbol{d})$ be the set of all such sequences $\boldsymbol{d}'$. Then the probability of transitioning to $\boldsymbol{d}'$ is (up to an additional polynomial factor because of the first step of the load-exchange Markov chain) equal to

$$\frac{|\mathcal{G}(\boldsymbol{d}')|}{\sum\limits_{\boldsymbol{f} \in N(\boldsymbol{d})} |\mathcal{G}(\boldsymbol{f})|}\,,$$

which can again be upper and lower bounded by a polynomial because of strong stability, and because $|N(\boldsymbol{d})| \leq n^2$. ◀

Lemma 19 implies that we may focus on bounding $\mathrm{Gap}(P'_{\mathcal{D}_m})$. Now, by the arguments given in Section 3.1 in combination with another simple comparison argument as in [2, Appendix B] and Theorem 17, it suffices to bound $\mathrm{Gap}(P''_{\mathcal{D}_m})$ where P'' is the transition matrix of the hinge flip Markov chain in which we replace the weights $w(\boldsymbol{d})$ by the approximations $\bar{z}(\boldsymbol{d})$ as in (12).

In Section 4, we showed that the polynomial in (13) is in fact SLC, so then Corollary 9 implies that the modified log-Sobolev constant of this chain can be lower bounded by a polynomial, which implies the same for the spectral gap by what is explained in [2, Appendix B]. This completes this section, and shows in particular that the switch-hinge flip Markov chain is rapidly mixing, which in turn completes the proof of Theorem 2.

A.2 Bounding β, γ and $\mathrm{Gap}(P_{\mathcal{T}})$

Recall that $\mathcal{M}_{\mathcal{T}}$ is the Metropolis-Hastings chain on the index set $\mathcal{T} = \{m_1, \ldots, m_2\}$. For simplicity, we use $w_m = |\mathcal{G}_m(\boldsymbol{\ell}, \boldsymbol{u})|$ to denote the number of feasible graphical realizations with m edges. Note that for any $m \in \mathcal{T}$ we have $\pi_{\mathcal{T}}(m) = w_m / \sum_{i \in \mathcal{T}} w_i$, and that $P_{\mathcal{T}}(m, m') > 0$ if and only if $|m - m'| \leq 1$. From the definition of the degree interval Markov chain, it immediately follows that $\beta \geq 1/q(n)$ for some polynomial $q(n)$. We lower bound γ in the following lemma following the same approach as for Lemma 18.

▶ **Lemma 20.** *There exists a polynomial $q_2(n)$ such that, for all $m, m' \in \mathcal{T}$ with $|m-m'|=1$, we have $\pi_{\mathcal{T}}(\partial_m(\mathcal{G}_{m'})) \geq \frac{1}{q_2(n)}\pi_{\mathcal{T}}(\mathcal{G}_{m'})$.*

Proof. Assume that $m' = m+1$ (the case $m' = m-1$ is analogous). Let $G \in \mathcal{G}_{\boldsymbol{d}}$ for some $\boldsymbol{d} \in \mathcal{D}_m$. Note that $m' \geq m_1 + 1 > m_1$, which implies that there are nodes i and j whose degrees in G are not equal to the upper bound of their degree interval. Note that the set

$$\mathcal{H} = \{G \in \mathcal{G}_m : \exists i,j \in [n] \text{ with } d_i(G) < u_i, d_j(G) < u_j \text{ and } \{i,j\} \notin E(G)\}$$

has the property that

$$|\partial_m(\mathcal{G}_{m'})| \geq \frac{1}{m+1}|\mathcal{H}|\,. \tag{17}$$

In order to see this, note that for any $G \in \mathcal{H}$ we can add the edge $\{i,j\}$ (recall that these nodes depend on the choice of G) to obtain an element in $\mathcal{G}_{m'}$. On the other hand,there can be at most $m+1$ graphs G that map onto a given graph $H \in \mathcal{G}_{m'}$ using this procedure. This gives the inequality (17).

Moreover, using the first part of Lemma 13 and following the same argument as in the proof of Lemma 18 it can be shown that

$$|\mathcal{H}| \geq \frac{1}{n^{12}}|\mathcal{G}_m|\,. \tag{18}$$

To see this, note that for any graph $G \in \mathcal{G}_m$, nodes i and j with $d_i(G) < u_i$ and $d_j(G) < u_j$ always exist, as $m < m_2$. Moreover, if $\{i,j\} \in E(G)$ we know from Lemma 13 that there is a graph G' with the same degree sequence not containing edge $\{i,j\}$ close to G.

We next show that $|\mathcal{G}_m| \geq |\mathcal{G}_{m'}|/p(n)$ for some polynomial $p(n)$. To see this, note that for any $G' \in |\mathcal{G}_{m'}|$ there exist nodes x and y such that $d_x(G') > \ell_x$ and $d_y(G') > \ell_y$ as $m' = m+1 > m_1$. If $\{x,y\} \in E(G')$ we can remove it to obtain an element of $|\mathcal{G}_m|$. Otherwise, again using Lemma 13 we can first find an element $G'' \in \mathcal{G}_{m'}$ close to G' that contains $\{x,y\}$ and the remove it. Combining this with (18) yields the existence of a polynomial $q_2(n)$ such that

$$|\mathcal{H}| \geq \frac{1}{q_2(n)}|\mathcal{G}_{m'}|\,.$$

Finally, combining the latter inequality with (17) and dividing both sides by $\sum_{m\in\mathcal{T}} w_m$, then gives the desired result. ◀

In order to show that $\mathcal{M}_{\mathcal{T}}$ is rapidly mixing or, in particular, that the gap $\mathrm{Gap}(\mathcal{T})$ can be polynomially bounded, it is sufficient to show that the sequence $(w_m)_{m\in\mathcal{T}}$ is *log-concave*. Log-concavity means that for any $m \in \mathcal{T} \setminus \{m_1, m_2\}$, $w_{m-1}w_{m+1} \leq w_m^2$.

▶ **Theorem 21.** *The sequence $(w_m)_{m\in\mathcal{T}}$ is log-concave for all interval sequences $[\boldsymbol{\ell}, \boldsymbol{u}]$ for which $u_i \in \{\ell_i, \ell_i + 1\}$ for all $i \in [n]$.*

Proof. We follow the notation, terminology and general outline of the proof of Theorem 5.1 in [33]. Define $A = \mathcal{G}_{m+1} \times \mathcal{G}_{m-1}$ and $B = \mathcal{G}_m \times \mathcal{G}_m$. We will show that $|A| \leq |B|$, from which the claim follows.

Note that the symmetric difference of any two subgraphs of K_n can be decomposed into a collection of alternating cycles and simple paths. We will do this in a canonical way.[5] Fix

[5] This decomposition is the main extra step we need compared to the proof of Theorem 5.1 in [33]. The symmetric difference of two matchings is by construction already a disjoint union of cycles and paths. This is also where the analysis breaks down in case the degree intervals have length at least two.

some total order $\preceq_e$ on the edges of K_n. For two subgraphs G and G' we will call edges in $E(G) \setminus E(G')$ blue, and edges in $E(G') \setminus E(G)$ red. Around every node, we will pair up blue edges with red edges as much as possible. We do this by repeatedly selecting a node and pairing up the lowest ordered red and blue edge that have not yet been paired up. This yields a decomposition of the symmetric difference into i) alternating red-blue cycles, ii) alternating simple red-blue paths of even length (with same number of red and blue edges), iii) simple paths ending and starting with a red edge, iv) simple paths ending and starting with a blue edge. We call this the *canonical symmetric difference decomposition of* $E(G) \triangle E(G')$ *with respect to* $\preceq_e$, or simply the canonical decomposition of $E(G) \triangle E(G')$. We call a simple path a G-path if it contains one more edge of G than of G' (i.e., red edges are one more than blue edges), and a G'-path if it contains one more edge of G'. We emphasize that any path of odd length in the symmetric difference is of one of these two types.

Now, for every pair $(G, G') \in A$ it holds that the number of G-paths exceeds the number of G'-paths by precisely two (as G has two edges more than G'). For this reason, we partition A into disjoint classes $\{A_r : r = 1, \dots, m\}$ where

$$A_r = \{(G, G') \in A : \begin{array}{l} \text{the canonical decomposition of } E(G) \triangle E(G') \\ \text{contains } r+1 \text{ } G\text{-paths and } r-1 \text{ } G'\text{-paths}\}\,. \end{array}$$

In order to prove $|A| \leq |B|$ it suffices to show $|A_r| \leq |B_r|$ for all r. We call a pair $(L, L') \in B$ *reachable* from $(G, G') \in A$ if and only if $E(G) \triangle E(G) = E(L) \triangle E(L')$ and L is obtained by from G by taking some G-path in the canonical decomposition and flipping the parity of the edges with respect to G and G'. It is important to see that the canonical symmetric difference decomposition of the pairs (G, G') and (L, L') is the same because all degree intervals have length one. Note that the number of pairs in B_r reachable from a given $(G, G') \in A_r$ is precisely the number of G-paths in the canonical decomposition of G and G', which is $r+1$. Conversely, any given $(L, L') \in B_r$ is reachable from precisely r pairs in A_r. Therefore, if $|A_r| > 0$, we have

$$\frac{|B_r|}{|A_r|} = \frac{r+1}{r} > 1\,.$$

This proves the claim. ◀

We are ready to bound the spectral gap of $P_{\mathcal{T}}$. Note that Ω in the statement of Theorem 22 is actually $\mathcal{T}$. Recall that $|\mathcal{T}| = m_2 - m_1 + 1 \leq n/2 + 1 \leq n$. Moreover, the ratios w_i/w_j are also polynomially bounded for any $i, j \in \mathcal{T}$ with $|i - j| = 1$. This can be shown exactly as in the proofs of the Lemmata 18 and 20; see also [2, Appendix C]. As a result, it is sufficient to prove the statement in Theorem 22 below in order to bound the gap of $P_{\mathcal{T}}$.

▶ **Theorem 22.** *Let* $(w_m)_{m \in \Omega}$ *be a log-concave sequence of non-negative numbers and let* $\mathcal{M} = (\Omega, P)$ *be a Markov chain with transition probabilities*

$$P(i,j) = \begin{cases} \frac{1}{4} \min\{1, w_j/w_i\} & \text{if } |i-j| = 1\,, \\ 0 & \text{if } |i-j| > 1\,, \\ 1 - P(i, i-1) - P(i, i+1) & \text{if } i = j\,. \end{cases}$$

Then $1/\operatorname{Gap}(P) \leq 4\,|\Omega|^3 \max_{i,j:|i-j|=1} w_i/w_j$.[6]

[6] We suspect a similar result is true without the dependence on the w_i but this is not needed for our purpose.

Proof. First note that the stationary distribution π of $\mathcal{M}$ is proportional to the weights $(w_i)_{i\in\Omega}$, i.e., $\pi(i) = w_i/\sum_{p\in\Omega} w_p$, as desired. We bound the congestion of the straightforward multi-commodity flow f in which we route $\pi(i)\,\pi(j)$ units of flow over the path $i \to (i+1) \to \cdots \to j$ if $i < j$, or $i \to (i-1) \to \cdots \to j$ if $i > j$.

We consider a fixed transition $e = (z, z+1)$. Note that the proof for transitions of the form $(z, z-1)$ is symmetric, since a sequence $(w_i)_{i\in\Omega}$ is log-concave if and only if the sequence $(w_{|\Omega|-i+1})_{i\in\Omega}$ is log-concave. We have

$$\begin{aligned}\mathrm{load}(e) &= \sum_{1\le i\le z}\sum_{z<j\le|\Omega|}\frac{\pi(i)\,\pi(j)}{\pi(z)P(z,z+1)} \le 4\max_{i,j:|i-j|=1}\frac{w_i}{w_j}\sum_{1\le i\le z}\sum_{z<j\le|\Omega|}\frac{\pi(i)\,\pi(j)}{\pi(z)}\\ &= 4\max_{i,j:|i-j|=1}\frac{w_i}{w_j}\left(\sum_{p\in\Omega}w_p\right)^{-1}\sum_{1\le i\le z}\sum_{z<j\le|\Omega|}\frac{w_iw_j}{w_z}\,. \end{aligned} \tag{19}$$

Log-concavity of the sequence $(w_q)_{q\in\Omega}$ implies that for any fixed $i < j$, and any $a \in \mathbb{N}$ such that $i + a \le j - a$, we have

$$w_iw_j \le w_{i+a}w_{j-a}\,. \tag{20}$$

This follows from repeatedly applying the log-concavity condition. Indeed, log-concavity gives us $\frac{w_i}{w_{i+1}} \le \frac{w_{i+1}}{w_{i+2}} \le \ldots \le \frac{w_{j-2}}{w_{j-1}} \le \frac{w_{j-1}}{w_j}$ and thus $w_iw_j \le w_{i+1}w_{j-1}$. By repeating this with $i+1$ and $j-1$ (i.e., by removing the outer terms) we get $\frac{w_{i+1}}{w_{i+2}} \le \ldots \le \frac{w_{j-2}}{w_{j-1}}$ and thus $w_{i+1}w_{j-1} \le w_{i+2}w_{j-2}$. After a steps we get (20).

Now, for a fixed i and j in the double summation in (19), let a_{ij} be such that $w_{i+a_{ij}}$ or $w_{j-a_{ij}}$ (or both) equals w_z. Then (20) gives us that $w_iw_j \le w_z\,w_p$ for some $p \in \Omega$. Note that for any choice of z, the double summation in (19) has at most $|\Omega|^2$ terms (as there are at most $|\Omega|$ choices for i and j). This implies that

$$\sum_{1\le i\le z}\sum_{z<j\le|\Omega|} w_iw_j/w_z \le |\Omega|^2\sum_{p\in\Omega} w_zw_p/w_z = |\Omega|^2\sum_{p\in\Omega} w_p\,.$$

Combining this inequality with (19), we obtain

$$\mathrm{load}(e) \le 4|\Omega|^2\max_{i,j:|i-j|=1} w_i/w_j\,.$$

The proof is completed once we notice that, by the definition of the flow f, $\mathrm{length}(f) \le |\Omega|$. ◀

This then completes the proof of Theorem 3.

Enumerating Regular Languages with Bounded Delay

Antoine Amarilli
LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Mikaël Monet
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract

We study the task, for a given language L, of enumerating the (generally infinite) sequence of its words, without repetitions, while bounding the *delay* between two consecutive words. To allow for delay bounds that do not depend on the current word length, we assume a model where we produce each word by editing the preceding word with a small edit script, rather than writing out the word from scratch. In particular, this witnesses that the language is *orderable*, i.e., we can write its words as an infinite sequence such that the Levenshtein edit distance between any two consecutive words is bounded by a value that depends only on the language. For instance, $(a + b)^*$ is orderable (with a variant of the Gray code), but $a^* + b^*$ is not.

We characterize which regular languages are enumerable in this sense, and show that this can be decided in PTIME in an input deterministic finite automaton (DFA) for the language. In fact, we show that, given a DFA A, we can compute in PTIME automata $A_1, \ldots, A_t$ such that $\mathrm{L}(A)$ is partitioned as $\mathrm{L}(A_1) \sqcup \ldots \sqcup \mathrm{L}(A_t)$ and every $\mathrm{L}(A_i)$ is orderable in this sense. Further, we show that the value of t obtained is optimal, i.e., we cannot partition $\mathrm{L}(A)$ into less than t orderable languages.

In the case where $\mathrm{L}(A)$ is orderable (i.e., $t = 1$), we show that the ordering can be produced by a bounded-delay algorithm: specifically, the algorithm runs in a suitable pointer machine model, and produces a sequence of bounded-length edit scripts to visit the words of $\mathrm{L}(A)$ without repetitions, with bounded delay – exponential in $|A|$ – between each script. In fact, we show that we can achieve this while only allowing the edit operations *push* and *pop* at the beginning and end of the word, which implies that the word can in fact be maintained in a double-ended queue.

By contrast, when fixing the distance bound d between consecutive words and the number of classes of the partition, it is NP-hard in the input DFA A to decide if $\mathrm{L}(A)$ is orderable in this sense, already for finite languages.

Last, we study the model where push-pop edits are only allowed at the end of the word, corresponding to a case where the word is maintained on a stack. We show that these operations are strictly weaker and that the *slender languages* are precisely those that can be partitioned into finitely many languages that are orderable in this sense. For the slender languages, we can again characterize the minimal number of languages in the partition, and achieve bounded-delay enumeration.

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

Keywords and phrases Regular language, constant-delay enumeration, edit distance

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.8

Related Version *Full Version*: `https://arxiv.org/abs/2209.14878` [5]

Funding *Antoine Amarilli*: Partially supported by the ANR project EQUUS ANR-19-CE48-0019 and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758.

Acknowledgements We thank Florent Capelli and Charles Paperman for their insights during initial discussions about this problem. We thank user pcpthm on the Theoretical Computer Science Stack Exchange forum for giving the argument for Proposition 6.1 in [18]. We thank Jeffrey Shallit for pointing us to related work. We thank Torsten Mütze and Arturo Merino for other helpful pointers. We thank the anonymous reviewers for their valuable feedback. Finally, we are grateful to Louis Jachiet and Lê Thành Dũng (Tito) Nguyễn for feedback on the draft.

© Antoine Amarilli and Mikaël Monet;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Enumeration algorithms [23, 26] are a way to study the complexity of problems beyond decision or function problems, where we must produce a large number of outputs without repetitions. In such algorithms, the goal is usually to minimize the worst-case *delay* between any two consecutive outputs. The best possible bound is to make the delay *constant*, i.e., independent from the size of the input. This is the case, for example, when enumerating the results of acyclic free-connex conjunctive queries [7] or of MSO queries over trees [6, 14].

Unfortunately, constant-delay is an unrealistic requirement when the objects to enumerate can have unbounded size, simply because of the time needed to write them out. Faced by this problem, one option is to neglect this part of the running time, e.g., following Ruskey's "Do not count the output principle" [21, p. 8]. In this work, we address this challenge in a different way: we study enumeration where each new object is not written from scratch but produced by *editing the previous object*, by a small sequence of edit operations called an *edit script*. This further allows us to study the enumeration of *infinite* collections of objects, with an algorithm that runs indefinitely and ensures that each object is produced after some finite number of steps, and exactly once. The size of the edit scripts must be *bounded*, i.e., it only depends on the collection of objects to enumerate, but not on the size of the current object. The algorithm thus outputs an infinite series of edit scripts such that applying them successively yields the infinite collection of all objects. In particular, the algorithm witnesses that the collection admits a so-called *ordering*: it can be ordered as an infinite sequence with a bound on the *edit distance* between any two consecutive objects, namely, the number of edit operations.

In this paper, we study enumeration for regular languages in this sense, with the Levenshtein edit distance and variants thereof. One first question is to determine if a given regular language L admits an ordering, i.e., can we order its words such that the Levenshtein distance of any two consecutive words only depends on L and not on the word lengths? For instance, the language a^* is easily orderable in this sense. The language a^*b^* is orderable, e.g., following any Hamiltonian path on the infinite $\mathbb{N} \times \mathbb{N}$ grid. More interestingly, the language $(a + b)^*$ is orderable, for instance by considering words by increasing length and using a *Gray code* [17], which enumerates all n-bit words by changing only one bit at each step. More complex languages such as $a(a + bc)^* + b(cb)^*ddd^*$ can also be shown to be orderable (as our results will imply). However, one can see that some languages are not orderable, e.g., $a^* + b^*$. We can nevertheless generalize orderability by allowing multiple "threads": then we can partition $a^* + b^*$ as a^* and b^*, both of which are orderable. This leads to several questions: Can we characterize the orderable regular languages? Can every regular language be partitioned as a finite union of orderable languages? And does this lead to a (possibly multi-threaded) enumeration algorithm with bounded delay (i.e., depending only on the language but not on the current word length)?

Contributions. The present paper gives an affirmative answer to these questions. Specifically, we show that, given a DFA A, we can decide in PTIME if $\mathrm{L}(A)$ is orderable. If it is not, we can compute in PTIME DFAs $A_1, \ldots, A_t$ partitioning the language as $\mathrm{L}(A) = \mathrm{L}(A_1) \sqcup \ldots \sqcup \mathrm{L}(A_t)$ such that each $\mathrm{L}(A_i)$ is orderable; and we show that the t given in this construction is optimal, i.e., no smaller such partition exists. If the language *is* orderable (i.e., if $t = 1$), we show in fact that the same holds for a much more restricted notion of distance, the *push-pop distance*, which only allows edit operations at the beginning and end of the word. The reason we are interested in this restricted edit distance is that edit scripts featuring push and pop can be

easily applied in constant-time to a word represented in a double-ended queue; by contrast, Levenshtein edit operations are more difficult to implement, because they refer to integer word positions that change whenever characters are inserted or deleted.[1]

And indeed, this result on the push-pop distance then allows us to design a bounded-delay algorithm for $\mathrm{L}(A)$, which produces a sequence of bounded edit scripts of push or pop operations that enumerates $\mathrm{L}(A)$. The length of the edit scripts is polynomial in $|A|$ and the delay of our algorithm is exponential in $|A|$, but crucially it remains bounded throughout the (generally infinite) execution of the algorithm, and does not depend on the size of the words that are achieved. Formally, we show:

▶ **Result 1.** *Given a DFA A, one can compute in PTIME automata $A_1, \ldots, A_t$ for some $t \leqslant |A|$ such that $\mathrm{L}(A)$ is the disjoint union of the $\mathrm{L}(A_i)$, and we can enumerate each $\mathrm{L}(A_i)$ with bounded delay for the push-pop distance with distance bound $48|A|^2$ and exponential delay in $|A|$. Further, $\mathrm{L}(A)$ has no partition of cardinality $t-1$ into orderable languages, even for the Levenshtein distance.*

Thus, we show that orderability and enumerability, for the push-pop or Levenshtein edit distance, are in fact all logically equivalent on regular languages, and we characterize them (and find the optimal partition cardinality) in PTIME. By contrast, as was pointed out in [18], testing orderability for a fixed distance d is NP-hard in the input DFA, even for finite languages.

Last, we study the *push-pop-right distance*, which only allows edits at the end of the word. The motivation for studying this distance is that it corresponds to enumeration algorithms in which the word is maintained on a stack. We show that, among the regular languages, the *slender languages* [19] are then precisely those that can be partitioned into finitely many orderable languages, and that these languages are themselves enumerable. Further, the optimal cardinality of the partition can again be computed in PTIME:

▶ **Result 2.** *Given a DFA A, then $\mathrm{L}(A)$ is partitionable into finitely many orderable languages for the push-pop-right distance if and only if $\mathrm{L}(A)$ is slender (which we can test in PTIME in A). Further, in this case, we can compute in PTIME the smallest partition cardinality, and each language in the partition is enumerable with bounded delay with distance bound $2|A|$ and linear delay in $|A|$.*

In terms of proof techniques, our PTIME characterization of Result 1 relies on a notion of *interchangeability* of automaton states, defined via paths between states and via states having common loops. We then show orderability by establishing *stratum-connectivity*, i.e., for any *stratum* of words of the language within some length interval, there are finite sequences obeying the distance bound that connect any two words in that stratum. We show stratum-connectivity by pumping and de-pumping loops close to the word endpoints. We then deduce an ordering from this by adapting a standard technique [25] of visiting a spanning tree and enumerating even and odd levels in alternation (see also [22, 13]). The bounded-delay enumeration algorithm then proceeds by iteratively enlarging a custom data structure called a *word DAG*, where the construction of the structure for a stratum is amortized by enumerating the edit scripts to achieve the words of the previous stratum.

[1] There is, in fact, an $\Omega(\log |w| / \log \log |w|)$ lower bound on the complexity of applying Levenshtein edit operations and querying which letter occurs at a given position: crucially, this bound depends on the size of the word. See `https://cstheory.stackexchange.com/q/46746` for details. This is in contrast to the application of push-pop-right edit operations, which can be performed in constant time (independent from the word length) when the word is stored in a double-ended queue.

Related work. As we explained, enumeration has been extensively studied for many structures [26]. For regular languages specifically, some authors have studied the problem of enumerating their words in *radix order* [15, 1, 2, 10]. For instance, the authors of [1, 2] provide an algorithm that enumerates all words of a regular language in that order, with a delay of $O(|w|)$ for w the next word to enumerate. Thus, this delay is not bounded, and the requirement to enumerate in radix order makes it challenging to guarantee a bounded distance between consecutive words (either in the Levenshtein or push-pop distance), which is necessary for bounded-delay enumeration in our model. Indeed, our results show that not all regular languages are orderable in our sense, whereas their linear-delay techniques apply to all regular languages.

We have explained that enumeration for $(a + b)^*$ relates to Gray codes, of which there exist several variants [17]. Some variants, e.g., the so-called *middle levels problem* [16], aim at enumerating binary words of a restricted form; but these languages are typically finite (i.e., words of length n), and their generalization is typically not regular. While Gray codes typically allow arbitrary substitutions, one work has studied a variant that only allows restricted operations on the endpoints [9], implying the push-pop orderability of the specific language $(a + b)^*$.

Independently, some enumeration problems on automata have been studied recently in the database theory literature, in particular for *document spanners* [8], which can be defined by finite automata with capture variables. It was recently shown [11, 3] that we can enumerate in constant delay all possible assignments of the capture variables of a fixed spanner on an input word. In these works, the delay is constant in *data complexity*, which means that it only depends on the (fixed) automaton, and does not depend on the word; this matches what we call *bounded delay* in our work (where there is no input word and the automaton is given as input). However, our results do not follow from these works, which focus on the enumeration of results of constant size. Some works allow second-order variables and results of non-constant size [4] but the delay would then be linear in each output, hence unbounded.

Paper structure. We give preliminaries in Section 2. In Section 3 we present our PTIME construction of a partition of a regular language into finitely many orderable languages, and prove that the cardinality of the obtained partition is minimal for orderability. We then show in Section 4 that each term of the union is orderable, and then that it is enumerable in Section 5. We present the NP-hardness result on testing orderability for a fixed distance and our results on push-pop-right operations in Section 6. We conclude and mention some open problems in Section 7. Due to space constraints, we mostly present the general structure of the proofs and give the main ideas; detailed proofs of all statements can be found in the extended version of this work [5].

2 Preliminaries

We fix a finite non-empty *alphabet* Σ of *letters*. A *word* is a finite sequence $w = a_1 \cdots a_n$ of letters. We write $|w| = n$, and write ϵ for the empty word. We write Σ^* the infinite set of words over Σ. A *language* L is a subset of Σ^*. For $k \in \mathbb{N}$, we denote $L^{<k}$ the language $\{w \in L \mid |w| < k\}$. In particular we have $L^{<0} = \varnothing$.

In this paper we study *regular languages*. Recall that such a language can be described by a *deterministic finite automaton* (DFA) $A = (Q, \Sigma, q_0, F, \delta)$, which consists of a finite set Q of *states*, an *initial state* $q_0 \in Q$, a set $F \subseteq Q$ of *final states*, and a partial *transition function* $\delta \colon Q \times \Sigma \to Q$. We write $|A|$ the size of representing A, which is $O(|Q| \times |\Sigma|)$. A *(directed)*

path in A from a state $q \in Q$ to a state $q' \in Q$ is a sequence of states $q = q_0, \ldots, q_n = q'$ where for each $0 \leqslant i < n$ we have $q_{i+1} = \delta(q_i, a_i)$ for some a_i. For a suitable choice $a_0, \ldots, a_{n-1}$, we call the word $a_0 \cdots a_{n-1} \in \Sigma^*$ a *label* of the path. In particular, there is an empty path with label ϵ from every state to itself. The *language* $\mathrm{L}(A)$ accepted by A consists of the words w that label a path from q_0 to some final state. We assume without loss of generality that all automata are *trimmed*, i.e., every state of Q has a path from q_0 and has a path to some final state; this can be enforced in linear time.

Edit distances. For an alphabet Σ, we denote by $\delta_{\mathrm{Lev}} \colon \Sigma^* \times \Sigma^* \to \mathbb{N}$ the *Levenshtein edit distance*: given $u, v \in \Sigma^*$, the value $\delta_{\mathrm{Lev}}(u, v)$ is the minimum number of *edits* needed to transform u into v, where the edit operations are single-letter *insertions*, *deletions* or *substitutions* (we omit their formal definitions).

While our lower bounds hold for the Levenshtein distance, our positive results already hold with a restricted set of $2|\Sigma| + 2$ edit operations called the *push-pop edit operations*: $\mathrm{pushL}(a)$ and $\mathrm{pushR}(a)$ for $a \in \Sigma$, which respectively insert a at the beginning and at the end of the word, and $\mathrm{popL}()$ and $\mathrm{popR}()$, which respectively remove the first and last character of the word (and cannot be applied if the word is empty). Thus, we define the *push-pop edit distance*, denoted δ_{pp}, like δ_{Lev} but allowing only these edit operations.

Orderability. Fixing a distance function $\delta \colon \Sigma^* \times \Sigma^* \to \mathbb{N}$ over Σ^*, for a language $L \subseteq \Sigma^*$ and $d \in \mathbb{N}$, a *d-sequence in L* is a (generally infinite) sequence $\boldsymbol{s}$ of words $w_1, \ldots, w_n, \ldots$ of L *without repetition*, such that for every two consecutive words w_i, w_{i+1} in $\boldsymbol{s}$ we have $\delta(w_i, w_{i+1}) \leqslant d$. We say that $\boldsymbol{s}$ *starts at* w_1 and, in case $\boldsymbol{s}$ is finite and has n elements, that $\boldsymbol{s}$ *ends at* w_n (or that $\boldsymbol{s}$ is *between* w_1 *and* w_n). A *d-ordering of L* is a d-sequence $\boldsymbol{s}$ in L such that every word of L occurs in $\boldsymbol{s}$; equivalently, it is a permutation of L such that any two consecutive words are at distance at most d. An *ordering* is a d-ordering for some $d \in \mathbb{N}$. If these exist, we call the language L, respectively, *d-orderable* and *orderable*. We call L *(t, d)-partition-orderable* if it can be partitioned into t languages that each are d-orderable:

▶ **Definition 2.1.** *Let L be a language and $t, d \in \mathbb{N}$. We call L* (t, d)-partition-orderable *if L has a partition $L = \bigsqcup_{1 \leqslant i \leqslant t} L_i$ such that each L_i is d-orderable.*[2]

Note that, if we allowed repetitions in d-orderings, then the language of any DFA A would be $O(|A|)$-orderable: indeed, any word w can be transformed into a word w' of length $O(|A|)$ by iteratively removing simple loops in the run of w. By contrast, we will see in Section 3 that allowing a *constant* number of repetitions of each word makes no difference.

▶ **Example 2.2.** We consider the Levenshtein distance in this example. The language $(aa)^*$ is $(1, 2)$-partition-orderable (i.e., 2-orderable) and not $(k, 1)$-partition-orderable for any $k \in \mathbb{N}$. The language $a^* + b^*$ is $(2, 1)$-partition-orderable and not orderable, i.e., not d-orderable for any $d \in \mathbb{N}$. Any finite language is d-orderable with d the maximal length of a word in L. The non-regular language $\{a^{n^2} \mid n \in \mathbb{N}\}$ is not (t, d)-partition-orderable for any $t, d \in \mathbb{N}$.

Enumeration algorithms. We study *enumeration algorithms*, which output a (generally infinite) sequence of *edit scripts* $\sigma_1, \sigma_2, \ldots$. We only study enumeration algorithms where each *edit script* σ_i is a finite sequence of push-pop edit operations. The algorithm enumerates a language L if the sequence satisfies the following condition: letting w_1 be the result of

[2] We use $\bigsqcup$ for disjoint unions.

applying σ_1 on the empty word, w_2 be the result of applying σ_2 to w_1, and so on, then all w_i are distinct and $L = \{w_1, w_2, \ldots\}$. If L is infinite then the algorithm does not terminate, but the infinite sequence ensures that every $w \in L$ is produced as the result of applying (to ϵ) some finite prefix $\sigma_1, \ldots, \sigma_n$ of the output.

We aim for *bounded-delay* algorithms, i.e., each edit script must be output in time that only depends on the language L that is enumerated, but not on the current length of the words. Formally, the algorithm can emit any push-pop edit operation and a delimiter Output, it must successively emit the edit operations of σ_i followed by Output, and there is a bound $T > 0$ (the delay) depending only on L such that the first Output is emitted at most T operations after the beginning of the algorithm, and for each $i > 1$ the i-th Output is emitted at most T operations after the $(i-1)$-th Output. Note that our notion of delay also accounts for what is usually called the preprocessing phase in the literature, i.e., the phase before the first result is produced. Crucially the words w_i obtained by applying the edit scripts σ_i are not written, and T does not depend on their length.

We say that a bounded-delay algorithm *d-enumerates* a language L if it produces a d-ordering of L (for the push-pop distance). Thus, if L is d-enumerable (by an algorithm), then L is in particular d-orderable, and we will show that for regular languages, the converse also holds.

▶ **Example 2.3.** Consider the regular language $L := a^*b^* + b^*a^*$. This language is 2-orderable for the push-pop distance. Indeed, we can order it by increasing word length, finishing for word length i by the word a^i as follows. We start by length zero with the empty word ϵ (so the first edit script is empty), then, assuming we have ordered all words of L of size $\leqslant i$ while finishing with a^i, we continue with words of L of size $i+1$ in the following manner: we push-right the letter b to obtain a^ib, and then we "shift" with edit scripts of the form $(\mathrm{pushR}(b); \mathrm{popL}())$ until we obtain b^{i+1}, and then we shift again with edit scripts of the form $(\mathrm{pushR}(a); \mathrm{popL}())$ until we obtain a^{i+1} as promised. This gives us an enumeration algorithm for L, shown in Algorithm 1. As such, the delay of Algorithm 1 is not bounded, because of the time needed to increment the integer variable *size*: this variable becomes arbitrarily large throughout the enumeration, so it is not realistic to assume that we can increment it in constant time. This can however be fixed by working in a suitable *pointer machine model*, as explained next.

Note that our enumeration algorithms run indefinitely, and thus use unbounded memory: this is unavoidable because their output would necessarily be ultimately periodic otherwise, which is not suitable in general. To avoid specifying the size of memory cells or the complexity of arithmetic computations (e.g., incrementing the integer *size* in Algorithm 1), we consider a different model called *pointer machines* [24] which only allows arithmetic on a bounded domain. We use this model for our enumeration algorithms (but not, e.g., our other complexity results such as PTIME bounds).

Intuitively, a pointer machine works with *records* consisting of a constant number of labeled *fields* holding either *data values* (in our case of constant size, i.e., constantly many possible values) or *pointers* (whose representation is not specified). The machine has *memory* consisting of a finite but unbounded collection of records, a constant number of which are designated as *registers* and are always accessible. The machine can allocate records in constant time, retrieving a pointer to the memory location of the new record. We can access the fields of records, read or write pointers, dereference them, and test them for equality, all in constant time, but we cannot perform any other manipulation on pointers or other arithmetic operations. (We can, however, count in unary with a linked list, or perform arbitrary operations on the constant-sized data values.) See the full version [5] for details.

Algorithm 1 Push-pop enumeration algorithm for the language $a^*b^* + b^*a^*$ from Example 2.3.

```
// The first edit script is empty, corresponding to the empty word.
Output;
int size = 0;
while true do
    size++;
    pushR(b) ; Output;
    for int j = 0; j < size − 1; j++ do
        pushR(b) ; popL() ; Output;
    end
    for int j = 0; j < size; j++ do
        pushR(a) ; popL() ; Output;
    end
end
```

▶ **Example 2.4.** Continuing Example 2.3, Algorithm 1 can easily be adapted to a pointer-machine algorithm that 2-enumerates L, maintaining the word in a double-ended queue (deque) and keeping pointers to the first and last positions in order to know when to stop the **for** loops. Deques can indeed be simulated in this machine model, e.g., with linked lists.

3 Interchangeability partition and orderability lower bound

In this section, we start the proof of our main result, Result 1. Let A be the DFA and let Q be its set of states. The result is trivial if the language $\mathrm{L}(A)$ is finite, as we can always enumerate it naively with distance $O(|A|)$ and some arbitrary delay bound, so in the rest of the proof we assume that $\mathrm{L}(A)$ is infinite.

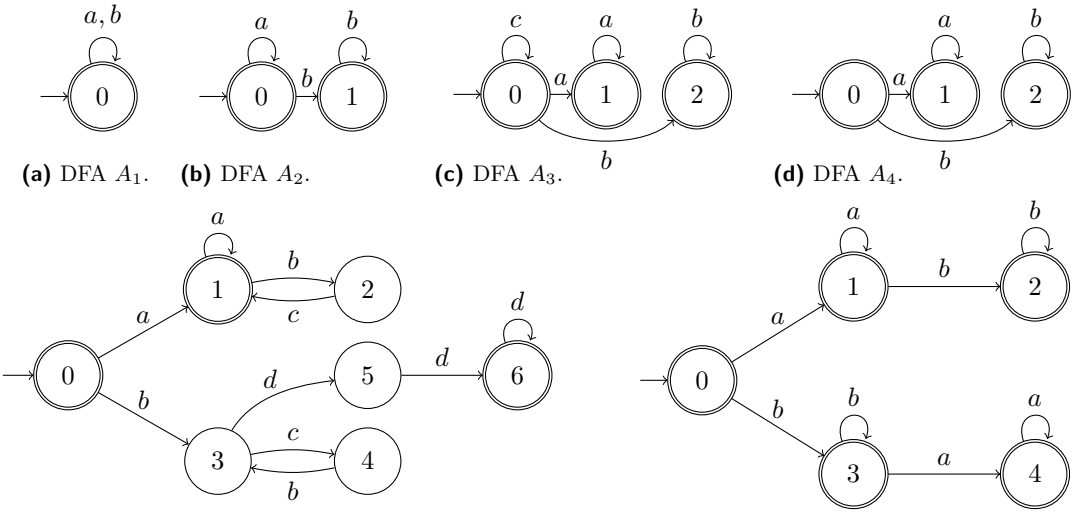
We will first define a notion of *interchangeability* on DFAs by introducing the notions of *connectivity* and *compatibility* on DFA states (this notion will be used in the next section to characterize orderability). We then partition $\mathrm{L}(A)$ into languages $\mathrm{L}(A_1) \sqcup \cdots \sqcup \mathrm{L}(A_t)$ following a so-called *interchangeability partition*, with each A_i having this interchangeability property. Last, we show in the section our lower bound establishing that t is optimal.

Interchangeability. To define our notion of interchangeability, we first define the *loopable* states of the DFA as those that are part of a non-empty cycle (possibly a self-loop):

▶ **Definition 3.1.** *For a state $q \in Q$, we let A_q be the DFA obtained from A by setting q as the only initial and final state. We call q* loopable *if $\mathrm{L}(A_q) \neq \{\epsilon\}$, and* non-loopable *otherwise.*

We then define the *interchangeability* relation on loopable states as the transitive closure of the union of two relations, called *connectivity* and *compatibility*:

▶ **Definition 3.2.** *We say that two loopable states q and q' are* connected *if there is a directed path from q to q', or from q' to q. We say that two loopable states q, q' are* compatible *if $\mathrm{L}(A_q) \cap \mathrm{L}(A_{q'}) \neq \{\epsilon\}$. These two relations are symmetric and reflexive on loopable states. We then say that two loopable states q and q' are* interchangeable *if they are in the transitive closure of the union of the connectivity and compatibility relations. In other words, q and q'*



(a) DFA A_1. **(b)** DFA A_2. **(c)** DFA A_3. **(d)** DFA A_4.

(e) DFA A_5. **(f)** DFA A_6.

Figure 1 Example DFAs from Example 3.4.

are interchangeable if there is a sequence $q = q_0, \ldots, q_n = q'$ of loopable states such that for any $0 \leqslant i < n$, the states q_i and q_{i+1} are either connected or compatible. Interchangeability is then an equivalence relation over loopable states.

Note that if two loopable states q, q' are in the same strongly connected component (SCC) of A then they are connected, hence interchangeable. Thus, we can equivalently see the interchangeability relation at the level of SCCs (excluding those that do not contain a loopable state, i.e., excluding the trivial SCCs containing only one state having no self-loop).

▶ **Definition 3.3.** *We call* classes of interchangeable states*, or simply* classes*, the equivalence classes of the interchangeability relation. Recall that, as* $\mathrm{L}(A)$ *is infinite, there is at least one class. We say that the DFA A is* interchangeable *if the partition has only one class, in other words, if all loopable states of A are interchangeable.*

▶ **Example 3.4.** The DFA A_1 shown in Figure 1a for the language $(a + b)^*$ has only one loopable state, so A_1 is interchangeable.

The DFA A_2 shown in Figure 1b for the language a^*b^* has two loopable states 0 and 1 which are connected, hence interchangeable. Thus, A_2 is interchangeable.

The DFA A_3 shown in Figure 1c for the language $c^*(a^* + b^*)$ has three loopable states: 0, 1 and 2. The states 0 and 1 are connected, and 0 and 2 are also connected, so all loopable states are interchangeable and A_3 is interchangeable.

The DFA A_4 shown in Figure 1d for the language $a^* + b^*$ has two loopable states 1 and 2 which are neither connected nor compatible. So A_4 is not interchangeable.

The DFA A_5 shown in Figure 1e for the language $a(a + bc)^* + b(cb)^*ddd^*$ mentioned in the introduction has five loopable states: 1, 2, 3, 4, and 6. Then 1 and 2 are connected, 3 and 4 are connected, 3 and 6 are connected, and 1 and 4 are compatible (with the word bc). Hence, all loopable states are interchangeable and A_5 is interchangeable.

The DFA A_6 shown in Figure 1f for the language $a^*b^* + b^*a^*$ from Example 2.3 has four loopable states: 1, 2, 3, and 4. Then 1 and 2 are connected, 3 and 4 are connected, and (for instance) 1 and 4 are compatible (with the word a). Hence all loopable states are interchangeable and A_6 is interchangeable.

Interchangeability partition. We now partition $\mathrm{L}(A)$ using interchangeable DFAs:

▶ **Definition 3.5.** *An* interchangeability partition *of A is a sequence $A_1, \ldots, A_t$ of DFAs such that $\mathrm{L}(A)$ is the disjoint union of the $\mathrm{L}(A_i)$ and every A_i is interchangeable. Its* cardinality *is the number t of DFAs.*

Let us show how to compute an interchangeability partition whose cardinality is the number of classes. We will later show that this cardinality is optimal. Here is the statement:

▶ **Proposition 3.6.** *We can compute in polynomial time in A an interchangeability partition $A_1, \ldots, A_t$ of A, with $t \leqslant |A|$ the number of classes of interchangeable states.*

Intuitively, the partition is defined following the classes of A. Indeed, considering any word $w \in \mathrm{L}(A)$ and its accepting run ρ in A, for any loopable state q and q' traversed in ρ, the word w witnesses that q and q' are connected, hence interchangeable. Thus, we would like to partition the words of $\mathrm{L}(A)$ based on the common class of the loopable states traversed in their accepting run. The only subtlety is that $\mathrm{L}(A)$ may also contain words whose accepting run does not traverse any loopable state, called *non-loopable words*. For instance, ϵ is a non-loopable word of $\mathrm{L}(A_5)$ for A_5 given in Figure 1e. Let us formally define the non-loopable words, and our partition of the loopable words based on the interchangeability classes:

▶ **Definition 3.7.** *A word $w = a_1 \cdots a_n$ of $\mathrm{L}(A)$ is* loopable *if, considering its accepting run $q_0, \ldots, q_n$ with q_0 the initial state and $q_i = \delta(q_{i-1}, a_i)$ for $1 \leqslant i \leqslant n$, one of the q_i is loopable. Otherwise, w is* non-loopable*. We write $\mathrm{NL}(A)$ the set of the non-loopable words of $\mathrm{L}(A)$.*

Letting $\mathcal{C}$ be a class of interchangeable states, we write $\mathrm{L}(A, \mathcal{C})$ the set of (loopable) words of $\mathrm{L}(A)$ whose accepting run traverses a state of $\mathcal{C}$.

We then have the following, with finiteness of $\mathrm{NL}(A)$ shown by the pigeonhole principle:

▷ Claim 3.8. The language $\mathrm{L}(A)$ can be partitioned as $\mathrm{NL}(A)$ and $\mathrm{L}(A, \mathcal{C}_1), \ldots, \mathrm{L}(A, \mathcal{C}_t)$ over the classes $\mathcal{C}_1, \ldots, \mathcal{C}_t$ of interchangeable states, and further $\mathrm{NL}(A)$ is finite.

We now construct an interchangeability partition of A of the right cardinality by defining one DFA A_i for each class of interchangeable states, where we simply remove the loopable states of the other classes. These DFAs are interchangeable by construction. We modify the DFAs to ensure that the non-loopable words are only captured by A_1. This construction (explained in the full version [5]) is doable in PTIME, in particular the connectivity and compatibility relations can be computed in PTIME, testing compatibility by checking the nonemptiness of product automata. This establishes Proposition 3.6.

Lower bound. We have shown how to compute an interchangeability partition of a DFA A with cardinality the number t of classes. Let us now show that this value of t is optimal, in the sense that $\mathrm{L}(A)$ cannot be partitioned into less than t orderable (even non-regular) languages. This lower bound holds even when allowing Levenshtein edits. Formally:

▶ **Theorem 3.9.** *For any partition of the language $\mathrm{L}(A)$ as $\mathrm{L}(A) = L_1 \sqcup \cdots \sqcup L_{t'}$ if for each $1 \leqslant i \leqslant t'$ the language L_i is orderable for the Levenshtein distance, then we have $t' \geqslant t$ for t the number of classes of A.*

This establishes the negative part of Result 1. Incidentally, this lower bound can also be shown even if the unions are not disjoint, indeed even if we allow repetitions, provided that there is some constant bound on the number of repetitions of each word.

Theorem 3.9 can be shown from the following claim which establishes that sufficiently long words from different classes are arbitrarily far away for the Levenshtein distance:

▶ **Proposition 3.10.** *Letting $\mathcal{C}_1, \ldots, \mathcal{C}_t$ be the classes of A, for any distance $d \in \mathbb{N}$, there is a threshold $l \in \mathbb{N}$ such that for any two words $u \in \mathrm{L}(A, \mathcal{C}_i)$ and $v \in \mathrm{L}(A, \mathcal{C}_j)$ with $i \neq j$ and $|u| \geqslant l$ and $|v| \geqslant l$, we have $\delta_{\mathrm{Lev}}(u, v) > d$.*

This proposition implies Theorem 3.9 because, if we could partition $\mathrm{L}(A)$ into less than t orderable languages, then some ordering must include infinitely many words from two different classes $\mathrm{L}(A, \mathcal{C}_i)$ and $\mathrm{L}(A, \mathcal{C}_j)$, hence alternate infinitely often between the two. Fix the distance d, and consider a point when all words of L of length $\leqslant \max(l, \max_{w \in \mathrm{NL}(A)} |w|)$ have been enumerated, for l the threshold of the proposition: then it is no longer possible for any ordering to move from one class to another, yielding a contradiction. As for the proof of Proposition 3.10, we give a sketch below (the complete proofs are in the full version [5]):

Proof sketch. Given a sufficiently long word $u \in \mathrm{L}(A, \mathcal{C}_i)$, by the pigeonhole principle its run must contain a large number of loops over some state $q \in \mathcal{C}_i$. Assume that we can edit u into $v \in \mathrm{L}(A, \mathcal{C}_j)$ with d edit operations: this changes at most d of these loops. Now, considering the accepting run of v and using the pigeonhole principle again on the sequence of endpoints of contiguous unmodified loops, we deduce that some state q' occurs twice; then $q' \in \mathcal{C}_j$ by definition of $\mathrm{L}(A, \mathcal{C}_j)$. The label of the resulting loop on q' is then also the label of a loop on q, so q and q' are compatible, hence $\mathcal{C}_i = \mathcal{C}_j$. ◀

4 Orderability upper bound

We have shown in the previous section that we could find an interchangeability partition of any regular language $\mathrm{L}(A)$ into languages $\mathrm{L}(A_1), \ldots, \mathrm{L}(A_t)$ of interchangeable DFAs, for t the number of classes. We know by our lower bound (Theorem 3.9) that we cannot hope to order $\mathrm{L}(A)$ with less than t sequences. Thus, in this section, we focus on each interchangeable A_i separately, and show how to order $\mathrm{L}(A_i)$ as one sequence. Hence, we fix for this section a DFA A that is interchangeable, write k its number of states, and show that $\mathrm{L}(A)$ is orderable. We will in fact show that this is the case for the push-pop distance:

▶ **Theorem 4.1.** *For any interchangeable DFA A, the language $\mathrm{L}(A)$ is $48k^2$-orderable for the push-pop distance.*

We show this result in the rest of this section, and strengthen it in the next section to a bounded-delay algorithm. Before starting, we give an overview of the structure of the proof. The proof works by first introducing *d-connectivity* of a language (not to be confused with the connectivity relation on loopable automaton states). This weaker notion is necessary for d-orderability, but for finite languages we will show a kind of converse: d-connectivity implies $3d$-orderability. We will then show that $\mathrm{L}(A)$ is *stratum-connected*, i.e., the finite *strata* of words of $\mathrm{L}(A)$ in some length interval are each d-connected for some common d. Last, we will show show that this implies orderability, using the result on finite languages.

Connectivity implies orderability on finite languages. We now define *d-connectivity*:

▶ **Definition 4.2.** *A language L is* d-connected *if for every pair of words $u, v \in L$, there exists a d-sequence in L between u and v.*

Clearly d-connectivity is a necessary condition for d-orderability: indeed if $w_1, w_2, \ldots$ is a d-ordering of L, and $u = w_i$, $v = w_j$ are two words of L with $i \leqslant j$ (without loss of generality), then $w_i, w_{i+1}, \ldots, w_j$ is indeed a d-sequence in L between u and v. What is more, for finite languages, the converse holds, up to multiplying the distance by a constant factor:

▶ **Lemma 4.3.** *Let L be a finite language that is d-connected and $s \neq e$ be words of L. Then there exists a $3d$-ordering of L starting at s and ending at e.*

Proof sketch. We use the fact, independently proved by Sekanina and by Karaganis [22, 13], that the cube of every connected graph G has a Hamiltonian path between any pair of vertices (see also [17]). One algorithmic way to see this is by traversing a spanning tree of G and handling odd-depth and even-depth nodes in prefix and postfix fashion (see, e.g., [25]). Applying this to the graph G whose vertices are the words of L and where two words w, w' are connected by an edge when $\delta(w, w') \leqslant d$ yields the result. ◀

The constant 3 in this lemma is optimal, as follows from [20]; see the full version [5] for more details. Note that the result does not hold for infinite languages: $a^* + b^*$ is 1-connected (via ϵ) but not d-orderable for any d.

Stratum-connectivity. To show orderability for infinite languages, we will decompose them into *strata*, which simply contain the words in a certain length range. Formally:

▶ **Definition 4.4.** *Let L be a language, let $\ell > 0$ be an integer, and let $i > 0$. The* i-th stratum of width ℓ *(or* ℓ-stratum*) of L, written* $\mathrm{strat}_\ell(L, i)$*, is $L^{<i\ell} \backslash L^{<(i-1)\ell}$.*

We will show that, for the language $\mathrm{L}(A)$ of our interchangeable DFA A, we can pick ℓ and d such that every ℓ-stratum of $\mathrm{L}(A)$ is d-connected, i.e., $\mathrm{L}(A)$ is (ℓ, d)-*stratum-connected*:

▶ **Definition 4.5.** *Let L be a regular language and fix $\ell, d > 0$. We say that L is* (ℓ, d)-stratum-connected *if every ℓ-stratum* $\mathrm{strat}_\ell(L, i)$ *is d-connected.*

Note that our example language $a^* + b^*$, while 1-connected, is not (ℓ, d)-stratum-connected for any ℓ, d, because any i-th ℓ-stratum for $i > d$ is not d-connected. We easily show that stratum-connectivity implies orderability:

▶ **Lemma 4.6.** *Let L be an infinite language recognized by a DFA with k' states, and assume that L is (ℓ, d)-stratum-connected for some $\ell \geqslant 2k'$ and some $d \geqslant 3k'$. Then L is $3d$-orderable.*

Proof sketch. We show by pumping that we can move across contiguous strata. Thus, we combine orderings on each stratum obtained by Lemma 4.3 with well-chosen endpoints. ◀

We can then show using several pumping and de-pumping arguments that the language of our interchangeable DFA A is (ℓ, d)-stratum-connected for $\ell := 8k^2$ and $d := 16k^2$.

▶ **Proposition 4.7.** *The language $\mathrm{L}(A)$ is $(8k^2, 16k^2)$-stratum-connected.*

Proof sketch. As there are only a finite number of non-loopable words, we focus on loopable words. Consider a stratum S and two loopable words u and v of S. Their accepting runs involve loopable states, respectively q and q', that are interchangeable because A is. We first show that u is d-connected (in S) to a *normal form*: a repeated loop on q plus a prefix and suffix whose length is bounded, i.e., only depends on the language. We impose this in two steps: first we move the last occurrence of q in u near the end of the word by pumping at the left end and de-pumping at the right end, second we pump the loop on q at the right end while de-pumping the left end. This can be done while remaining in the stratum S. We obtain similarly a normal form consisting of a repeated loop on q' with bounded-length prefix and suffix that is d-connected to v in S.

Then we do an induction on the number of connectivity and compatibility relations needed to witness that q and q' are interchangeable. If $q = q'$, we conclude using the normal forms of u and v. If q is connected to q', we impose the normal form on u, then we modify

it to a word whose accepting run also visits q', and we apply the previous case. If q is compatible with q', we conclude using the normal form with some loop label z in $A_q \cap A_{q'}$ (of length $\leqslant k^2$) that witnesses their compatibility. The induction case is then easy. ◀

From this, we deduce with Lemma 4.6 that $\mathrm{L}(A)$ is $48k^2$-orderable, so Theorem 4.1 holds. Note that the construction ensures that the words are ordered stratum after stratum, so "almost" by increasing length: in the ordering that we obtain, after producing some word w, we will never produce words of length less than $|w| - \ell$.

5 Bounded-delay enumeration

In this section, we show how the orderability result of the previous section yields a bounded-delay algorithm. We use the pointer-machine model from Section 2, which we modify for convenience to allow data values and the number of fields of records to be exponential in the automaton (but fixed throughout the enumeration, and independent on the size of words): see the full version [5] for more explanations. We show:

▶ **Theorem 5.1.** *There is an algorithm which, given an interchangeable DFA A with k states, enumerates the language $\mathrm{L}(A)$ with push-pop distance bound $48k^2$ and exponential delay in $|A|$.*

Let us accordingly fix the interchangeable DFA A with k states. Following Proposition 4.7, we let $d := 16k^2$ and $\ell := 8k^2$.

Overall amortized scheme. The algorithm will run two processes in parallel: the first process simply enumerates a previously prepared sequence of edit scripts that gives a $3d$-ordering of some stratum, while the second process computes the sequences for subsequent strata (and of course imposing that the endpoints of the sequences for contiguous strata are sufficiently close). We initialize this by computing in an arbitrary way a $3d$-ordering for the first stratum.

The challenging part is to prepare efficiently the sequences for all strata, and in particular to build a data structure that represents the strata. We will require of our algorithm that it processes each stratum in *amortized linear time* in its size. Formally, letting $N_j := |\mathrm{strat}_\ell(L, j)|$ be the number of words of the j-th stratum for all $j \geqslant 1$, there is a value $C \in \mathbb{N}$ that is exponential in $|A|$ such that, after having run for $C \sum_{j=1}^{i} N_j$ steps, the algorithm is done processing the i-th stratum. Note that this is weaker than processing each separate stratum in linear time: the algorithm can go faster to process some strata and spend this spared time later so that some later strata are processed arbitrarily slowly relative to their size.

If we can achieve amortized linear time, then the overall algorithm runs with bounded delay. To see why, notice that the prepared sequence for the i-th stratum has length at least its size N_i, and we can show that the size N_{i+1} of the next stratum is within a factor of N_i that only depends on L (this actually holds for any infinite regular language and does not use interchangeability):

▶ **Lemma 5.2.** *Letting $C_A := (k+1)|\Sigma|^{\ell+k+1}$, for all $i \geqslant 1$ we have $N_i/C_A \leqslant N_{i+1} \leqslant C_A N_i$.*

Proof. Each word in the $(i+1)$-th stratum of L can be transformed into a word in the i-th stratum as follows: letting k be the number of DFA states, first remove a prefix of length at most $\ell + k$ to get a word (not necessarily in L) of length $i\ell - k - 1$, and then add back a prefix corresponding to some path of length $\leqslant k$ from the initial state to get a word in the

i-th stratum of L as desired. Now, for any word w of the i-th stratum, the number of words of the $(i+1)$-th stratum that lead to w in this way is bounded by C_A, by considering the reverse of this rewriting, i.e., all possible ways to rewrite w by removing a prefix of length at most k and then adding a prefix of length at most $\ell + k$. A simple union bound gives $N_{i+1} \leqslant C_A N_i$. Now, a similar argument in the other direction gives $N_i / C_A \leqslant N_{i+1}$. ◀

Thanks to this lemma, it suffices to argue that we can process the strata in amortized linear time, preparing $3d$-orderings for each stratum: enumerating these orderings in parallel with the first process thus guarantees (non-amortized) bounded delay.

Preparing the enumeration sequence. We now explain in more detail the working of the amortized linear time algorithm. The algorithm consists of two components. The first component runs in amortized linear time over the successive strata, and prepares a sequence $\Gamma_1, \Gamma_2, \ldots$ of concise graph representations of each stratum, called *stratum graphs*; for each $i \geqslant 1$, after $C \sum_{j=1}^{i} N_j$ computation steps, it has finished preparing the i-th stratum graph Γ_i in the sequence. The second component will run as soon as some stratum graph Γ_i is finished: it reads the graph Γ_i and computes a $3d$-ordering for $\mathrm{strat}_\ell(L, i)$ in (non-amortized) linear-time, using Lemma 4.3. Let us formalize the notion of a stratum graph:

▶ **Definition 5.3.** *Let Δ be the set of all push-pop edit scripts of length at most d; note that $|\Delta| \leqslant (2|\Sigma| + 2)^{d+1}$, and this bound depends on the alphabet and on d. For $i \geqslant 1$, the* i-th stratum graph *is the edge-labeled directed graph $\Gamma_i = (V_i, \eta_i)$ where the nodes $V_i = \{v_w \mid w \in \mathrm{strat}_\ell(L, i)\}$ correspond to words of the i-th stratum, and the directed (labeled) edges are given by the function $\eta_i \colon V_i \times \Delta \to V_i \cup \{\bot\}$ and describe the possible scripts: for each $v_w \in V_i$ and each $s \in \Delta$, if the script s is applicable to w and the resulting word w' is in $\mathrm{strat}_\ell(L, i)$ then $\eta(v_w, s) = v_{w'}$, otherwise $\eta(v_w, s) = \bot$.*

In our machine model, each node v_w of Γ_i is a record with $|\Delta|$ pointers, i.e., we do not store the word w. Hence, Γ_i has linear size in N_i.

A stratum graph sequence *is an infinite sequence $(\Gamma_1, v_{s_1}, v_{e_1}), (\Gamma_2, v_{s_2}, v_{e_2}), \ldots$ consisting of the successive stratum graphs together with couples of nodes of these graphs such that, for all $i \geqslant 1$, s_i and e_i are distinct words of the i-th stratum, and we have $\delta_{\mathrm{pp}}(e_i, s_{i+1}) \leqslant d$.*

We can now present the second component of our algorithm. Note that the algorithm runs on the in-memory representations of the stratum graphs, in which, e.g., the subscripts are not stored.

▶ **Proposition 5.4.** *For $i \geqslant 1$, given the stratum graph Γ_i and starting and ending nodes $v_{s_i} \neq v_{e_i}$ of Γ_i, we can compute in time $O(|\Gamma_i|)$ a sequence of edit scripts $\sigma_1, \ldots, \sigma_{N_i - 1}$ such that, letting $s_i = u_1, \ldots, u_{N_i}$ be the successive results of applying $\sigma_1, \ldots, \sigma_{N_i - 1}$ starting with s_i, then $u_1, \ldots, u_{N_i}$ is a $3d$-ordering of $\mathrm{strat}_\ell(L, i)$ starting at s_i and ending at e_i.*

Proof sketch. We apply the spanning tree enumeration technique from Lemma 4.3 (in $O(|\Gamma_i|)$) on Γ_i, starting with v_{s_i} and ending with v_{e_i}, and read the scripts from the edge labels. ◀

In the rest of the section we present the first component of our enumeration algorithm:

▶ **Proposition 5.5.** *There is an integer $C \in \mathbb{N}$ exponential in $|A|$ such that we can produce a stratum graph sequence $(\Gamma_1, v_{s_1}, v_{e_1}), (\Gamma_2, v_{s_2}, v_{e_2}), \ldots$ for L in amortized linear time, i.e., for each $i \geqslant 1$, after having run $C \sum_{j=1}^{i} N_j$ steps, the algorithm is done preparing $(\Gamma_i, v_{s_i}, v_{e_i})$.*

Word DAGs. The algorithm to prove Proposition 5.5 will grow a large structure in memory, common to all strata, from which we can easily compute the $(\Gamma_i, v_{s_i}, v_{e_i})$. We call this structure a *word DAG*. A word DAG is informally a representation of a collection of words, each of which has outgoing edges corresponding to the possible left and right push operations.

▶ **Definition 5.6.** *Let* $\Lambda := \{\texttt{pushR}(a) \mid a \in \Sigma\} \cup \{\text{pushL}(a) \mid a \in \Sigma\}$ *be the set of labels corresponding to the possible left and right push operations. A* pre-word DAG *is an edge-labeled directed acyclic graph (DAG)* $G = (V, \eta, \mathsf{root})$ *where* V *is a set of anonymous vertices,* $\mathsf{root} \in V$ *is the* root*, and* $\eta \colon V \times \Lambda \to V \cup \{\bot\}$ *represents the labeled edges in the following way: for each node* $v \in V$ *and label* $s \in \Lambda$*, if* $\eta(v, s) \neq \bot$ *then* v *has one successor* $\eta(v, s)$ *for label* s*, and none otherwise. We impose:*

- *The root has no incoming edges. All other nodes have exactly two incoming edges: one labeled* $\texttt{pushR}(a)$ *for some* $a \in \Sigma$*, the other labeled* $\text{pushL}(b)$ *for some* $b \in \Sigma$*. Each node stores two pointers leading to these two parents, which may be identical.*
- *All nodes can be reached from the root via at least one directed path.*
- *The root has one outgoing edge for each child, i.e., for all* $s \in \Lambda$*, we have* $\eta(\mathsf{root}, s) \neq \bot$*.*

The word *represented by a directed path from the root to a node* n *is defined inductively:*

- *the word represented by the empty path is* ϵ*,*
- *the word represented by a path* $P, \texttt{pushR}(a)$ *is* wa *where* w *is the word represented by* P*,*
- *the word represented by a path* $P, \text{pushL}(a)$ *is* aw *where* w *is the word represented by* P*.*

The pre-word DAG G *is called a* word DAG *if for each node* n*, all paths from* root *to* n *represent the same word. This word is then called* the word represented by n*.*

Example pre-word DAGs and word DAGs are shown on Figures 2 and 3 in the appendix. In our machine model, each node is represented by a record; crucially, like for stratum graphs, the word that the node represents is not explicitly written.

Crucially, word DAGs do not us allow not to create two different nodes that represent the same word – these would be problematic since we have to enumerate without repetition.

▶ **Fact 5.7.** *There are no two different nodes in a word DAG that represent the same word.*

We can then show the following theorem, intuitively saying that we can discover all the words of the language by only visiting words that are not too far from it:

▶ **Proposition 5.8.** *We can build a word DAG* G *representing the words of* L *in amortized linear time: specifically, for some value* C *that is exponential in* $|A|$*, for all* i*, after* $C \times \sum_{j=1}^{i} N_j$ *computation steps, for each word* w *of* Σ^* *whose push-pop distance to a word of* $\bigcup_{j=1}^{i} \text{strat}_\ell(L, j)$ *is no greater than* d*, then* G *contains a node that represents* w*. Moreover, there is also a value* D *exponential in* $|A|$ *such that any node that is eventually created in the word DAG represents a word that is at push-pop distance at most* D *from a word of* L*.*

Proof sketch. We progressively add nodes to a word DAG while efficiently preserving its properties, and thus avoid creating duplicate nodes. By labeling each node with the element of $Q \cup \{\bot\}$ achieved by the word represented by that node, and also by the distance to the closest known word of L, we can restrict the exploration to nodes corresponding to words that are close to the words of L, which ensures the amortized linear time bound. ◀

This is enough to prove Proposition 5.5: we run the algorithm of Proposition 5.8 and, whenever it has built a stratum, construct the stratum graph Γ_i and nodes v_{s_i}, v_{e_i} by exploring the relevant nodes of the word DAG. Full proofs are deferred to the full version [5].

6 Extensions

Complexity of determining the optimal distance. We have shown in Result 1 that, given a DFA A, we can compute in PTIME a minimal cardinality partition of $\mathrm{L}(A)$ into languages that are each d-orderable, for $d = 48|A|^2$. However, we may achieve a smaller distance d if we increase the cardinality, e.g., $a^* + bbba^*$ is $(1, 3)$-partition-orderable and not $(1, d)$-partition-orderable for $d < 3$, but is $(2, 1)$-partition-orderable. This tradeoff between t and d seems difficult to characterize, and in fact it is NP-hard to determine if an input DFA is (t, d)-partition-orderable, already for fixed t, d and for finite languages. Indeed, there is a simple reduction pointed out in [18] from the Hamiltonian path problem on grid graphs [12]:

▶ **Proposition 6.1** ([18]). *For any fixed $t, d \geqslant 1$, it is NP-complete, given a DFA A with $\mathrm{L}(A)$ finite, to decide if $\mathrm{L}(A)$ is (t, d)-partition-orderable (with the push-pop or Levenshtein distance).*

Push-pop-right distance. A natural restriction of the push-pop distance would be to only allow editions at the right endpoint of the word, called the *push-pop-right* distance. A d-ordering for this distance witnesses that the words of the language can be produced successively while being stored in a stack, each word being produced after at most d edits.

Unlike the push-pop distance, one can show that some regular languages are not even partition-orderable for this distance, e.g., a^*b^* is not (t, d)-partition-orderable with any $t, d \in \mathbb{N}$. The enumerable regular languages for this distance in fact correspond to the well-known notion of *slender languages*. Recall that a regular language L is *slender* [19] if there is a bound $C \in \mathbb{N}$ such that, for each $n \geqslant 0$, we have $|L \cap \Sigma^n| \leqslant C$. It is known [19] that we can test in PTIME if an input DFA represents a slender language. Rephrasing Result 2 from the introduction, we can show that a regular language is enumerable for the push-pop-right distance if and only if it is slender; further, if it is, then we can tractably compute the optimal number t of sequences (by counting the number of different paths to loops in the automaton), and we can do the enumeration with bounded delay:

▶ **Theorem 6.2.** *Given a DFA A, the language $\mathrm{L}(A)$ is (t, d)-partition-orderable for the push-pop-right distance for some $t, d \in \mathbb{N}$ if and only if $\mathrm{L}(A)$ is slender. Further, if $\mathrm{L}(A)$ is slender, we can compute in PTIME the smallest t such that $\mathrm{L}(A)$ is (t, d)-partition-orderable for some $d \in \mathbb{N}$ for the push-pop-right distance.*

In addition, there is an algorithm which, given a DFA A for which $\mathrm{L}(A)$ is slender and $t = 1$, enumerates the language $\mathrm{L}(A)$ with push-pop-right distance bound $2k$ and linear delay in $|A|$. Further, the sequence of edit scripts produced by the algorithm is ultimately periodic.

Of course, our results for the push-pop-right distance extend to the push-pop-left distance up to reversing the language, except for the complexity results because the reversal of the input DFA is generally no longer deterministic.

7 Conclusion and future work

We have introduced the problem of ordering languages as sequences while bounding the maximal distance between successive words, and of enumerating these sequences with small edit scripts to achieve bounded delay. Our main result is a PTIME characterization of the regular languages that can be ordered in this sense for the push-pop distance (or equivalently the Levenshtein distance), for any specific number of sequences; and a bounded-delay enumeration algorithm for the orderable regular languages. Our characterization uses the

number of classes of interchangeable states of a DFA A for the language, which, as our results imply, is an intrinsic parameter of $\mathrm{L}(A)$, shared by all (trimmed) DFAs recognizing the same language. We do not know if this parameter can be of independent interest.

Our work opens several questions for future research. The questions of orderability and enumerability can be studied for more general languages (e.g., context-free languages), other distances (in particular substitutions plus push-right operations, corresponding to the Hamming distance on a right-infinite tape), or other enumeration models (e.g., reusing factors of previous words). We also do not know the computational complexity, e.g., of optimizing the distance while allowing any finite number of threads, in particular for slender languages. Another complexity question is to understand if the bounded delay of our enumeration algorithm could be made polynomial in the input DFA rather than exponential, or what delay can be achieved if the input automaton is nondeterministic.

References

1 Margareta Ackerman and Erkki Mäkinen. Three new algorithms for regular language enumeration. In *ICCC*, 2009. URL: https://maya-ackerman.com/wp-content/uploads/2018/09/ThreeNewAlgorithmsForRegularLanEnum.pdf.

2 Margareta Ackerman and Jeffrey Shallit. Efficient enumeration of words in regular languages. *Theoretical Computer Science*, 410(37), 2009. URL: https://maya-ackerman.com/wp-content/uploads/2018/09/Enumeration_AckermanShallit_TCS.pdf.

3 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, 2019. arXiv:1807.09320.

4 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *PODS*, 2019. arXiv:1812.09519.

5 Antoine Amarilli and Mikaël Monet. Enumerating regular languages with bounded delay, 2023. Full version with proofs. arXiv:2209.14878.

6 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, 2006.

7 Guillaume Bagan, Arnaud Durand, and Étienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, 2007. URL: https://grandjean.users.greyc.fr/Recherche/PublisGrandjean/EnumAcyclicCSL07.pdf.

8 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2), 2015. URL: https://pdfs.semanticscholar.org/8df0/ad1c6aa0df93e58071b8afe3371a16a3182f.pdf, doi:10.1145/2699442.

9 Rainer Feldmann and Peter Mysliwietz. The shuffle exchange network has a Hamiltonian path. *Mathematical systems theory*, 29(5), 1996.

10 Lukas Fleischer and Jeffrey Shallit. Recognizing lexicographically smallest words and computing successors in regular languages. *International Journal of Foundations of Computer Science*, 32(06), 2021.

11 Fernando Florenzano, Cristian Riveros, Martin Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, 2018. arXiv:1803.05277.

12 Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982. URL: http://www.cs.technion.ac.il/~itai/publications/Algorithms/Hamilton-paths.pdf.

13 Jerome J. Karaganis. On the cube of a graph. *Canadian Mathematical Bulletin*, 11(2), 1968.

14 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *TOCL*, 14(4), 2013. URL: https://hal.archives-ouvertes.fr/docs/00/90/70/85/PDF/cdlin-survey.pdf.

15 Erkki Mäkinen. On lexicographic enumeration of regular and context-free languages. *Acta Cybernetica*, 13(1):55–61, 1997. URL: `http://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3479/3464`.

16 Torsten Mütze. Proof of the middle levels conjecture. *Proceedings of the London Mathematical Society*, 112(4):677–713, 2016. `arXiv:1404.4442`.

17 Torsten Mütze. Combinatorial Gray codes—An updated survey, 2022. `arXiv:2202.01280`.

18 pcpthm (https://cstheory.stackexchange.com/users/65605/pcpthm). Enumerating finite set of words with Hamming distance 1. Theoretical Computer Science Stack Exchange. Version: 2022-07-02. URL: `https://cstheory.stackexchange.com/q/51653`.

19 Jean-Éric Pin. Mathematical foundations of automata theory, 2019. URL: `https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf`.

20 Jakub Radoszewski and Wojciech Rytter. Hamiltonian paths in the square of a tree. In *ISAAC*, 2011. URL: `https://www.mimuw.edu.pl/~rytter/MYPAPERS/isaac2011_rytter.pdf`.

21 Frank Ruskey. Combinatorial generation. Preliminary working draft, 2003. URL: `https://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf`.

22 Milan Sekanina. On an ordering of the set of vertices of a connected graph. *Publ. Fac. Sci. Univ. Brno*, 412, 1960.

23 Yann Strozecki et al. Enumeration complexity. *Bulletin of EATCS*, 3(129), 2019. URL: `http://eatcs.org/beatcs/index.php/beatcs/article/view/596`.

24 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of computer and system sciences*, 18(2):110–127, 1979.

25 Takeaki Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical report, National Institute of Informatics, 2003. URL: `https://www.nii.ac.jp/TechReports/public_html/03-004E.pdf`.

26 Kunihiro Wasa. Enumeration of enumeration algorithms. *CoRR*, 2016. `arXiv:1605.05102`.

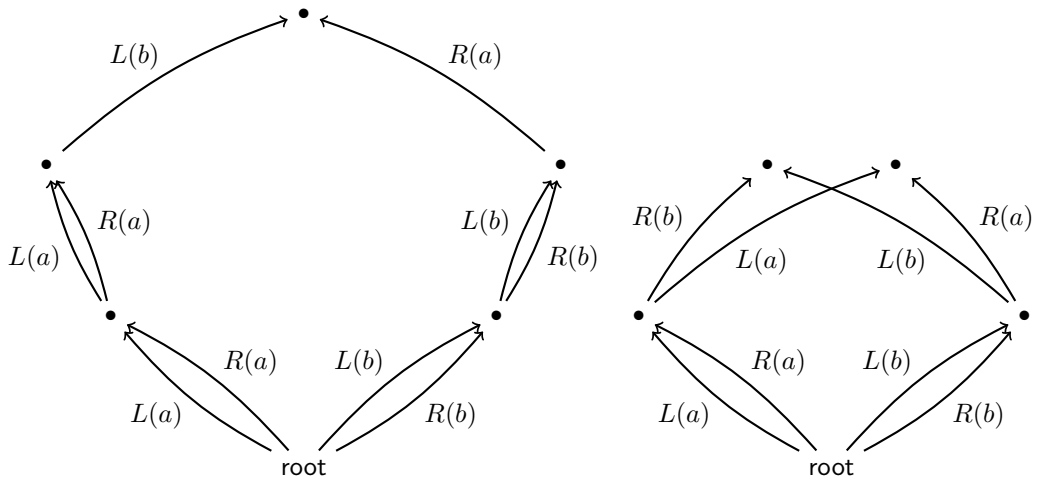


Figure 2 Two example pre-word DAGs which are not word DAGs. The labels pushL and `pushR` are abbreviated for legibility. In the left pre-word DAG, the four paths to the top node that start to the left of the root all represent the word *baa*, whereas the four paths to that same node that start to the right of the root all represent the word *bba*. In the right pre-word DAG, the left topmost node represents *ab* and *bb* and the right topmost node represents *aa* and *ba*. The criteria of word DAGs, and our construction to enlarge them, are designed to prevent these problems.

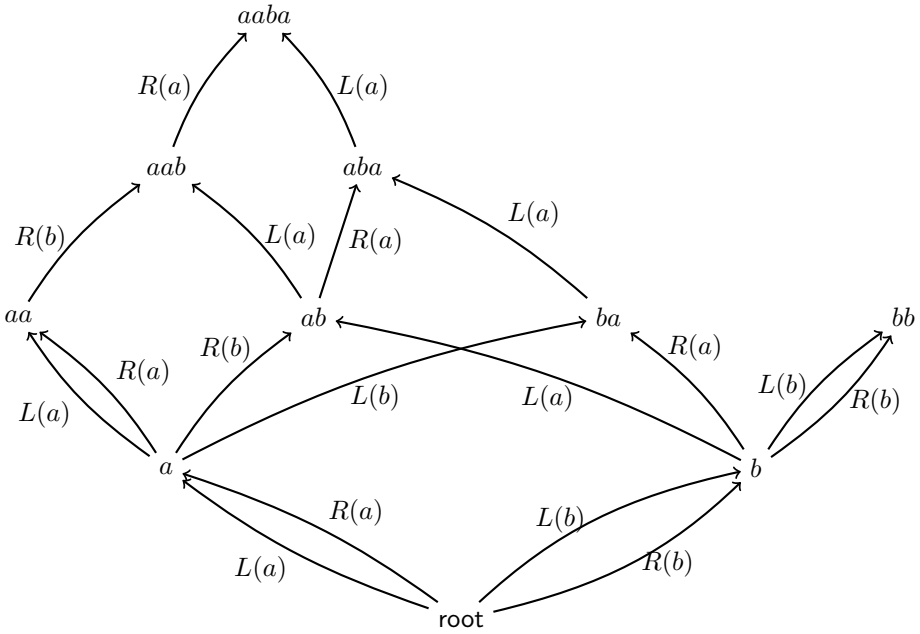


Figure 3 An example word DAG. We annotate the nodes with the word that they represent, even though in the memory representation the nodes are anonymous and the words are not represented. The labels pushL and `pushR` are abbreviated for legibility.

Regular Separability in Büchi VASS

Pascal Baumann
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Roland Meyer
TU Braunschweig, Germany

Georg Zetzsche
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Abstract

We study the (ω-)regular separability problem for Büchi VASS languages: Given two Büchi VASS with languages L_1 and L_2, check whether there is a regular language that fully contains L_1 while remaining disjoint from L_2. We show that the problem is decidable in general and PSPACE-complete in the 1-dimensional case, assuming succinct counter updates. The results rely on several arguments. We characterize the set of all regular languages disjoint from L_2. Based on this, we derive a (sound and complete) notion of inseparability witnesses, non-regular subsets of L_1. Finally, we show how to symbolically represent inseparability witnesses and how to check their existence.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Separability problem, Vector addition systems, Infinite words, Decidability

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.9

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2301.11242

Funding Funded by the European Union (ERC, FINABIS, 101077902). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. The second author was supported by the DFG project EDS@SYN: Effective Denotational Semantics for Synthesis.

1 Introduction

The separability problem asks, given languages L_1 and L_2, whether there exists a language R that *separates* L_1 and L_2, meaning $L_1 \subseteq R$ and $R \cap L_2 = \emptyset$. Here, R is constrained to be from a particular class $\mathcal{S}$ of admitted separators. Since safety verification of systems with concurrent components is usually phrased as an intersection problem for finite-word languages, and separators certify disjointness, deciding separability can be viewed as synthesizing safety certificates. Analogously, deciding separability for infinite-word languages is a way of certifying liveness. If $\mathcal{S}$ is the class of (ω-)regular languages, we speak of *regular separability*.

Separability problems have been studied intensively over the last few years. If the input languages are themselves regular and $\mathcal{S}$ is a subclass [36, 35, 34, 33, 37, 38, 30, 14], then separability generalizes the classical subclass membership problem. Moreover, separability for languages of infinite-state systems has received a significant amount of attention [16, 15, 13, 12, 9, 8, 11, 1, 44, 42, 10, 7]. Let us point out two prominent cases.

First, one of the main open problems in this line of research is whether regular separability is decidable for (reachability) languages of *vector addition systems with states* (VASS): A VASS consist of finitely many control states and a set of counters that can be *incremented* and *decremented*, but not tested for zero. Moreover, each transition is labeled by a word over the input alphabet. Here, a run is accepting if it reaches a final state with all counters being zero. While there have been several decidability results for subclasses of the VASS languages [16, 13, 12, 9, 8], the general case remains open. Second, a surprising result is that

© Pascal Baumann, Roland Meyer, and Georg Zetzsche;
licensed under Creative Commons License CC-BY 4.0

if K and L are coverability languages of *well-structured transition systems* (WSTS), then K and L are separable by a regular language if and only if they are disjoint [13]. As VASS are one example of WSTS, this result also applies to their coverability languages.

Regular separability in Büchi VASS. In this paper, we study the regular separability problem for Büchi VASS. These are VASS that accept languages of *infinite words*. A run is accepting if it visits some final state infinitely often. Since no condition is placed on the counter values, Büchi VASS languages are an infinite-word analogue of finite-word *coverability languages*, where acceptance is defined by the reached state (not the counters). The *regular separability problem* is to decide, given Büchi VASS $\mathcal{V}_1$ and $\mathcal{V}_2$, whether there exists an ω-regular language R such that $L(\mathcal{V}_1) \subseteq R$ and $L(\mathcal{V}_2) \cap R = \emptyset$.

Our main results are that (i) regular separability for Büchi VASS is decidable, and that (ii) for one-dimensional Büchi VASS (i.e. those with a single counter) the problem is PSPACE-complete. Here, we assume that the counter updates are encoded in binary.

Given that Büchi VASS accept using final states and their transition systems are WSTS, one may suspect that there is an analogue of the aforementioned result for WSTS: Namely, that two languages of Büchi VASS are separable by an ω-regular language if and only if they are disjoint. We show that this is not the case: There are Büchi VASS $\mathcal{V}_1$ and $\mathcal{V}_2$ such that $L(\mathcal{V}_1)$ and $L(\mathcal{V}_2)$ are disjoint, but not separable by an ω-regular language. In fact, we show an even larger disparity between these two problems for WSTS in the infinite-word case: We exhibit a natural class of WSTS for which intersection is decidable but regular separability is not. Thus, regular separability for Büchi VASS requires significantly new ideas and involves several phenomena that do not occur for finite-word languages of VASS.

New phenomena and key ingredients. We first observe that we can assume one input language to be fixed, namely an infinite-word version D_n of the Dyck language. Then, following the *basic separator* approach from [16], we identify a small class $\mathcal{B}$ of ω-regular languages such that L is separable from D_n if and only if L is included in a finite union of sets from $\mathcal{B}$. Here, a crucial insight is that a Büchi automaton can guarantee disjointness from D_n without knowing exactly when the letter balance crosses zero. Note that a negative letter balance is the exact condition for non-membership in D_n. In contrast, in the finite word case, there are always separating automata that can tell when zero is crossed [16]. This insight is also key to the example differentiating disjointness and separability in Büchi VASS, and to the undecidability proof for certain WSTS despite decidable disjointness.

We then develop a decomposition of Büchi VASS languages into *finitely many* pieces, which are induced by what we call *profiles*. Inspired by Büchi automata, the idea of a profile is to fix the set of transitions that can and have to be taken infinitely often in a run. Finding the right generalization to Büchi VASS, however, turned out to be non-trivial. Our formulation refers to edges in the Karp-Miller graph, augmented by constraints that guarantee the existence of an accepting run. The resulting decomposition has properties similar to the decomposition of VASS languages into run ideals [28], which has been useful for previous separability procedures [16, 11].

We associate to each profile a system of linear inequalities and show that separability holds if and only if each of these systems is feasible. While this yields decidability, checking feasibility is not sufficient to obtain a PSPACE-upper bound in the one-dimensional case. Instead, we use Farkas' Lemma to obtain a dual system of inequalities so that separability fails if and only if one dual system is satisfiable. A solution to a dual system yields a pattern in the Karp-Miller graph, called *inseparability flower*, which witnesses inseparability.

Compared to prior witnesses for deciding properties of VASS languages (e.g. regularity [17], language boundedness [6], and other properties [2]), inseparability flowers are quite unusual: they contain a non-linear condition, requiring one vector to be a scalar multiple of another.

For one-dimensional Büchi VASS, the condition degenerates into a linear one. This allows us to translate inseparability flowers into particular runs in a two-dimensional VASS subject to additional linear constraints. Using methods from [4], this yields a PSPACE procedure.

Related work. It was already shown in 1976 that regular separability is undecidable for context-free languages [41, 25]. Over the last decade, there has been intense interest in deciding regular separability for subclasses of finite-word VASS reachability languages: The problem is decidable for (i) reachability languages of one-dimensional VASS [12], (ii) coverability languages of VASS [13], (iii) reachability languages of Parikh automata [8], and (iv) commutative reachability languages of VASS [9]. Moreover, decidability still holds if one input language is an arbitrary VASS language and the other is as in (i)-(iii) [16]. As discussed above, for finite-word coverability languages of WSTS, regular separability is equivalent to disjointness [13]. Moreover, the aforementioned undecidability for context-free languages has been strengthened to visibly pushdown languages [27]. To our knowledge, for languages of infinite words, separability has only been studied for regular input languages [32, 24].

Our result makes use of Farkas' Lemma to demonstrate the absence of what can be understood as a linear ranking function (on letter balances). There are precursors to this. In liveness verification [39], Farkas' Lemma has been used to synthesize, in a complete way, linear ranking functions proving the termination of while programs over integer variables. In the context of separability for finite words, Farkas' Lemma was used to distinguish separable from non-separable instances [16], similar to our approach. The novelty here is the combination of Farkas' Lemma with the new notion of profiles needed to deal with infinite runs.

The languages of Büchi VASS have first been studied by Valk [43] and (in the deterministic case) Carstensen [5]. Some complexity results (such as EXPSPACE-complexity of the emptiness problem) were shown by Habermehl [23]. More recently, there have been several papers on the topological complexity of Büchi VASS languages (and restrictions) [21, 18, 22]. See the recent article by Finkel and Skrzypczak [22] for an overview.

2 Preliminaries

Dyck Language. We use an infinite-word version of the Dyck language over n pairs of matching letters $a_i, \bar{a}_i$. We denote the underlying alphabet by $\Sigma_n := \bigcup_{i=1}^{n} \{a_i, \bar{a}_i\}$. The *Dyck language* contains those infinite words where every occurrence of $\bar{a}_i$ has a matching occurrence of a_i to its left: $D_n := \{w \in \Sigma_n^\omega \mid \forall v \in \mathrm{prefix}(w) \colon \forall i \in [1,n] \colon \varphi_i(v) \geq 0\}$. Here, $\varphi_i : \Sigma_n^* \to \mathbb{Z}$ is the ith (*letter*) *balance* function that computes for a given word w the difference $|w|_{a_i} - |w|_{\bar{a}_i}$. We also use $\varphi(w)$ for the vector $(\varphi_1(w), \ldots, \varphi_n(w)) \in \mathbb{Z}^n$.

Büchi VASS and Automata. A *Büchi vector addition system with states (Büchi VASS)* of dimension $d \in \mathbb{N}$ over alphabet Σ is a tuple $\mathcal{V} = (Q, q_0, T, F)$ consisting of a finite set of states Q, an initial state $q_0 \in Q$, a set of final states $F \subseteq Q$, and a finite set of transitions $T \subseteq Q \times \Sigma^* \times \mathbb{Z}^d \times Q$. The size of the Büchi VASS is $|\mathcal{V}| := |Q| + 1 + |F| + \sum_{(q,w,\delta,q') \in T} |w| + \sum_{i=1}^{d} \max\{\log |\delta(i)|, 1\}$. If $d = 0$, we call $\mathcal{V}$ a *Büchi automaton*.

The semantics of the Büchi VASS is defined over *configurations*, which are elements of $Q \times \mathbb{N}^d$. We call the second component in a configuration the *counter valuation* and refer to the entry in dimension i as the *value of counter* i. The *initial configuration* is $(q_0, \mathbf{0})$. We lift

the transitions of the Büchi VASS to a relation over configurations $\to \subseteq Q \times \mathbb{N}^d \times \Sigma^* \times Q \times \mathbb{N}^d$ as follows: $(q, \boldsymbol{m}) \xrightarrow{w} (q', \boldsymbol{m}')$ if there is $(q, w, \delta, q') \in T$ so that $\boldsymbol{m}' = \boldsymbol{m} + \delta$. A *run* of the Büchi VASS is an infinite sequence of transitions of the form $(q_0, \mathbf{0}) \xrightarrow{w_1} (q_1, \boldsymbol{m}_1) \xrightarrow{w_2} \cdots$ Thus, the sequence starts in the initial configuration and makes sure the target of one transition is the source of the next. The run is *accepting* if it visits final states infinitely often, meaning there are infinitely many configurations $(q, \boldsymbol{m})$ with $q \in F$. The run is said to be *labeled* by the word $w = w_0 w_1 \cdots$ in Σ^ω. The *language* $L(\mathcal{V})$ of the Büchi VASS consists of all infinite words that label an accepting run. Note that we can always ensure that every accepting run has an infinite-word label, by tracking in the state whether a non-ε-transition has occurred since the last visit to a final state. An infinite-word language is (ω-)*regular*, if it is the language of a Büchi automaton. As we only consider infinite-word languages, we just call them languages.

Karp-Miller Graphs. We work with the Karp-Miller graph $\mathsf{KM}(\mathcal{V})$ associated with a Büchi VASS $\mathcal{V}$ [26]. Since we are interested in infinite runs, we define the Karp-Miller graph as a Büchi automaton. Its state set is a finite set of *extended configurations*, which are elements of $Q \times (\mathbb{N} \cup \{\omega\})^d$. The initial state is the initial configuration in the Büchi VASS. The final states are those extended configurations $(q, \boldsymbol{m})$ with $q \in F$. The transitions are labeled by T, so instead of letters they carry full Büchi VASS transitions. An entry ω in an extended configuration denotes the fact that a prefix of a run can be repeated to produce arbitrarily high counter values. More precisely, the Karp-Miller graph is constructed as follows. From an extended configuration $(q, \boldsymbol{m})$ we have a transition labeled by (q_1, a, δ, q_2), if $q = q_1$ and $\boldsymbol{m} + \delta$ remains non-negative. The latter addition is defined componentwise and assumes $\omega + k := \omega =: k + \omega$ for all $k \in \mathbb{Z}$. The result of taking the transition is the extended configuration $(q_2, \boldsymbol{m}_2)$, where $\boldsymbol{m}_2$ is constructed from $\boldsymbol{m} + \delta$ as follows. We raise to ω all counters i for which there is an earlier configuration $(q_2, \boldsymbol{m}_1)$ with $\boldsymbol{m}_1 \leq \boldsymbol{m} + \delta$ and $\boldsymbol{m}_1(i) < [\boldsymbol{m} + \delta](i)$, earlier meaning on some path from $(q_0, \mathbf{0})$ to $(q, \boldsymbol{m})$. If this is the case, the path from $(q_2, \boldsymbol{m}_1)$ to $(q_2, \boldsymbol{m} + \delta)$ can be repeated indefinitely to produce arbitrarily high values for counter i. We refer to the repetition of such a path in a run as *pumping*.

The Karp-Miller graph over-approximates the language of the Büchi VASS in the following sense. Every infinite sequence of transitions that leads to a run of the Büchi VASS is the labeling of an infinite run in the Karp-Miller graph. Moreover, if the run of the Büchi VASS is accepting, so is the run in the Karp-Miller graph. In the other direction, every finite transition sequence in the Karp-Miller graph represents a transition sequence in the Büchi VASS. The sequence in the Büchi VASS, however, may be longer to compensate negative effects on ω-entries by pumping.

3 Problem, Main Result, and Proof Outline

A language R is a *regular separator* for a pair of languages L_1, L_2, if R is regular, $L_1 \subseteq R$, and $R \cap L_2 = \emptyset$. We write $L_1 \,|\, L_2$ for the fact that a regular separator exists. We consider here languages of Büchi VASS, and formulate the *regular separability problem* as follows. Given Büchi VASS $\mathcal{V}_1$, $\mathcal{V}_2$, check whether $L(\mathcal{V}_1) \,|\, L(\mathcal{V}_2)$ holds. Our main result is the following.

▶ **Theorem 3.1.** *The regular separability problem for Büchi VASS is decidable.*

It should be noted that our procedure is non-primitive recursive, as it explicitly constructs the Karp-Miller graph of an input Büchi VASS, which can be of Ackermannian size [31, Theorem 2]. In the case of VASS coverability languages (and even for more general WSTS),

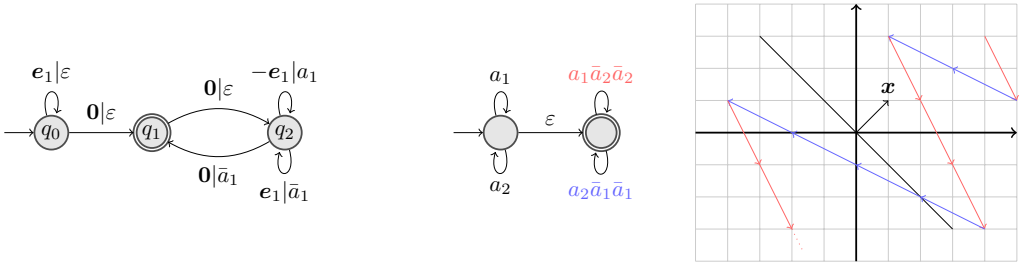


Figure 1 Left: A Büchi VASS accepting a language S with $S \cap D_1 = \emptyset$ but $S \not| D_1$. Here, $e_1 \in \mathbb{Z}$ is the one-dim. vector with entry 1. Right: A regular language that is not included in a finite union of languages $P_{i,k}$ and $S_{i,k}$, but that is included in $S_{\boldsymbol{x},k}$ for $\boldsymbol{x} = (1,1)$, $k = 1$. The horizontal and vertical dimensions denote the balance for a_1 resp. a_2.

it is known that regular separability is equivalent to disjointness [13]. Thus, for finite words, separability reduces to the much better understood problem of disjointness. For the infinite-word languages considered here, the situation is different.

▶ **Theorem 3.2.** *There are Büchi VASS languages L_1, L_2 with $L_1 \cap L_2 = \emptyset$ and $L_1 \not| L_2$. There are classes of WSTS where intersection is decidable but separability is not.*

For the second statement, we introduce the class of *weak Büchi reset VASS*, which are VASS with reset instructions, with the additional constraint that each run can only use resets a finite number of times.

For the first statement of Theorem 3.2, let us give an intuition. We choose $L_1 = L(\mathcal{V})$, where $\mathcal{V}$ is the Büchi VASS in Figure 1(left), and $L_2 = D_1$, the Dyck language. To show $L(\mathcal{V}) \not| D_1$, suppose there is a Büchi automaton $\mathcal{A}$ with n states such that $L(\mathcal{V}) \subseteq L(\mathcal{A})$ and $L(\mathcal{A}) \cap D_1 = \emptyset$. Then $\mathcal{A}$ has to accept $(a_1^n \bar{a}_1^{n+1})^\omega \in L(\mathcal{V})$. However, pumping yields that for some $m > n$ the word $(a_1^m \bar{a}_1^{n+1})^\omega \in D_1$ also has to be accepted by $\mathcal{A}$, contradiction. Moreover, to show $L(\mathcal{V}) \cap D_1 = \emptyset$ we observe that in accepting runs of $\mathcal{V}$, almost every visit (meaning: all but finitely many) to the final state drops the letter balance by 1. Therefore on any accepting run this balance eventually becomes negative, yielding a word outside of D_1.

In the remainder of the section, we outline the proof of Theorem 3.1. Assume we are given $L_1 = L(\mathcal{V}_1)$ and $L_2 = L(\mathcal{V}_2)$ and this is a non-trivial instance of separability, meaning L_1, L_2 are not regular and $L_1 \cap L_2 = \emptyset$. For proving separability, we could enumerate regular languages until we find a separator. The difficult part is disproving separability. Inseparability of L_1 and L_2 is witnessed by a set of words $W \subseteq L_1$ so that every regular language R containing them already intersects L_2, formally: $W \subseteq R$ implies $R \cap L_2 \neq \emptyset$. Showing the existence of such a set W is difficult for two reasons. First, it is unclear which sets of words ensure the universal quantification over all regular languages. Second, as we have a non-trivial instance of separability, W (if it exists) will be a non-regular language. So it is unclear how to represent it in a finite way and how to check its existence.

To address the first problem and understand the sets of words that disprove separability, we use diagonalization. Call an *(L_2-)separator candidate* a regular language that is disjoint from L_2. Let $R_1, R_2, \ldots$ be an enumeration of the separator candidates. If L_1 is not separable from L_2, for every R_i there is a word $w_i \in L_1$ with $w_i \notin R_i$. We call such a set of words $W = \{w_1, w_2, \ldots\}$ that escapes every separator candidate an *inseparability witness*.

▶ **Observation 3.3.** *$L_1 \not| L_2$ if and only if there is an inseparability witness.*

Our decision procedure will check the existence of an inseparability witness. We obtain the procedure in four steps: the first is a simplification, the second is devoted to understanding the separator candidates, the third is another simplification, and the last characterizes the inseparability witnesses and checks their existence.

Step 1: Fixing L_2. We first reduce general regular separability to regular separability from the Dyck language. The reduction is simple and works just as for finite words [16].

▶ **Lemma 3.4.** *Given Büchi VASS $\mathcal{V}_1$ and $\mathcal{V}_2$, we can compute a Büchi VASS $\mathcal{V}$ over Σ_n so that $L(\mathcal{V}_1) \,|\, L(\mathcal{V}_2)$ if and only if $L(\mathcal{V}) \,|\, D_n$, where n is the dimension of $\mathcal{V}_2$.*

Step 2: Understanding the Separator Candidates. To understand the regular languages that are disjoint from D_n, we will define *basic separators*, sets $P_{i,k}$ and $S_{\boldsymbol{x},k}$, on which we elaborate in a moment. The following theorem says that finite unions of basic separators are sufficient for regular separability. This is our first technical result and shown in Section 4.

▶ **Theorem 3.5.** *If $R \subseteq \Sigma_n^\omega$ is regular and $R \cap D_n = \emptyset$, then R is included in a finite union of basic separators.*

For the definition of $P_{i,k}$, we note that the words outside D_n have, for some index $i \in [1, n]$, an earliest moment in time where the balance between a_i and $\bar{a}_i$ falls below zero. To turn this into a regular language, we impose an upper bound $k \in \mathbb{N}$ on the (positive) balance between the letters a_i and $\bar{a}_i$ that is maintained until the earliest moment is reached. This yields the regular language

$$P_{i,k} \;:=\; \{w \in \Sigma_n^\omega \mid \exists v \in \mathrm{prefix}(w) \colon \varphi_i(v) < 0 \wedge \forall u \in \mathrm{prefix}(v) \colon \varphi_i(u) \leq k\}.$$

The family of languages $P_{i,k}$ already captures the complement of D_n. The problem is that we may need infinitely many such languages to cover the language R of interest. For every bound k, a regular R with $R \cap D_1 = \emptyset$ may contain a word with a higher balance before falling below zero, take for example $R = a_1^* \bar{a}_1^\omega$. The first insight is that if R can fall below zero from arbitrarily high values, then the underlying Büchi automaton has to contain loops with a negative balance. The R thus contains words uv with an unconstrained prefix and a suffix that decomposes into $v = v_1 v_2 \cdots$ so that every infix $w = v_\ell$ has a negative balance on letter a_i. The observation suggests the definition of a language that contains precisely the words $u.v$. To make the language regular, we impose a bound k on the positive balance that can be used during the infixes w. Call the resulting language $S_{i,k}$. Unfortunately, taking the $P_{i,k}$ and the $S_{i,k}$ as basic separators is still not enough: Figure 1(right) exhibits a regular language, disjoint from D_1, that is not included in a finite union of $P_{i,k}$ and $S_{i,k}$, because it contains infixes where the balance on each letter exceeds all bounds in each coordinate.

The second insight is that we can catch the remaining words with a version of $S_{i,k}$ that weights coordinates with some $\boldsymbol{x} \in \mathbb{N}^n$. Let us give some intuition on this. The words from R that we cannot catch with a $P_{i,k}$ must come across, for each i that becomes negative, a loop with positive balance on i (otherwise, the balance on those i would be bounded). But then, the only way such words can avoid D_1 is by ending up in a strongly connected component where *every* loop (with a final state) makes progress towards crossing 0, i.e. is negative in some coordinate. One can then conclude that even all $\mathbb{Q}_{\geq 0}$-linear combinations of loops (a convex set) must avoid the positive orthant $\mathbb{Q}^n_{\geq 0} \subset \mathbb{Q}^n$. By the Hyperplane Separation Theorem (we use it in the form of Farkas' Lemma), this is certified by a hyperplane that separates all loop effects from $\mathbb{Q}^n_{\geq 0}$. This hyperplane is given by some orthogonal vector $\boldsymbol{x} \in \mathbb{N}^n$, meaning that every loop balance must have negative scalar product with $\boldsymbol{x}$. Hence, we can catch these words by:

$$S_{\boldsymbol{x},k} := \left\{ u.v \in \Sigma_n^\omega \;\middle|\; \begin{array}{l} \text{a.) } \forall f \in \mathrm{infix}(v)\colon \langle \boldsymbol{x}, \varphi(f)\rangle \leq k, \text{ and} \\ \text{b.) } v = v_0.v_1.v_2 \cdots \wedge \forall \ell \in \mathbb{N}\colon \langle \boldsymbol{x}, \varphi(v_\ell)\rangle < 0 \end{array} \right\}.$$

Coming back to Figure 1(right), the weight vector $\boldsymbol{x} = (1,1)$ guarantees that the weighted balance decreases indefinitely and also the weighted balances of all infixes stay bounded. In [16], a similar argument has been used to show sufficiency of basic separators.

Step 3: Pumpable Languages. With the basic separators at hand, the task is to understand the sets of words witnessing inseparability. While studying this problem, we observed that the argumentation for the $P_{i,k}$ was always similar to the one for the $S_{\boldsymbol{x},k}$. This led us to the question of whether we can get rid of the $P_{i,k}$ in separators. The answer is positive, and hinges on a new notion of pumpability for languages over Σ_n.

Call infinite words u and v *equivalent*, written $u \sim v$, if v can be obtained from u by removing and inserting finitely many letters: There are $u_0, v_0 \in \Sigma^*$ and $w \in \Sigma^\omega$ such that $u = u_0 w$ and $v = v_0 w$. We say that a language $L \subseteq \Sigma_n^\omega$ is *pumpable* if for every $w \in L$ and every $k \in \mathbb{N}$, there exists a decomposition $w = w_0 w_1$ and a word $w_0' \in \Sigma_n^*$ that is a prefix of a word in D_n such that $w_0'.w_1 \in L$ and the letter balance satisfies the following: (a) $\varphi(w_0') \geq \varphi(w_0)$ and (b) for the indices $i \in [1,n]$ where φ_i becomes negative on some prefix of w, we have $\varphi_i(w_0') \geq \max\{\varphi_i(w_0), 0\} + k$. The consequence of this definition is that a pumpable language leaves every language $P_k := \bigcup_{i\in[1,n]} P_{i,k}$. Indeed, for every word $w \in L$ and every $k \in \mathbb{N}$, there is a word $w' \in L$ with $w \sim w'$ where the letter balance exceeds k before becoming negative, and thus $w' \notin P_k$. With the previous characterization of separator candidates, what is left to separate L from D_n are the languages $S_{\boldsymbol{x},k}$.

▶ **Lemma 3.6.** *If $L \subseteq \Sigma_n^\omega$ is pumpable, then $L \mid D_n$ if and only if $L \mid\!\!\!\!_{\mathsf{lim}} D_n$, where $L \mid\!\!\!\!_{\mathsf{lim}} D_n$ means $L \subseteq \bigcup_{\boldsymbol{x}\in X} S_{\boldsymbol{x},k}$ for some finite set $X \subseteq \mathbb{N}^n$ and some $k \in \mathbb{N}$.*

In our context, pumpability is interesting because we can turn every Büchi VASS language into a pumpable language without affecting separability.

▶ **Theorem 3.7.** *Let $\mathcal{V}$ be a d-dim. Büchi VASS over Σ_n. We can compute a d-dim. Büchi VASS $\mathcal{V}_{\mathsf{pump}}$ that satisfies the following:*

1. *$L(\mathcal{V}_{\mathsf{pump}})$ is pumpable,*
2. *there is a $k \in \mathbb{N}$ so that $L(\mathcal{V}_{\mathsf{pump}}) \subseteq L(\mathcal{V}) \subseteq L(\mathcal{V}_{\mathsf{pump}}) \cup P_k$, and*
3. *$L(\mathcal{V}) \mid D_n$ if and only if $L(\mathcal{V}_{\mathsf{pump}}) \mid D_n$.*

The construction of $\mathcal{V}_{\mathsf{pump}}$ employs the Karp-Miller graph in an original way, namely to track the unboundedness of letter balances. Let $\bar{\mathcal{V}}$ be the $(d+n)$-dimensional Büchi VASS obtained from $\mathcal{V}$ by tracking the effect of the letters from Σ_n in n additional counters. For $\bar{\mathcal{V}}$, we construct the Karp-Miller graph. The relationship between the languages of $\mathsf{KM}(\bar{\mathcal{V}})$ and $\mathcal{V}$ is as follows. For all words where every letter balance stays non-negative, their runs in $\mathcal{V}$ can be mimicked in $\mathsf{KM}(\bar{\mathcal{V}})$. For all other words, where the balance eventually becomes negative, this only holds if the corresponding counter in $\bar{\mathcal{V}}$ has been raised to ω beforehand. Essentially, the new Büchi VASS $\mathcal{V}_{\mathsf{pump}}$ restricts $\mathcal{V}$ to those runs that have counterparts in $\mathsf{KM}(\bar{\mathcal{V}})$. This is achieved with a simple product construction of $\mathcal{V}$ and $\mathsf{KM}(\bar{\mathcal{V}})$. The thing to note is that every word from $L(\mathcal{V})$ that does not make it into $L(\mathcal{V}_{\mathsf{pump}})$ belongs to P_k, where k is the maximum concrete number in $\mathsf{KM}(\bar{\mathcal{V}})$: A run in $\mathcal{V}$ that cannot be mimicked in $\mathsf{KM}(\bar{\mathcal{V}})$ will at some point have a negative letter balance, before reaching ω in $\mathsf{KM}(\bar{\mathcal{V}})$ in that component; thus all counter values had been at most k until that point.

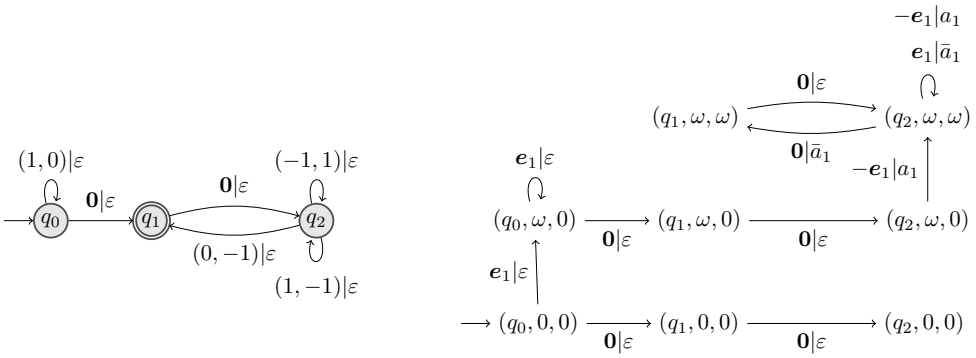


Figure 2 Left: The Büchi VASS $\bar{\mathcal{V}}$ constructed from the Büchi VASS $\mathcal{V}$ found in Figure 1(left). Note how the added second counter tracks the letter balance of the now removed transition labels, incrementing on letter a_1 and decrementing on letter $\bar{a}_1$. Right: The Büchi VASS $\mathcal{V}_{\mathsf{pump}}$ corresponding to $\mathcal{V}$ as given by Theorem 3.7. Here we did not mark the final states to reduce visual clutter; every state that includes q_1 is considered final. Similarly, the two labels above the loop in the top right correspond to two distinct transitions. Note that $\mathcal{V}_{\mathsf{pump}}$ essentially looks like $\mathsf{KM}(\bar{\mathcal{V}})$, just with different transition labels.

An example on how to construct $\bar{\mathcal{V}}$ and $\mathcal{V}_{\mathsf{pump}}$ can be found in Figure 2, where both were constructed for the Büchi VASS found in Figure 1(left).

In the proof of Theorem 3.5, we make use of Theorem 3.7 (recall that a regular language is the language of a 0-dimensional Büchi VASS). This may look like cyclic reasoning, but it is not: We will show Theorem 3.7(1)+(2) directly, using the arguments above. With this, we prove Theorem 3.5, which in turn is used to derive Lemma 3.6 and Theorem 3.7(3).

Step 4: Non-Separability Witnesses and Decidability. Because of pumpability, it remains to decide whether a Büchi VASS language $L(\mathcal{V})$ is included in a finite union $\bigcup_{\boldsymbol{x}\in X} S_{\boldsymbol{x},k}$ for some k. Part of the difficulty is that we have no bound on the cardinality of X. To circumvent this, we decompose $L(\mathcal{V})$ into a finite union $\bigcup_\pi L_\pi(\mathcal{V})$, where π is a *profile*, meaning a set of edges in $\mathsf{KM}(\mathcal{V})$ seen infinitely often during a run of $\mathcal{V}$. We then show that each $L_\pi(\mathcal{V})$ is either (i) included in a single separator $S_{\boldsymbol{x},k}$ or (ii) escapes every finite union $\bigcup_{\boldsymbol{x}\in X} S_{\boldsymbol{x},k}$.

Here, it is key to show an even stronger fact: In case (i), not only $L_\pi(\mathcal{V})$ is included in some $S_{\boldsymbol{x},k}$, but the entire set of runs in $\mathsf{KM}(\mathcal{V})$ that eventually remain in π. The advantage of strengthening is that finiteness of $\mathsf{KM}(\mathcal{V})$ allows us to express inclusion in $S_{\boldsymbol{x},k}$, for some k, as a *finite* system of linear inequalities over $\boldsymbol{x}$: We say that (1) the balance of every primitive cycle, weighted by $\boldsymbol{x}$, is at most zero and (2) the balance, weighted by $\boldsymbol{x}$, of some cycle containing all edges from π is negative. Here, (1) and (2) correspond to Conditions a.) and b.) of $S_{\boldsymbol{x},k}$. If they are met, then the runs of $\mathsf{KM}(\mathcal{V})$ along π are included in $S_{\boldsymbol{x},k}$ for some k.

We then prove that if the system is not feasible, then $\mathcal{V}$ has runs that escape every finite union $\bigcup_{\boldsymbol{x}\in X} S_{\boldsymbol{x},k}$. To this end, we employ Farkas' Lemma: It tells us that if there is no solution, then the dual system has a solution. The solution of the dual system can be interpreted as an executable linear combination of primitive cycles with non-negative balances. We show that these cycles can be arranged in a pattern in $\mathsf{KM}(\mathcal{V})$ we call *inseparability flower*. Such an inseparability flower then yields a sequence of runs $\rho_1, \rho_2, \ldots$ in $\mathsf{KM}(\mathcal{V})$ such that ρ_k escapes $S_{\boldsymbol{x},k}$ for every vector $\boldsymbol{x}$. Finally, pumpability allows us to lift these runs of $\mathsf{KM}(\mathcal{V})$ to runs of $\mathcal{V}$ and thus conclude inseparability.

This equips us with two possible decision procedures: We can either check solvability of each system of inequalities, or detect inseparability flowers in $\mathsf{KM}(\mathcal{V})$.

4 Basic Separators

We prove Theorem 3.5, that any regular language R over Σ_n with $R \cap D_n = \emptyset$ is contained in a finite union of languages $P_{i,k}$ and $S_{\boldsymbol{x},k}$. Note that a single value of k is sufficient, since we have $P_{i,k} \subseteq P_{i,k+1}$ and $S_{\boldsymbol{x},k} \subseteq S_{\boldsymbol{x},k+1}$ for each $i, \boldsymbol{x}, k$. The proof decomposes the Büchi automaton for R in a way that allows us to forget about connectedness issues and reason over cycles (and their letter balances) using techniques from linear algebra. We make use of the following basic fact from linear programming [40, Corollary 7.1f].

▶ **Theorem 4.1** (Farkas' Lemma (variant), [40]). *Let $\boldsymbol{A} \in \mathbb{Q}^{m \times n}$ be a matrix and let $\boldsymbol{b} \in \mathbb{Q}^m$ be a vector. Then the system $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ has a solution $\boldsymbol{x} \in \mathbb{Q}^n_{\geq 0}$ if and only if $\boldsymbol{y}^\top \boldsymbol{b} \geq 0$ for each vector $\boldsymbol{y} \in \mathbb{Q}^m_{\geq 0}$ with $\boldsymbol{y}^\top \boldsymbol{A} \geq \boldsymbol{0}$.*

Decomposing with profiles. We decompose $R = L(\mathcal{A})$ into a (not necessarily disjoint) union of several languages, each linked to a so-called *profile*. We will later see that for pumpable R, every such profile language already has to be contained in a single $S_{\boldsymbol{x},k}$.

▶ **Definition 4.2.** *Let $\mathcal{A}$ be a Büchi automaton. A* profile of $\mathcal{A}$ *is a set π of transitions of $\mathcal{A}$ for which there exists a cycle σ_π in $\mathcal{A}$ such that (a) σ_π contains exactly the transitions in π, and (b) σ_π starts (and ends) in a final state q_π.*

We denote by $\Pi(\mathcal{A})$ the finite set of profiles of $\mathcal{A}$. Moreover, we associate to every accepting run ρ of $\mathcal{A}$ its profile $\Pi(\rho)$, which contains exactly the transitions appearing infinitely often in ρ. This definition is sound, as the infinitely occurring transitions of an accepting run must form a cycle due to repetition, which visits a final state due to acceptance.

Given a profile π of $\mathcal{A}$, we define $L_\pi(\mathcal{A}) \subseteq L(\mathcal{A})$ to be the language of all words that have an accepting run ρ of $\mathcal{A}$ with $\Pi(\rho) = \pi$. Note that this language is still regular: From $\mathcal{A}$ one can construct a Büchi automaton that guesses a point after which only transitions from π can occur, and once this point is reached it keeps a list of already used transitions from π in each state. Then only once all transitions of π have been used the state becomes final and the list is set back to empty.

This now allows us to view R as the union of the languages $L_\pi(\mathcal{A})$ with $\pi \in \Pi(\mathcal{A})$. We show that each language $L_\pi(\mathcal{A})$ is either contained in $S_{\boldsymbol{x},k}$ for some $\boldsymbol{x}, k$, or there is a cycle that, assuming the pumpability from the previous section, makes $L_\pi(\mathcal{A})$ intersect D_n.

▶ **Lemma 4.3.** *Let $\mathcal{A}$ be a Büchi automaton over Σ_n and let π be one of its profiles. Then one of the following conditions holds:*

- **(i)** *There is a number $k \in \mathbb{N}$ and a vector $\boldsymbol{x} \in \mathbb{N}^n$ such that $L_\pi(\mathcal{A}) \subseteq S_{\boldsymbol{x},k}$, or*
- **(ii)** *there is a cycle σ' in $\mathcal{A}$ over w' with $\varphi(w') \geq \boldsymbol{0}$, and σ' contains all transitions from π.*

Assume $L_\pi(\mathcal{A}) \neq \emptyset$, otherwise Condition (i) trivially holds. We build a system $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ of linear inequalities as follows. It contains one inequality $\langle \boldsymbol{x}, \varphi(v) \rangle \leq 0$ for each word v read by a primitive cycle of transitions in π. By *primitive cycle* we mean a cycle that does not repeat a state. Moreover, the system contains the inequality $\langle \boldsymbol{x}, \varphi(v_\pi) \rangle \leq -1$ for the cycle σ_π over v_π that justifies the profile π. Let us quickly remark that the solution space of the system $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ is independent of the precise choice of the justifying cycle σ_π: To see this, we claim that $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ holds if and only if all primitive cyles in π have an $\boldsymbol{x}$-weighted balance at most zero, and at least one primitive cycle in π has a strictly negative $\boldsymbol{x}$-weighted balance. For the "if" direction, note that a sufficiently long repetition of σ_π will contain each primitive cycle as a (possibly non-contiguous) subsequence. This means, the repetition, and thus σ_π,

must have a strictly negative $\boldsymbol{x}$-weighted balance. For the converse, we observe that σ_π can be decomposed into primitive cycles. Thus, if σ_π has strictly negative $\boldsymbol{x}$-weighted balance, then so must at least one of its constituent primitive cycles.

Applying Farkas' Lemma to $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ either yields a solution $\boldsymbol{x} \in \mathbb{Q}^n_{\geq 0}$ or a vector $\boldsymbol{y} \in \mathbb{Q}^m_{\geq 0}$ with $\boldsymbol{y}^\top \boldsymbol{A}_\pi \geq \boldsymbol{0}$ and $\boldsymbol{y}^\top \boldsymbol{b} < 0$. In both cases we assume wlog. that the given vector has entries in $\mathbb{N}$, as we can always multiply with the lcm of the denominators.

Suppose we have a solution $\boldsymbol{x}$. We claim that then $L_\pi(\mathcal{A}) \subseteq S_{\boldsymbol{x},k}$, where $k = |Q_\pi| \cdot h$ and h is the maximal length of a transition label of $\mathcal{A}$. This is because $\boldsymbol{x}$ weights primitive cycles non-positively, and k is chosen such that for any infix v of a word in $L_\pi(\mathcal{A})$, if $|v| > k$, then v's associated transition sequence has to contain a primitive cycle. Thus, infixes at almost all start positions of a word in $L_\pi(\mathcal{A})$ must have $\boldsymbol{x}$-weighted balance $\leq k$.

If we obtain a vector $\boldsymbol{y} = (y_1, \ldots, y_m)$, then we can view it as a selection of rows in the matrix $\boldsymbol{A}_\pi$, where the jth row is being selected y_j many times. Since each row corresponds to a cycle, this is also a selection of cycles. Then by $\boldsymbol{y}^\top \boldsymbol{b} < 0$ we selected σ_π, where we can insert the other selected cycles. By $\boldsymbol{y}^\top \boldsymbol{A}_\pi \geq \boldsymbol{0}$ this forms a cycle σ' as required, with non-negative letter balance for all letter pairs.

Here, we used a system of linear inequalities $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$, which was solely dependent on $\mathcal{A}$ and π. We reasoned that if this system has a solution, then Condition (i) has to hold. This is a fact that we want to refer to in a later proof, and therefore we formalize it here.

▶ **Corollary 4.4.** *If $\mathcal{A}$ is a Büchi automaton with a profile π for which there is an $\boldsymbol{x} \in \mathbb{N}^n$ with $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$, then $L_\pi(\mathcal{A}) \subseteq S_{\boldsymbol{x},k}$ for some $k \in \mathbb{N}$.*

With Theorem 3.7 and Lemma 4.3, we can now show Theorem 3.5. Suppose $R = L(\mathcal{A})$ for some Büchi automaton $\mathcal{A}$. First, applying Theorem 3.7 with $d = 0$ yields a Büchi automaton $\mathcal{A}_{\mathsf{pump}}$ such that $L(\mathcal{A}) \subseteq L(\mathcal{A}_{\mathsf{pump}}) \cup P_\ell$ for some $\ell \in \mathbb{N}$ and $L(\mathcal{A}_{\mathsf{pump}}) \cap D_n = \emptyset$. Therefore, it suffices to show that $L(\mathcal{A}_{\mathsf{pump}})$ is included in a finite union of languages $S_{\boldsymbol{x},k}$. Suppose not. Then the set $L(\mathcal{A}_{\mathsf{pump}})$ decomposes into the sets $L_\pi(\mathcal{A}_{\mathsf{pump}})$ for $\pi \in \Pi(\mathcal{A}_{\mathsf{pump}})$. By Lemma 4.3, we know that for some π, Condition (ii) must hold: Otherwise, each $L_\pi(\mathcal{A}_{\mathsf{pump}})$ would be included in some $S_{\boldsymbol{x},k}$. But if (ii) holds for π, then there is a cycle σ' in $\mathcal{A}_{\mathsf{pump}}$ that contains π (and thus visits a final state) and reads a word v with $\varphi(v) \geq \boldsymbol{0}$. Now for some finite prefix u, the word uv^ω belongs to $L(\mathcal{A}_{\mathsf{pump}})$. Since $\varphi(v) \geq \boldsymbol{0}$, there is some lower bound $B \in \mathbb{Z}$ such that for each $i \in [1, n]$ and every prefix p of uv^ω, we have $\varphi_i(p) \geq B$. Finally, since $L(\mathcal{A}_{\mathsf{pump}})$ is pumpable, we can exchange a prefix in $w = uv^\omega$ to obtain another word $w' \in L(\mathcal{A}_{\mathsf{pump}})$ where every prefix p has $\varphi(p) \geq \boldsymbol{0}$. Hence $w' \in D_n$ and thus $L(\mathcal{A}_{\mathsf{pump}}) \cap D_n \neq \emptyset$, a contradiction.

5 Deciding Regular Separability

We now present the algorithm to decide, given a Büchi VASS $\mathcal{V}$ whether $L(\mathcal{V}) \mid D_n$. We first employ Theorem 3.7, because for pumpable languages we only have to deal with one type of basic separators. The next step is to generalize the notion of profiles from Büchi automata to Büchi VASS. Recall that for a sequence χ of transitions in $\mathcal{V}$, $\delta(\chi)$ denotes its effect on the counters of $\mathcal{V}$. If χ is a transition sequence in $\mathsf{KM}(\mathcal{V})$, then χ is labeled with a transition sequence of $\mathcal{V}$, so we define $\delta(\chi)$ accordingly. Since we consider Büchi VASS with input alphabet Σ_n, we write $\varphi(\chi)$ for the image of the input word under φ. Again, this notation is used for transition sequences in $\mathsf{KM}(\mathcal{V})$. We also write $\Delta(\chi) = (\delta(\chi), \varphi(\chi))$.

▶ **Definition 5.1.** *Let $\mathcal{V}$ be a Büchi VASS. A* profile for $\mathcal{V}$ *is a set π of edges in $\mathsf{KM}(\mathcal{V})$ for which there exists a cycle σ in $\mathsf{KM}(\mathcal{V})$ such that (i) σ contains exactly the edges in π, (ii) σ starts (and ends) in a final state, and (iii) $\delta(\sigma) \geq \boldsymbol{0}$.*

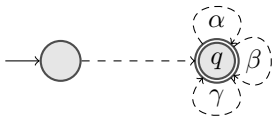
Clearly, every Büchi VASS has a finite set of profiles, which we denote by $\Pi(\mathcal{V})$. Moreover, $\Pi(\mathcal{V})$ can be constructed effectively: Given a set of edges, a simple reduction to checking unboundedness of a counter can be used to check if it is a profile. Furthermore, to every run ρ of $\mathcal{V}$, we can associate a profile: The run ρ must have a corresponding run in $\mathsf{KM}(\mathcal{V})$, which has a finite set $\Pi(\rho)$ of edges that are used infinitely often. Thus, ρ decomposes as $\rho_0\rho_1$ such that ρ_1 only contains edges from π. Then, ρ_1 decomposes into $\sigma_1\sigma_2\cdots$ such that each σ_i uses every edge from $\Pi(\rho)$ at least once and starts (and ends) in a final state. Since $\leq$ is a well-quasi ordering on $\mathbb{N}^n$, there are $r < s$ such that $\delta(\sigma_r \cdots \sigma_s) \geq \mathbf{0}$. Thus, $\sigma = \sigma_r \cdots \sigma_s$ is our desired transition sequence showing that $\Pi(\rho)$ is a profile. For each $\pi \in \Pi(\mathcal{V})$, we denote by $L_\pi(\mathcal{V})$ the set of all words accepted by runs ρ of $\mathcal{V}$ for which $\Pi(\rho) = \pi$. Then clearly:

▶ **Lemma 5.2.** $L(\mathcal{V}) = \bigcup_{\pi\in\Pi(\mathcal{V})} L_\pi(\mathcal{V})$.

A system of inequalities for each profile. Our next step is to associate with each profile $\pi \in \Pi(\mathcal{V})$ a system of linear inequalities. We need some terminology. A *π-cycle* is a cycle σ in $\mathsf{KM}(\mathcal{V})$ that only contains edges in π. If in addition, σ visits each state of $\mathsf{KM}(\mathcal{V})$ at most once, except for the initial state, which is visited twice, then σ is a *primitive π-cycle*. Clearly, a primitive π-cycle has length $\leq |\pi|$. Moreover, from every π-cycle σ, one can successively cut out primitive π-cycles until it is empty. Therefore, if $\tau_1, \ldots, \tau_m$ are the primitive π-cycles of $\mathsf{KM}(\mathcal{V})$, then there are numbers $r_1, \ldots, r_m \in \mathbb{N}$ such that $\Delta(\sigma) = r_1 \cdot \Delta(\tau_1) + \cdots + r_m \cdot \Delta(\tau_m)$. We call σ a *complete π-cycle* if this holds for some $r_1, \ldots, r_m \geq 1$. Observe that if π is a profile, then this is always witnessed by a complete π-cycle: Take any cycle σ witnessing that π is a profile. Then $\sigma^{|\pi|}$ contains each primitive π-cycle as a subsequence. Hence, the cycle $\sigma^{m\cdot|\pi|}$ is complete: We can carry out the cutting in each factor $\sigma^{|\pi|}$ so as to cut some τ_i at least once. Moreover, $\sigma^{m\cdot|\pi|}$ still witnesses that π is a profile, since $\delta(\sigma^{m\cdot|\pi|}) = m \cdot |\pi| \cdot \delta(\sigma) \geq \mathbf{0}$.

Let us now construct the system of inequalities associated with π. Let σ be a complete π-cycle witnessing that π is a profile and let $\tau_1, \ldots, \tau_m$ be the primitive π-cycles. Let $\boldsymbol{A}_\pi \in \mathbb{Z}^{(m+1)\times n}$ be the matrix with rows $\varphi(\tau_1), \ldots, \varphi(\tau_m), \varphi(\sigma)$, and let $\boldsymbol{b} \in \mathbb{Z}^{m+1}$ be the column vector $(0, \ldots, 0, -1)$. Then clearly, $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ is equivalent to $\langle \boldsymbol{x}, \varphi(\sigma)\rangle < 0$ and $\langle \boldsymbol{x}, \varphi(\tau)\rangle \leq 0$ for each primitive π-cycle τ.

Inseparability flowers. An *inseparability flower* is a structure in the Karp-Miller graph $\mathsf{KM}(\mathcal{V})$ as depicted below. It consists of a final state q and three cycles α, β, γ that all start in q and that meet the given conditions.



$\delta(\alpha\beta\gamma) \geq \mathbf{0}$
$\varphi(\alpha\beta) \geq \mathbf{0}$
$\varphi(\alpha\beta\gamma) \in \mathbb{Q} \cdot \varphi(\alpha)$

Let us give some intuition on why such a flower is the relevant structure to look for. True to its name, an inseparability flower guarantees the existence of an inseparability witness, i.e. a family of words accepted by the pumpable Büchi VASS $\mathcal{V}$ that escape every basic separator $S_{\boldsymbol{x},k}$. Such a family of words therefore needs an accepting run for each member, and the three conditions of the flower provide such runs: The first condition ensures that the three cycles actually correspond to a transition sequence enabled in $\mathcal{V}$. The second condition guarantees that for every $\boldsymbol{x} \in \mathbb{N}^n$, the $\boldsymbol{x}$-weighted letter balance of α or of β is positive; unless they are both zero, in which case the third condition ensures that $\alpha\beta\gamma$ has $\boldsymbol{x}$-weighted balance zero. This allows us to construct, for each k, a run that escapes $S_{\boldsymbol{x},k}$ for all $\boldsymbol{x}$: By sufficiently

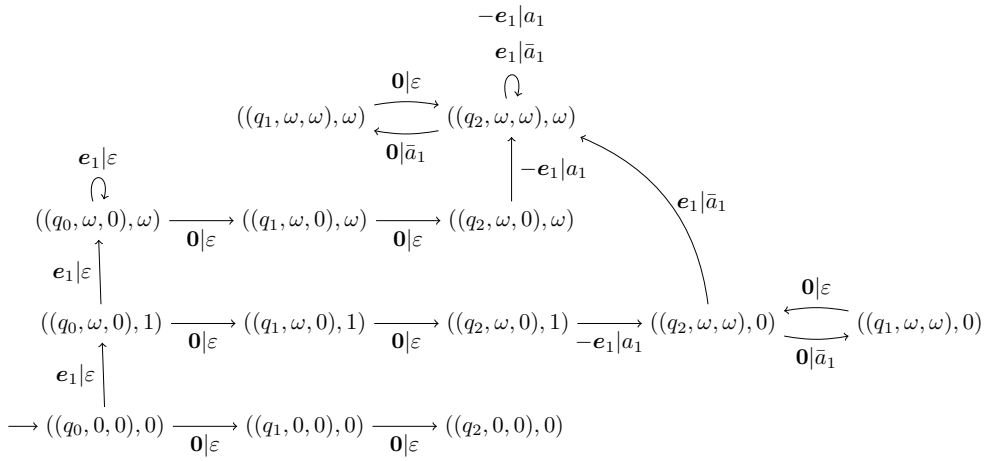


Figure 3 The Karp-Miller graph $\mathsf{KM}(\mathcal{V}_{\mathsf{pump}})$ of the Büchi VASS $\mathcal{V}_{\mathsf{pump}}$ from Figure 2(left). Here we did not mark the final states to reduce visual clutter; every state that includes q_1 is considered final. For similar reasons, we also only labelled the edges of the graph with letters and counter effects. The proper edge labels would be full transitions of $\mathcal{V}_{\mathsf{pump}}$, including source and target state.

repeating each cycle α, β, and γ, we obtain a run that for each $\boldsymbol{x} \in \mathbb{N}^n$, will either (i) have infixes with $\boldsymbol{x}$-weighted balance $> k$, or (ii) attain some $\boldsymbol{x}$-weighted balance infinitely often. Each of these properties rules out membership in $S_{\boldsymbol{x},k}$. Proposition 5.5 proves this formally.

▶ **Theorem 5.3.** *Let $\mathcal{V}$ be a Büchi VASS such that $L(\mathcal{V})$ is pumpable. Then the following are equivalent: (i) $L(\mathcal{V}) \not| D_n$. (ii) There is a profile $\pi \in \Pi(\mathcal{V})$ such that the system $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ has no solution $\boldsymbol{x} \in \mathbb{N}^n$. (iii) There exists an inseparability flower in $\mathsf{KM}(\mathcal{V})$.*

The decision procedure. Before we prove Theorem 5.3, let us see how to use it to decide separability. Given Büchi VASS $\mathcal{V}_1$ and $\mathcal{V}_2$, we can compute $\mathcal{V}$ so that $L(\mathcal{V}_1) \mid L(\mathcal{V}_2)$ if and only if $L(\mathcal{V}) \mid D_n$, by Lemma 3.4. Then Theorem 3.7 tells us that $L(\mathcal{V}_{\mathsf{pump}})$ is pumpable and we have $L(\mathcal{V}) \mid D_n$ if and only if $L(\mathcal{V}_{\mathsf{pump}}) \mid D_n$. Finally, by Theorem 5.3, we can check whether $L(\mathcal{V}_{\mathsf{pump}}) \mid D_n$ by checking the systems $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ for satisfiability: If there is a solution for every $\pi \in \Pi(\mathcal{V}_{\mathsf{pump}})$, then we have separability; otherwise, we have inseparability. Since the systems $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ are constructed directly from $\mathsf{KM}(\mathcal{V}_{\mathsf{pump}})$, we need to explicitly construct the latter. Therefore our procedure may take Ackermann time, because Karp-Miller graphs can be Ackermann large [31, Theorem 2].

▶ **Example 5.4.** Consider the instance of regular separability where our two inputs are the Büchi VASS $\mathcal{V}$ found in Figure 1(left), and another Büchi VASS accepting the language D_1. Since we are already in the case of wanting to decide $L(\mathcal{V}) \mid D_1$, we can skip the first step of applying Lemma 3.4. The second step is to apply Theorem 3.7 and construct $\mathcal{V}_{\mathsf{pump}}$, which we have already done for this case in Figure 2(right).

Now we have to construct $\mathsf{KM}(\mathcal{V}_{\mathsf{pump}})$, which can be found in Figure 3. There are two relevant parts of $\mathsf{KM}(\mathcal{V}_{\mathsf{pump}})$, where we can find cycles involving a final state: (1) the part on the right, where the state tuples contain ω twice and the counter value is 0, and (2) the part at the top with triple ωs. In the following we will only write down the states, as the counter values and the other contents of the state tuples will be clear from context.

For part (1), the Büchi VASS $\mathcal{V}_{\mathsf{pump}}$ has only a single profile π_1 containing only the two edges between q_1 and q_2. Since each π_1-cycle σ only consists of repetitions of the primitive cycle $q_1 \xrightarrow{\mathbf{0}|\varepsilon} q_2 \xrightarrow{\mathbf{0}|\bar{a}_1} q_1$, we have $\varphi(\sigma) < 0$. Therefore the system $\boldsymbol{A}_{\pi_1}\boldsymbol{x} \leq \boldsymbol{b}$ trivially has a solution $\boldsymbol{x} = 1$.

Regarding part (2), $\mathcal{V}_{\mathsf{pump}}$ has exactly two more profiles: profile π_2 containing only the two edges between q_1 and q_2, and profile π_3, which additionally contains the two loop edges on q_2. The cycles of π_2 look almost exactly like the cycles of π_1 with only the counter values of the nodes in the graph being different. Thus, the system $\boldsymbol{A}_{\pi_2}\boldsymbol{x} \leq \boldsymbol{b}$ is the exact same system as $\boldsymbol{A}_{\pi_1}\boldsymbol{x} \leq \boldsymbol{b}$ and also trivially has a solution $\boldsymbol{x} = 1$.

For π_3, we have as primitive cycles both the loop edges on q_2 as well as the primitive cycle of π_2. To obtain a complete π_3-cycle, we simply insert both loops into the π_2-cycle at q_2 forming the cycle $\sigma = q_1 \xrightarrow{\mathbf{0}|\varepsilon} q_2 \xrightarrow{-\boldsymbol{e}_1|a_1} q_2 \xrightarrow{\boldsymbol{e}_1|\bar{a}_1} q_2 \xrightarrow{\mathbf{0}|\bar{a}_1} q_1$. Since σ contains all primitive cycles exactly once without overlap, it is automatically complete. We also have $\delta(\sigma) = 0$, meaning σ is a cycle witnessing π_3 as a profile. Thus these cycles lead to the following system of inequalities $\boldsymbol{A}_{\pi_3}\boldsymbol{x} \leq \boldsymbol{b}$:

$$\begin{aligned}
1 \cdot x_1 &\leq 0 && \text{loop 1}\\
-1 \cdot x_1 &\leq 0 && \text{loop 2}\\
-1 \cdot x_1 &\leq 0 && \pi_2\text{-cycle}\\
-1 \cdot x_1 &\leq -1 && \text{complete } \pi_3\text{-cycle}
\end{aligned}$$

Clearly this system has no solution; the first and last inequality are contradictory. Therefore we conclude regular inseparability for $L(\mathcal{V})$ and D_1.

While not part of the decision procedure, for an inseparable instance of the problem as we have here, we can also find an inseparability flower in $\mathsf{KM}(\mathcal{V}_{\mathsf{pump}})$. In this case we have $\alpha = q_1 \xrightarrow{\mathbf{0}|\varepsilon} q_2 \xrightarrow{\mathbf{0}|\bar{a}_1} q_1$, $\beta = q_1 \xrightarrow{\mathbf{0}|\varepsilon} q_2 \xrightarrow{-\boldsymbol{e}_1|a_1} q_2 \xrightarrow{-\boldsymbol{e}_1|a_1} q_2 \xrightarrow{\mathbf{0}|\bar{a}_1} q_1$, and $\gamma = q_1 \xrightarrow{\mathbf{0}|\varepsilon} q_2 \xrightarrow{\boldsymbol{e}_1|\bar{a}_1} q_2 \xrightarrow{\boldsymbol{e}_1|\bar{a}_1} q_2 \xrightarrow{\mathbf{0}|\bar{a}_1} q_1$. This selection of cycles meets all the requirements of a flower: $\delta(\alpha\beta\gamma) = 0$, $\varphi(\alpha\beta) = 0$, and $\varphi(\alpha\beta\gamma) = -3 = 3 \cdot \varphi(\alpha)$.

Inseparability flowers disprove separability. The remainder of this section is devoted to proving Theorem 5.3. The implication "(i)⇒(ii)" follows by applying Corollary 4.4 to $\mathsf{KM}(\mathcal{V})$, viewed as a Büchi automaton (see full version). For "(iii)⇒(i)", we employ Lemma 3.6:

▶ **Proposition 5.5.** *If $L(\mathcal{V})$ is pumpable and $\mathsf{KM}(\mathcal{V})$ has an insep. flower, then $L(\mathcal{V}) \not| D_n$.*

Proof. Suppose there is an inseparability flower α, β, γ in $\mathsf{KM}(\mathcal{V})$ and also $L(\mathcal{V}) \mid D_n$. By Lemma 3.6, there is a $k \in \mathbb{N}$ and a finite set $X \subseteq \mathbb{N}^n$ such that $L(\mathcal{V}) \subseteq \bigcup_{\boldsymbol{x}\in X} S_{\boldsymbol{x},k}$. We claim that for every $\boldsymbol{x} \in \mathbb{N}^n$, at least one of the following holds:

$$\langle \boldsymbol{x}, \varphi(\alpha)\rangle > 0, \qquad \langle \boldsymbol{x}, \varphi(\beta)\rangle > 0, \qquad \text{or } \langle \boldsymbol{x}, \varphi(\alpha\beta\gamma)\rangle = 0. \tag{1}$$

Indeed, if $\langle \boldsymbol{x}, \varphi(\alpha)\rangle \leq 0$ and $\langle \boldsymbol{x}, \varphi(\beta)\rangle \leq 0$, then $\varphi(\alpha\beta) \geq \mathbf{0}$ implies that $\langle \boldsymbol{x}, \varphi(\alpha)\rangle = \langle \boldsymbol{x}, \varphi(\beta)\rangle = 0$. Since $\varphi(\alpha\beta\gamma) = N \cdot \varphi(\alpha)$ for some $N \in \mathbb{Q}$, we have $\varphi(\alpha\beta\gamma) = \mathbf{0}$. This proves the claim. Because of (1), the sequence $\alpha^{k+1}\beta^{k+1}\gamma^{k+1}$ either has an infix χ with $\langle \boldsymbol{x}, \varphi(\chi)\rangle > k$ or we have $\langle \boldsymbol{x}, \varphi(\alpha^{k+1}\beta^{k+1}\gamma^{k+1})\rangle = 0$. Since $\delta(\alpha^{k+1}\beta^{k+1}\gamma^{k+1}) \geq \mathbf{0}$, there is a run ρ such that $\rho\alpha^{k+1}\beta^{k+1}\gamma^{k+1}$ is a run in $\mathcal{V}$. Hence, $\rho(\alpha^{k+1}\beta^{k+1}\gamma^{k+1})^\omega$ is a run in $\mathcal{V}$ whose word cannot belong to $S_{\boldsymbol{x},k}$ for any $\boldsymbol{x} \in \mathbb{N}^n$, contradicting $L(\mathcal{V}) \subseteq \bigcup_{\boldsymbol{x}\in X} S_{\boldsymbol{x},k}$. ◀

Constructing inseparability flowers. It remains to show the implication "(ii)⇒(iii)". Suppose there is a profile $\pi \in \Pi(\mathcal{V})$ whose associated system of inequalities $\boldsymbol{A}_\pi \boldsymbol{x} \leq \boldsymbol{b}$ is unsatisfiable. By Farkas' Lemma, there exists a $\boldsymbol{y} \in \mathbb{N}^{m+1}$ such that $\boldsymbol{y}^\top \boldsymbol{A}_\pi \geq \boldsymbol{0}$ and $\boldsymbol{y}^\top \boldsymbol{b} < 0$. From this vector $\boldsymbol{y}$, we now construct an inseparability flower in $\mathsf{KM}(\mathcal{V})$.

Let σ be the complete π-cycle in $\mathsf{KM}(\mathcal{V})$ that was chosen to construct $\boldsymbol{A}_\pi$. Let $\tau_1, \ldots, \tau_m$ be the primitive π-cycles. Since σ is complete, there is a vector $\boldsymbol{r} = (r_1, \ldots, r_m) \in \mathbb{N}^m$ so that $r_1, \ldots, r_m \geq 1$ and $\Delta(\sigma) = r_1 \cdot \Delta(\tau_1) + \cdots + r_m \cdot \Delta(\tau_m)$. Moreover, since σ contains every edge of π, we can wlog. write $\sigma = \sigma_0 \cdots \sigma_m$ such that between σ_{i-1} and σ_i, σ arrives in the initial state of τ_i. The decomposition allows us to insert further repetitions of the primitive cycles. For $\boldsymbol{z} = (z_1, \ldots, z_m) \in \mathbb{N}^m$ with $\boldsymbol{z} \geq \boldsymbol{r}$, we define $\sigma^{\boldsymbol{z}}$ as $\sigma_0 \tau_1^{z_1 - r_1} \sigma_1 \cdots \tau_m^{z_m - r_m} \sigma_m$. Then $\Delta(\sigma^{\boldsymbol{z}}) = z_1 \cdot \Delta(\tau_1) + \cdots + z_m \cdot \Delta(\tau_m)$. In particular, for $\boldsymbol{s}, \boldsymbol{t} \geq \boldsymbol{r}$, we have $\Delta(\sigma^{\boldsymbol{s}} \sigma^{\boldsymbol{t}}) = \Delta(\sigma^{\boldsymbol{s}+\boldsymbol{t}})$.

Recall that every transition in a Karp-Miller graph is labeled by a VASS transition, and so every transition sequence χ in $\mathsf{KM}(\mathcal{V})$ is labeled by a transition sequence in $\mathcal{V}$, which we denote by $\mathsf{trans}(\chi)$. We now define the transition sequences α, β, and γ as $\mathsf{trans}(\sigma^{\boldsymbol{z}})$ for suitable vectors $\boldsymbol{z}$. For α, we take $\mathsf{trans}(\sigma)$, the transitions labeling the complete π-cycle. Observe that $\sigma = \sigma^{\boldsymbol{r}}$. We proceed to define $\beta = \mathsf{trans}(\sigma^{\boldsymbol{s}})$ and $\gamma = \mathsf{trans}(\sigma^{\boldsymbol{t}})$. The choice of the vectors $\boldsymbol{s}$ and $\boldsymbol{t}$ has to meet the requirements on an inseparability flower: $\varphi(\alpha\beta) \geq \boldsymbol{0}$, $\delta(\alpha\beta\gamma) \geq \boldsymbol{0}$, and $\varphi(\alpha\beta\gamma) \in \mathbb{Q} \cdot \varphi(\alpha)$.

Step I: Building β. We will define $\boldsymbol{s}$ so that $\varphi(\alpha\beta) = \varphi(\sigma^{\boldsymbol{r}}\sigma^{\boldsymbol{s}}) = \varphi(\sigma^{\boldsymbol{r}+\boldsymbol{s}}) \geq \boldsymbol{0}$. The remaining two requirements (i.e. $\delta(\alpha\beta\gamma) \geq \boldsymbol{0}$ and $\varphi(\alpha\beta\gamma) \in \mathbb{Q} \cdot \varphi(\alpha)$) will be ensured with an appropriate choice of $\boldsymbol{t}$ in Step II. Let us now describe how to pick $\boldsymbol{s}$. Recall that $\boldsymbol{y}$ is the vector from the application of Farkas' Lemma. It can be understood as assigning a repetition count y_i to every primitive cycle τ_i in the profile and a repetition count y_{m+1} to the complete π-cycle σ. Since $\boldsymbol{y}^\top \boldsymbol{A}_\pi \geq \boldsymbol{0}$, and since our goal is to make $\varphi(\alpha\beta)$ non-negative, we will use $\boldsymbol{y}$ to construct a vector $\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_m) \in \mathbb{N}^m$ so that $\varphi(\sigma^{\hat{\boldsymbol{y}}}) = \boldsymbol{y}^\top \boldsymbol{A}_\pi$. The right definition is $\hat{y}_i := y_i + y_{m+1} \cdot r_i$ for $i \in [1, m]$, because

$$\boldsymbol{y}^\top \boldsymbol{A}_\pi = \sum_{i=1}^{m} y_i \cdot \varphi(\tau_i) + y_{m+1} \cdot \varphi(\sigma) = \sum_{i=1}^{m} (y_i + y_{m+1} r_i) \varphi(\tau_i) = \varphi(\sigma^{\hat{\boldsymbol{y}}}).$$

We now choose $M \in \mathbb{N}$ such that $\boldsymbol{s} = M \cdot \hat{\boldsymbol{y}} - \boldsymbol{r} \geq \boldsymbol{r}$. This is possible since all entries in $\hat{\boldsymbol{y}}$ are positive, due to $y_{m+1} > 0$ by $\boldsymbol{y}^\top \boldsymbol{b} < 0$, and $r_i > 0$ for all i by definition. Then we have $\varphi(\alpha\beta) = \varphi(\sigma^{\boldsymbol{r}}\sigma^{\boldsymbol{s}}) = \varphi(\sigma^{\boldsymbol{r}+\boldsymbol{s}}) = \varphi(\sigma^{M \cdot \hat{\boldsymbol{y}}}) = M \cdot \varphi(\sigma^{\hat{\boldsymbol{y}}}) \geq \boldsymbol{0}$.

Step II: Building γ. It remains to define $\boldsymbol{t}$ so that $\gamma = \mathsf{trans}(\sigma^{\boldsymbol{t}})$ satisfies $\delta(\alpha\beta\gamma) = \delta(\sigma^{\boldsymbol{r}+\boldsymbol{s}+\boldsymbol{t}}) \geq \boldsymbol{0}$ and $\varphi(\alpha\beta\gamma) \in \mathbb{Q} \cdot \varphi(\alpha)$. The idea is to choose $\boldsymbol{t}$ so that $\boldsymbol{r} + \boldsymbol{s} + \boldsymbol{t}$ is a positive multiple of $\boldsymbol{r}$. Such a choice is possible, because $\boldsymbol{r}$ has positive entries everywhere: We pick $N \in \mathbb{N}$ such that $\boldsymbol{t} := N \cdot \boldsymbol{r} - \boldsymbol{s} - \boldsymbol{r} \geq \boldsymbol{r}$. Then indeed $\delta(\alpha\beta\gamma) = \delta(\sigma^{\boldsymbol{r}+\boldsymbol{s}+\boldsymbol{t}}) = \delta(\sigma^{N \cdot \boldsymbol{r}}) = N \cdot \delta(\sigma^{\boldsymbol{r}}) = N \cdot \delta(\sigma) \geq \boldsymbol{0}$ and $\varphi(\alpha\beta\gamma) = \varphi(\sigma^{\boldsymbol{r}+\boldsymbol{s}+\boldsymbol{t}}) = \varphi(\sigma^{N \cdot \boldsymbol{r}}) = N \cdot \varphi(\sigma^{\boldsymbol{r}}) = N \cdot \varphi(\alpha)$.

6 One-dimensional Büchi VASS

Our second contribution is the precise complexity of separability for the 1-dimensional case.

▶ **Theorem 6.1.** *Regular separability for 1-dimensional Büchi VASS with binary encoded updates is* PSPACE*-complete.*

For the lower bound, we use a simple reduction from the disjointness problem $L_1 \cap L_2 \stackrel{?}{=} \emptyset$ for finite-word languages of 1-dim. VASS [20]. However, we can also show that separability is PSPACE-hard even if the input languages are promised to be disjoint.

For the upper bound, we rely on the results in Section 5, but need a modification. There, to simplify the exposition, we first make the input language pumpable, which may incur an Ackermannian blowup. A closer look at the results, however, reveals that we can also check separability directly on the Karp-Miller graph of $\bar{\mathcal{V}}$ as defined in Section 3.

▶ **Proposition 6.2.** *Let $\mathcal{V}$ be a Büchi VASS with $L(\mathcal{V}) \subseteq \Sigma_n^\omega$. Then $L(\mathcal{V}) \not| D_n$ if and only if $\mathsf{KM}(\bar{\mathcal{V}})$ has an inseparability flower.*

Proposition 6.2 allows us to phrase inseparability as the existence of a run in $\bar{\mathcal{V}}$ that satisfies certain constraints. Recall that if $\mathcal{V}$ is 1-dimensional and over Σ_1, then $\bar{\mathcal{V}}$ has two counters the second of which tracks the letter balance.

▶ **Corollary 6.3.** *Let $\mathcal{V}$ be a 1-dimensional Büchi VASS with $L(\mathcal{V}) \subseteq \Sigma_1^\omega$ and $L(\mathcal{V}) \cap D_1 = \emptyset$. Then $L(\mathcal{V}) \not| D_1$ if and only if there exist states p, q, r with r final, and a run in $\bar{\mathcal{V}}$ as follows:*

$$(q_0,0,0) \xrightarrow{*} \overbrace{(p,x_1,y_1) \xrightarrow{*} (p,x_2,y_2)}^{\sigma_1} \xrightarrow{*} \overbrace{(q,x_3,y_3) \xrightarrow{*} (q,x_4,y_4)}^{\sigma_2}$$
$$\xrightarrow{*} \overbrace{(r,x_5,y_5) \xrightarrow{*} \underbrace{(r,x_6,y_6) \xrightarrow{*} (r,x_7,y_7)}_{\beta}}^{\alpha} \xrightarrow{*} \overbrace{(r,x_8,y_8)}^{\gamma}$$

(1) $y_3 < y_4$ *and also (a)* $x_3 \leq x_4$ *or (b)* $x_1 < x_2$ *and* $y_1 \leq y_2$
(2) $y_5 \leq y_7$
(3) $x_5 \leq x_8$
(4) *if* $y_5 = y_6$, *then* $y_5 = y_8$.

Observe that an inseparability flower in $\mathsf{KM}(\bar{\mathcal{V}})$ must carry ω in the second coordinate, meaning the letter balance is unbounded. Otherwise, it would yield an accepting run of $\bar{\mathcal{V}}$, which cannot exist because $L(\mathcal{V}) \cap D_1 = \emptyset$. If the flower has ω in the second coordinate, we can construct a *finite* run as above. The cycles σ_1 and σ_2 plus Condition 1 ensure that indeed the second coordinate becomes ω. Condition 2 is $\varphi(\alpha\beta) \geq \mathbf{0}$. Condition 3 says $\delta(\alpha\beta\gamma) \geq \mathbf{0}$. Finally, to express $\varphi(\alpha\beta\gamma) \in \mathbb{Q} \cdot \varphi(\alpha)$, note that for integers $a \in \mathbb{Q} \cdot b$ iff $b = 0$ implies $a = 0$. Condition 4 expresses that $y_6 - y_5 = 0$ implies $y_8 - y_5 = 0$.

In order to apply Corollary 6.3 for deciding $L(\mathcal{V}_1) \mid L(\mathcal{V}_2)$ for 1-dim. Büchi VASS $\mathcal{V}_1, \mathcal{V}_2$ with binary counter updates, we would like to follow the approach for the general case and use Lemma 3.4 to first construct $\mathcal{V}$ so that $L(\mathcal{V}_1) \mid L(\mathcal{V}_2)$ if and only if $L(\mathcal{V}) \mid D_1$. From $\mathcal{V}$, we would then construct the 2-dimensional Büchi VASS $\bar{\mathcal{V}}$ that tracks the letter balance, and on $\bar{\mathcal{V}}$ we would then check the conditions of Corollary 6.3. The problem is that, under binary updates, the intermediary $\mathcal{V}$ may become exponentially large. We use the fact that also $\bar{\mathcal{V}}$ has binary counters available. This allows us to directly construct a compact variant of $\bar{\mathcal{V}}$:

▶ **Lemma 6.4.** *Given 1-dim. Büchi VASS $\mathcal{V}_1, \mathcal{V}_2$ with binary updates, there is a a 1-dim. Büchi VASS $\mathcal{V}$ with $L(\mathcal{V}_1) \cap L(\mathcal{V}_2) = \emptyset$ iff $L(\mathcal{V}) \cap D_1 = \emptyset$, $L(\mathcal{V}_1) \mid L(\mathcal{V}_2)$ iff $L(\mathcal{V}) \mid D_1$, and we can construct in time polynomial in $|\mathcal{V}_1| + |\mathcal{V}_2|$ the 2-dim. Büchi VASS $\bar{\mathcal{V}}$ (binary updates).*

Detecting constrained runs in 2-VASS. It remains to check for the existence of runs in $\bar{\mathcal{V}}$ as described in Corollary 6.3, and to check whether $L(\mathcal{V}_1) \cap L(\mathcal{V}_2) = \emptyset$. Both of these problems reduce to what we call the *constrained runs problem for 2-VASS*. Recall that *Presburger arithmetic* is the first-order theory of $(\mathbb{N}, +, <, 0, 1)$. We will use the existential fragment to express conditions on counter values of VASS like the ones from Corollary 6.3. The *constrained runs problem* is the following:

Given A 2-dim. VASS $\mathcal{V}$ (with updates encoded in binary), a number $m \in \mathbb{N}$, states $q_1, \ldots, q_m$ in $\mathcal{V}$, a quantifier-free Presburger formula $\psi(x_1, y_1, \ldots, x_m, y_m)$, and $s, t \in [1, m]$, $s \leq t$.
Question Does there exist a run $(q_0, 0, 0) \xrightarrow{*} (q_1, x_1, y_1) \xrightarrow{*} \cdots \xrightarrow{*} (q_m, x_m, y_m)$ that visits a final state between (q_s, x_s, y_s) and (q_t, x_t, y_t) and satisfies $\psi(x_1, y_1, \ldots, x_m, y_m)$?

Lemma 6.4 and Corollary 6.3 imply that if $L(\mathcal{V}_1) \cap L(\mathcal{V}_2) = \emptyset$, then $L(\mathcal{V}_1) \mid L(\mathcal{V}_2)$ reduces to the constrained runs problem on $\bar{\mathcal{V}}$. Moreover, checking $L(\mathcal{V}_1) \cap L(\mathcal{V}_2) = \emptyset$ reduces via a product construction to checking emptiness of a 2-VASS. Such a 2-VASS has an accepting run iff $(q_0, 0, 0) \xrightarrow{*} (q, x, y) \xrightarrow{*} (q, x', y')$ with $(x, y) \leq (x', y')$ and q final. Hence, this problem also reduces to the constrained runs problem for 2-VASS. We thus need to show:

▶ **Proposition 6.5.** *The constrained runs problem for* 2*-VASS is solvable in* PSPACE*.*

For Proposition 6.5, we show that if there is a constrained run, then there is one with at most exponential counter values along the way. For this, we use methods from [4].

Complexity in higher dimension. We leave open two natural questions: (i) What is the complexity of regular separability for Büchi d-VASS, for each $d \geq 2$? (ii) What is the complexity of regular separability for Büchi VASS (where the dimension is part of the input)?

Given that the regular separability and the disjointness problem usually (but not always [27, 42]) coincide regarding decidability, we expect the complexity of regular separability to be PSPACE in every fixed dimension d and EXPSPACE in general. The lower bounds follow from Theorem 6.1 for fixed d and from [13] (because disjointness is EXPSPACE-complete [19, 29]). However, it is not clear how to show the upper bounds.

The clearest obstacle is that inseparability flowers involve a non-linear condition: The requirement $\varphi(\alpha\beta\gamma) \in \mathbb{Q} \cdot \varphi(\alpha)$ is not expressible in Presburger arithmetic. There are several generic results providing EXPSPACE upper bounds for detecting particular types of runs in VASS [17, 2, 3]. However, the numerical properties directly expressible there are confined to Presburger arithmetic. The only reason we could obtain the PSPACE upper bound for $d = 1$ is that the non-linear condition degenerates into a linear condition in dimension one: It is equivalent to "$\varphi(\alpha\beta\gamma) = 0$ or $\varphi(\alpha) \neq 0$".

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Vrunda Dave, and Shankara Narayanan Krishna. On the Separability Problem of String Constraints. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 16:1–16:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.16.

2 Mohamed Faouzi Atig and Peter Habermehl. On Yen's Path Logic for Petri Nets. *Int. J. Found. Comput. Sci.*, 22(4):783–799, 2011. doi:10.1142/S0129054111008428.

3 Michel Blockelet and Sylvain Schmitz. Model checking coverability graphs of vector addition systems. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 – 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2011. doi:10.1007/978-3-642-22993-0_13.

4 Michael Blondin, Matthias Englert, Alain Finkel, Stefan Göller, Christoph Haase, Ranko Lazic, Pierre McKenzie, and Patrick Totzke. The Reachability Problem for Two-Dimensional Vector Addition Systems with States. *J. ACM*, 68(5):34:1–34:43, 2021. doi:10.1145/3464794.

5 Heino Carstensen. Infinite behaviour if deterministic petri nets. In Michal Chytil, Ladislav Janiga, and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1988, MFCS'88, Carlsbad, Czechoslovakia, August 29 – September 2, 1988, Proceedings*, volume 324 of *Lecture Notes in Computer Science*, pages 210–219. Springer, 1988. doi:10.1007/BFb0017144.

6 Pierre Chambart, Alain Finkel, and Sylvain Schmitz. Forward analysis and model checking for trace bounded WSTS. *Theor. Comput. Sci.*, 637:1–29, 2016. doi:10.1016/j.tcs.2016.04.020.

7 Christian Choffrut, Flavio D'Alessandro, and Stefano Varricchio. On the separability of sparse context-free languages and of bounded rational relations. *Theor. Comput. Sci.*, 381(1-3):274–279, 2007. doi:10.1016/j.tcs.2007.04.003.

8 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Regular Separability of Parikh Automata. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 117:1–117:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.117.

9 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Separability of Reachability Sets of Vector Addition Systems. In Heribert Vollmer and Brigitte Vallée, editors, *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 24:1–24:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.24.

10 Lorenzo Clemente, Slawomir Lasota, and Radoslaw Piórkowski. Timed Games and Deterministic Separability. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 121:1–121:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.121.

11 Wojciech Czerwiński, Piotr Hofman, and Georg Zetzsche. Unboundedness problems for languages of vector addition systems. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proc. of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 119:1–119:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2018.119.

12 Wojciech Czerwinski and Slawomir Lasota. Regular separability of one counter automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005079.

13 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular Separability of Well-Structured Transition Systems. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory (CONCUR 2018)*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2018.35.

14 Wojciech Czerwinski, Wim Martens, and Tomás Masopust. Efficient Separability of Regular Languages by Subsequences and Suffixes. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2013. doi:10.1007/978-3-642-39212-2_16.

15 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A Characterization for Decidable Separability by Piecewise Testable Languages. *Discrete Mathematics and Theoretical Computer Science*, 19(4), 2017. doi:10.23638/DMTCS-19-4-1.

16 Wojciech Czerwiński and Georg Zetzsche. An Approach to Regular Separability in Vector Addition Systems. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *Proc. of the Thirty-Fifth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2020)*, pages 341–354. ACM, 2020. doi:10.1145/3373718.3394776.

17 Stéphane Demri. On selective unboundedness of VASS. *J. Comput. Syst. Sci.*, 79(5):689–713, 2013. doi:10.1016/j.jcss.2013.01.014.

18 Jacques Duparc, Olivier Finkel, and Jean-Pierre Ressayre. The wadge hierarchy of petri nets ω-languages. In Vasco Brattka, Hannes Diener, and Dieter Spreen, editors, *Logic, Computation, Hierarchies*, volume 4 of *Ontos Mathematical Logic*, pages 109–138. De Gruyter, 2014. doi:10.1515/9781614518044.109.

19 Javier Esparza. Decidability and complexity of Petri net problems – an introduction. In G. Rozenberg and W. Reisig, editors, *Lectures on Petri Nets I: Basic Models. Advances in Petri Nets*, number 1491 in Lecture Notes in Computer Science, pages 374–428, 1998.

20 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015. doi:10.1016/j.ic.2014.12.004.

21 Olivier Finkel. Borel ranks and wadge degrees of context free omega-languages. *Math. Struct. Comput. Sci.*, 16(5):813–840, 2006. doi:10.1017/S0960129506005597.

22 Olivier Finkel and Michal Skrzypczak. On the expressive power of non-deterministic and unambiguous petri nets over infinite words. *Fundam. Informaticae*, 183(3-4):243–291, 2021. doi:10.3233/FI-2021-2088.

23 Peter Habermehl. On the Complexity of the Linear-Time μ-calculus for Petri-Nets. In *ICATPN*, volume 1248 of *LNCS*, pages 102–116. Springer, 1997.

24 Christopher Hugenroth. Separating Regular Languages over Infinite Words with Respect to the Wagner Hierarchy. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 46:1–46:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.46.

25 Harry B. Hunt III. On the Decidability of Grammar Problems. *Journal of the ACM*, 29(2):429–447, 1982. doi:10.1145/322307.322317.

26 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.

27 Eryk Kopczynski. Invisible Pushdown Languages. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 867–872. ACM, 2016. doi:10.1145/2933575.2933579.

28 Jérôme Leroux and Sylvain Schmitz. Demystifying Reachability in Vector Addition Systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 56–67. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.16.

29 Richard Lipton. The reachability problem is exponential-space hard. *Yale University, Department of Computer Science, Report*, 62, 1976.

30 Tomás Masopust. Separability by piecewise testable languages is PTime-complete. *Theor. Comput. Sci.*, 711:109–114, 2018. doi:10.1016/j.tcs.2017.11.004.

31 Ernst W Mayr and Albert R Meyer. The complexity of the finite containment problem for Petri nets. *Journal of the ACM (JACM)*, 28(3):561–576, 1981.

32 Théo Pierron, Thomas Place, and Marc Zeitoun. Quantifier Alternation for Infinite Words. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures – 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2016. doi:10.1007/978-3-662-49630-5_14.

33 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating Regular Languages by Locally Testable and Locally Threshold Testable Languages. In Anil Seth and Nisheeth K. Vishnoi, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India*, volume 24 of *LIPIcs*, pages 363–375. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.FSTTCS.2013.363.

34 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14–18, 2014*, pages 75:1–75:10. ACM, 2014. doi:10.1145/2603088.2603098.

35 Thomas Place and Marc Zeitoun. Separation and the Successor Relation. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 662–675. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.662.

36 Thomas Place and Marc Zeitoun. Separating Regular Languages with First-Order Logic. *Log. Methods Comput. Sci.*, 12(1), 2016. doi:10.2168/LMCS-12(1:5)2016.

37 Thomas Place and Marc Zeitoun. Separating Without Any Ambiguity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 137:1–137:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.137.

38 Thomas Place and Marc Zeitoun. Separation and covering for group based concatenation hierarchies. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785655.

39 Andreas Podelski and Andrey Rybalchenko. A Complete Method for the Synthesis of Linear Ranking Functions. In *VMCAI*, volume 2937 of *LNCS*, pages 239–251. Springer, 2004.

40 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.

41 Thomas G. Szymanski and John H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2), 1976.

42 Ramanathan S. Thinniyam and Georg Zetzsche. Regular Separability and Intersection Emptiness are Independent Problems. In *Proc. of the 39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *LIPIcs*, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

43 Rüdiger Valk. Infinite behaviour of petri nets. *Theoretical computer science*, 25(3):311–341, 1983.

44 Georg Zetzsche. Separability by piecewise testable languages and downward closures beyond subwords. In Anuj Dawar and Erich Grädel, editors, *Proc. of the Thirty-Third Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 929–938. ACM, 2018. doi:10.1145/3209108.3209201.

Approximating Highly Inapproximable Problems on Graphs of Bounded Twin-Width

Pierre Bergé
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

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

Hugues Déprés
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Rémi Watrigant
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

For any $\varepsilon > 0$, we give a polynomial-time n^{ε}-approximation algorithm for Max Independent Set in graphs of bounded twin-width given with an $O(1)$-sequence. This result is derived from the following time-approximation trade-off: We establish an $O(1)^{2^q-1}$-approximation algorithm running in time $\exp(O_q(n^{2^{-q}}))$, for every integer $q \geqslant 0$. Guided by the same framework, we obtain similar approximation algorithms for Min Coloring and Max Induced Matching. In general graphs, all these problems are known to be highly inapproximable: for any $\varepsilon > 0$, a polynomial-time $n^{1-\varepsilon}$-approximation for any of them would imply that P=NP [Håstad, FOCS '96; Zuckerman, ToC '07; Chalermsook et al., SODA '13]. We generalize the algorithms for Max Independent Set and Max Induced Matching to the independent (induced) packing of any fixed connected graph H.

In contrast, we show that such approximation guarantees on graphs of bounded twin-width given with an $O(1)$-sequence are very unlikely for Min Independent Dominating Set, and somewhat unlikely for Longest Path and Longest Induced Path. Regarding the existence of better approximation algorithms, there is a (very) light evidence that the obtained approximation factor of n^{ε} for Max Independent Set may be best possible. This is the first in-depth study of the approximability of problems in graphs of bounded twin-width. Prior to this paper, essentially the only such result was a polynomial-time $O(1)$-approximation algorithm for Min Dominating Set [Bonnet et al., ICALP '21].

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

Keywords and phrases Approximation algorithms, bounded twin-width

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.10

Related Version *Full Version*: https://arxiv.org/abs/2207.07708

Funding This work was supported by the ANR projects TWIN-WIDTH (ANR-21-CE48-0014) and Digraphs (ANR-19-CE48-0013).

Acknowledgements We thank Colin Geniet, Eunjung Kim, and Stéphan Thomassé for useful discussions.

1 Introduction

Twin-width is a graph parameter introduced by Bonnet, Kim, Thomassé, and Watrigant [7]. Its definition involves the notions of *trigraphs* and of *contraction sequences*. A *trigraph* is a graph with two types of edges: black (regular) edges and red (error) edges. A (vertex) *contraction* consists of merging two (non-necessarily adjacent) vertices, say, u, v into a vertex w,

© Pierre Bergé, Édouard Bonnet, Hugues Déprés, and Rémi Watrigant;
licensed under Creative Commons License CC-BY 4.0

replacing an edge uz (resp. vz) by wz in the following way: we keep wz black if and only if uz and vz were previously black edges. The other edges incident to w become red (if not already), and the rest of the trigraph remains the same. A *contraction sequence* of an n-vertex[1] graph G is a sequence of trigraphs $G = G_n, \ldots, G_1 = K_1$ such that G_i is obtained from G_{i+1} by performing one contraction. A *d-sequence* is a contraction sequence in which every vertex of every trigraph has at most d red edges incident to it. The *twin-width* of G, denoted by $\mathrm{tww}(G)$, is then the minimum integer d such that G admits a d-sequence. Figure 1 gives an example of a graph with a 2-sequence, i.e., of twin-width at most 2. Twin-width can be naturally extended to matrices (with unordered [7] or ordered [6] row and column sets) over a finite alphabet, and thus to binary structures.

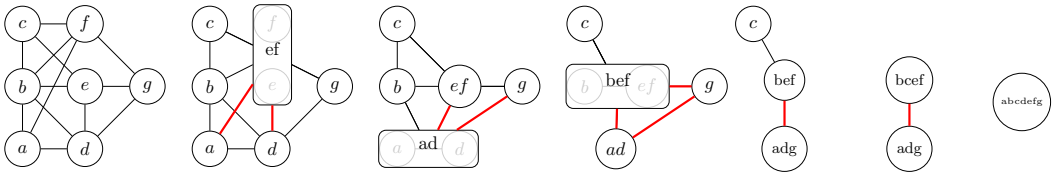


Figure 1 A 2-sequence witnessing that the initial graph has twin-width at most 2.

An equivalent viewpoint that will be somewhat more convenient is to consider a d-sequence as a sequence of partitions $\mathcal{P}_n := \{\{v\} \ : \ v \in V(G)\}, \mathcal{P}_{n-1}, \ldots, \mathcal{P}_1 := \{V(G)\}$ of $V(G)$, such that for every integer $1 \leqslant i \leqslant n-1$, $\mathcal{P}_i$ has i parts and is obtained by merging two parts of $\mathcal{P}_{i+1}$ into one. Now the *red degree* of a part $P \in \mathcal{P}_i$ is the number of other parts $Q \in \mathcal{P}_i$ such that there is in G at least one edge and at least one non-edge between P and Q. A d-sequence is such that no part of no partition of the sequence has red degree more than d. In that case the *maximum red degree* of each partition is at most d. And we similarly get the twin-width of G as the minimum integer d such that G admits a (partition) d-sequence. The *quotient trigraph* $G/\mathcal{P}_i$ is the trigraph G_i, if the (contraction) d-sequence $G_n, \ldots, G_1$ and the (partition) d-sequence $\mathcal{P}_n, \ldots, \mathcal{P}_1$ correspond.

Classes of binary structures with bounded twin-width include graph classes with bounded treewidth, and more generally bounded clique-width, proper minor-closed classes, posets with antichains of bounded size, strict subclasses of permutation graphs, as well as $\Omega(\log n)$-subdivisions of n-vertex graphs [7], and some classes of (bounded-degree) expanders [4]. A notable variety of geometrically defined graph classes have bounded twin-width such as map graphs, bounded-degree string graphs [7], classes with bounded queue number or bounded stack number [4], segment graphs with no $K_{t,t}$ subgraph, visibility graphs of 1.5D terrains without large half-graphs, visibility graphs of simple polygons without large independent sets [3].

For every class $\mathcal{C}$ mentioned so far, $O(1)$-sequences can be computed in polynomial time[2] on members of $\mathcal{C}$. For classes of binary structures including a binary relation interpreted as a linear order on the domain (called *ordered binary structures*), there is a fixed-parameter approximation algorithm for twin-width [6]. More precisely, given a graph G and an integer k, there are computable functions f and g such that one can output an $f(k)$-sequence of G or correctly report that $\mathrm{tww}(G) > k$ in time $g(k)n^{O(1)}$. Such an approximation algorithm

[1] In this introduction, we might implicitly use n to denote the number of vertices, and m, the number of edges of the graph at hand.

[2] Admittedly, for the geometric classes, a representation is (at least partially) needed.

is currently missing for classes of general (not necessarily ordered) binary structures, and in particular for the class of all graphs. We also observe that deciding if the twin-width of a graph is at most 4 is an NP-complete task [2].

We will therefore assume that the input graph is given with a d-sequence, and treat d as a constant (or that the input comes from any of the above-mentioned classes). Thus far, this is the adopted setting when designing faster algorithms on bounded twin-width graphs [7, 5, 26, 23, 15]. From the inception of twin-width [7] –actually already from the seminal work of Guillemot and Marx [17]– it was clear that structures wherein this invariant is bounded may *often* allow the design of parameterized algorithms. More concretely, it was shown [7] that, on graphs G given with a d-sequence, model checking a first-order sentence φ is fixed-parameter tractable –it can be solved in time $f(d, \varphi) \cdot n$–, the special cases of, say, k-Independent Set or k-Dominating Set admit single-exponential parameterized algorithms [5], an effective data structure almost linear in n can support constant-time edge queries [26], the triangles of G can be counted in time $O(d^2 n + m)$ [23].

So far, however, the connection between *having bounded twin-width* and *enjoying enhanced approximation factors* was tenuous. The only such result concerned Min Dominating Set, known to be inapproximable in polynomial-time within factor $(1-o(1)) \ln n$ unless P=NP [12], but yet admits a constant-approximation on graphs of bounded twin-width given with an $O(1)$-sequence [5]. We start filling this gap by designing approximation algorithms on graphs of bounded twin-width given with an $O(1)$-sequence for notably Max Independent Set (MIS, for short), Max Induced Matching, and Coloring. Getting better approximation algorithms for MIS and Coloring in that particular scenario was raised as an open problem [5]. Before we describe our results and elaborate on the developed techniques, let us briefly present the notorious inapproximability of these problems in general graphs.

MIS and Coloring are NP-hard [16], and very inapproximable: for every $\varepsilon > 0$, it is NP-hard to approximate these problems within ratio $n^{1-\varepsilon}$ [19, 27]. The same was shown to hold for Max Induced Matching [9]. Besides, there is only little room to improve over the brute-force algorithm in $2^{O(n)}$: Unless the Exponential Time Hypothesis[3] [21] (ETH) fails, no algorithm can solve MIS in time $2^{o(n)}$ [22] (nor the other two problems). For any r (possibly a function of n) WMIS can be r-approximated in time $2^{O(n/r)}$ [11, 8]. Bansal et al. [1] essentially shaved a $\log^2 r$ factor to the latter exponent. It is known though that polynomial shavings are unlikely. Chalermsook et al. [10] showed that, for any $\varepsilon > 0$ and sufficiently large r (again r can be function of n), an r-approximation for MIS and Max Induced Matching cannot take time $2^{O(n^{1-\varepsilon}/r^{1+\varepsilon})}$, unless the ETH fails. For instance, investing time $2^{O(\sqrt{n})}$, one cannot hope for significantly better than a $\sqrt{n}$-approximation.

Contributions and techniques

Our starting point is a constant-approximation algorithm for MIS running in time $2^{O(\sqrt{n})}$ when presented with an $O(1)$-sequence, which is very unlikely to hold in general graphs by the result of Chalermsook et al. [10].

▶ **Theorem 1.** *On n-vertex graphs given with a d-sequence Max Independent Set can be $O_d(1)$-approximated in time $2^{O_d(\sqrt{n})}$.*

Our algorithm builds upon the functional equivalence between twin-width and the so-called *versatile twin-width* [4]. We defer the reader to Section 2 for a formal definition of versatile twin-width. For our purpose, one only needs to know the following useful consequence of that

[3] That is, the assumption that there is a $\delta > 0$ such that n-variable 3-SAT cannot be solved in time δ^n.

equivalence. From a d'-sequence of G, we can compute in polynomial time another partition sequence $\mathcal{P}_n, \ldots, \mathcal{P}_1$ of G of width $d := f(d')$, for some computable function f, such that for every integer $1 \leqslant i \leqslant n$, all the i parts of $\mathcal{P}_i$ have size at most $d \cdot \frac{n}{i}$. Even if some parts of P_i can be very small, this partition is balanced in the sense that no part can be larger than d times the part size in a perfectly balanced partition. Of importance to us is $\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$ when the number of parts ($\lfloor\sqrt{n}\rfloor$) and the size of a larger part in the partition (at most $d\frac{n}{\lfloor\sqrt{n}\rfloor} \approx d\sqrt{n}$) are somewhat level.

We can then properly color the red graph (made by the red edges on the vertex set $\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$) with $d+1$ colors. Any color class X is a subset of parts of $\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$ such that between two parts there are either all edges (black edge) or no edge at all (non-edge). In graph-theoretic terms, the subgraph G_X of G induced by all the vertices of all the parts of X has a simple modular decomposition: a partition of at most $\sqrt{n}$ modules each of size at most $d\sqrt{n}$. It is thus routine to compute a largest independent set of G_X essentially in time exponential in the maximum between the number of modules and the maximum size of a module, that is, in at most $d\sqrt{n}$. As one color class X^* contains more than a $\frac{1}{d+1}$ fraction of the optimum, we get our $d+1$-approximation when computing a largest independent set of G_{X^*}. Figure 2 on page 10 serves as a visual summary of what we described so far.

The next step is to substitute exact exponential algorithms on induced subgraphs of size $O_d(\sqrt{n})$ by recursive calls of our approximation algorithm. Following this inductive process at depth $q = 2, 3, 4, \ldots$, we degrade the approximation ratio to $(d+1)^3, (d+1)^7, (d+1)^{15}$, etc. but meanwhile we boost the running time to $2^{O_d(n^{1/4})}, 2^{O_d(n^{1/8})}, 2^{O_d(n^{1/16})}$, etc. In effect we show by induction that:

▶ **Theorem 2.** *On n-vertex graphs given with a d-sequence Max Independent Set has an $O_d(1)^{2^q-1}$-approximation algorithm running in time $2^{O_{d,q}(n^{2^{-q}})}$, for every integer $q \geqslant 0$.*

The following polynomial-time algorithm is a corollary of Theorem 2 choosing $q = O_{d,\varepsilon}(\log\log n)$.

▶ **Theorem 3.** *For every $\varepsilon > 0$, Max Independent Set can be n^ε-approximated in polynomial-time $O_{d,\varepsilon}(1) \cdot \log^{O_d(1)} n \cdot n^{O(1)}$ on n-vertex graphs given with a d-sequence.*

Note that the exponent of the polynomial factor is an absolute constant (not depending on d nor on ε).

We then apply our framework to Coloring and Max Induced Matching.

▶ **Theorem 4.** *For every $\varepsilon > 0$, Coloring and Max Induced Matching admit polynomial-time n^ε-approximation algorithms on n-vertex graphs of bounded twin-width given with an $O(1)$-sequence.*

The main additional difficulty for Coloring is that one cannot satisfactorily solve/approximate that problem on a modular decomposition by simply coloring its modules and its quotient graph. One needs to tackle a more general problem called Set Coloring. Fortunately this generalization is the fixed point we are looking for: approximating Set Coloring can be done in our framework by mere recursive calls (to itself).

For Max Induced Matching, we face a new kind of obstacle. It can be the case that no decent solution is contained in any color class X –in the chosen $d+1$-coloring of the red graph $G/\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$. For instance, it is possible that any such color class X induces in G an edgeless graph, while very large induced matchings exist with endpoints in two distinct color classes. We thus need to also find large induced matchings within the black edges and within the red edges of $G/\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$. This leads to a more intricate strategy intertwining the coloring

of bounded-degree graphs (specifically the red graph and the square of its line graph) and recursive calls to induced subgraphs of G, and to special induced subgraphs of the total graph (i.e., made by both the red and black edges) of $G/\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$.

We then explore the limits of our results and framework in terms of amenable problems. We give the following technical generalization to the approximation algorithms for MIS and Max Induced Matching.

▶ **Theorem 5.** *For every connected graph H and $\varepsilon > 0$, Mutually Induced H-packing admits a polynomial-time n^{ε}-approximation algorithms on n-vertex graphs of bounded twin-width given with an $O(1)$-sequence.*

In this problem, one seeks for a largest induced subgraph that consists of a disjoint union of copies of H. All the previous technical issues are here combined. We try all the possibilities of batching the vertices of H into at most $|V(H)|$ parts of $G/\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$, based on the trigraph that these parts define. For instance with $H = K_2$ (an edge), i.e., the case of Max Induced Matching, the three possible trigraphs are the 1-vertex trigraph, two vertices linked by a red edge, and two vertices linked by a black edge. In the general case, the problem generalization is quite delicate to find. We have to keep some partitions of $V(G)$ and $V(H)$ to enforce that the copies of H in G follow a pattern that the algorithm committed to higher up in the recursion tree, and a weight function on $|V(H)|$-tuples of vertices of G, not to forget how many mutually induced copies of H can be packed *within* these vertices. The other novelty is that some recursive calls are on induced subgraphs of the total graph of $G/\mathcal{P}_{\lfloor\sqrt{n}\rfloor}$ that are *not* induced subgraphs of G. Fortunately, these graphs keep the same bound of versatile twin-width, and thus our framework allows it.

Defining, for a family of graphs $\mathcal{H}$, Mutually Induced $\mathcal{H}$-packing as the same problem where the connected components of the induced subgraph should all be in $\mathcal{H}$, we get a similar approximation factor when $\mathcal{H}$ is a finite set of connected graphs. (Note that Mutually Induced H-packing is sometimes called Independent Induced H-Packing.) In particular, we can similarly approximate Independent H-Packing, which is the same problem but the copies of H need not be induced. (Our approximation algorithms could extend to other H-packing variants without the independence requirement, but these problems can straightforwardly be $O(1)$-approximated in general graphs.)

We can handle some cases when $\mathcal{H}$ is infinite, too. For instance, by slightly adapting the case of MIS, we can get an n^{ε}-approximation when $\mathcal{H}$ is the set of all cliques. We also show the following result, also expressible as Mutually Induced $\mathcal{H}$-packing for $\mathcal{H}$ the set of all trees or the set all stars.

▶ **Theorem 6.** *For every $\varepsilon > 0$, finding the induced (star) forest with the most edges admits a polynomial-time n^{ε}-approximation algorithms on n-vertex graphs of bounded twin-width given with an $O(1)$-sequence.*

As we already mentioned, our framework is exclusively useful for problems that are very inapproximable in general graphs; at least for which an n^{ε}-approximation algorithm is not known for every $\varepsilon > 0$. Are there natural such problems that cannot be approximated better in graphs of bounded twin-width? We answer this question positively with the example of Min Independent Dominating Set.

▶ **Theorem 7.** *For every $\varepsilon > 0$, Min Independent Dominating Set does not admit an $n^{1-\varepsilon}$-approximation algorithm in n-vertex graphs given with an $O(1)$-sequence, unless P=NP.*

The reduction is the same as the one for general graphs [18], but performed from a planar variant of 3-SAT. The obtained instances are not planar but can be contracted to planar trigraphs, hence overall have bounded twin-width.

Finally the case of Longest Path and Longest Induced Path is interesting. The best approximation factor for the former [14] is worse than $n^{0.99}$, while the latter is known to have the same inapproximability as MIS [24]. However an n^ε-approximation algorithm (for every $\varepsilon > 0$) is not excluded for Longest Path. We show that the property of bounded twin-width is unlikely to help for these two problems, as it would lead to better approximation algorithms for Longest Path in general graphs. This is mainly because subdividing at least $2 \log n$ times every edge of any n-vertex graph gives a graph with twin-width at most 4 [2].

▶ **Theorem 8.** *For any $r = \omega(1)$, an r-approximation for Longest Induced Path or Longest Path on graphs given with an $O(1)$-sequence would imply a $(1 + o(1))r$-approximation for Longest Path in general graphs.*

In turn, this can be used to exhibit a family $\mathcal{H}$ with an infinite antichain for the *induced subgraph* relation such that Mutually Induced $\mathcal{H}$-packing is *hard* to n^ε-approximate on graphs of bounded twin-width. The family $\mathcal{H}$ is simply the set of all paths terminated by triangles at both ends.

▶ **Theorem 9.** *There is an infinite family $\mathcal{H}$ of connected graphs such that if for every $\varepsilon > 0$, Mutually Induced $\mathcal{H}$-packing admits an n^ε-approximation algorithm on n-vertex graphs given with an $O(1)$-sequence, then so does Longest Path on general graphs.*

Table 1 summarizes our results and hints at future work.

Table 1 Approximability status of graph problems in general graphs and in graphs of bounded twin-width given with an $O(1)$-sequence. Everywhere "ε" should be read as "$\forall \varepsilon > 0$". Our results are enclosed by boxes. "Longest Path-hard" means that getting an r-approximation would yield essentially the same ratio for Longest Path in general graphs. The other lower bounds are under standard complexity-theoretic assumptions, mostly P≠NP. Not to clutter the table, we do not put the references, which can all be found in the paper.

Problem name	lower bound general graphs	upper bound bounded tww	lower bound bounded tww
Max Independent Set	$n^{1-\varepsilon}$	$\boxed{n^\varepsilon}$	?, self-improvement
Coloring	$n^{1-\varepsilon}$	$\boxed{n^\varepsilon}$	$4/3 - \varepsilon$
Max Induced Matching	$n^{1-\varepsilon}$	$\boxed{n^\varepsilon}$	?
Mut. Ind. H-Packing	$n^{1-\varepsilon}$	$\boxed{n^\varepsilon \text{ } (H \text{ connected})}$	?
Mut. Ind. $\mathcal{H}$-Packing	$n^{1-\varepsilon}$	$\boxed{n^\varepsilon \text{ for some } \mathcal{H}}$	$\boxed{\text{Longest Path-hard}}$
Min Ind. Dom. Set	$n^{1-\varepsilon}$	$n/\mathsf{polylog}(n)$	$\boxed{n^{1-\varepsilon}}$
Longest Path	$2^{\log^{1-\varepsilon} n}$	$n/\exp(\Omega(\sqrt{\log n}))$	$\boxed{\text{Longest Path-hard}}$
Longest Induced Path	$n^{1-\varepsilon}$	$n/\mathsf{polylog}(n)$	$\boxed{\text{Longest Path-hard}}$
Min Dominating Set	$(1-\varepsilon)\ln n$	$O(1)$	?

For the main highly inapproximable graph problems, we either obtain an n^ε-approximation algorithm on graphs of bounded twin-width given with an $O(1)$-sequence, or a conditional obstruction to such an algorithm. In the former case, can we improve further the approximation factor? The next theorem was observed using the self-improvement reduction of Feige et al. [13], which preserves the twin-width bound. This reduction consists of going from a graph G to the lexicographic product $G[G]$, where every vertex of G is replaced by a module inducing a copy of G (and iterating this trick).

▶ **Theorem 10** ([5]). *Let $r : \mathbb{N} \to \mathbb{R}$ be any non-decreasing function such that for every $\varepsilon > 0$, $r(n) = o(n^\varepsilon)$. If Max Independent Set admits an $r(n)$-approximation algorithm on n-vertex graphs of bounded twin-width given with an $O(1)$-sequence, then it further admits an $r(n)^\varepsilon$-approximation.*

To our knowledge, the application of the self-improvement trick is always to strengthen a lower bound, and never to effortlessly obtain a better approximation factor. Therefore, we may take Theorem 10 as a weak indication that our approximation ratio is best possible. Still, not even a polynomial-time approximation scheme (PTAS) is ruled out for MIS (nor for Max Induced Matching, Min Dominating Set, etc.) and we would like to see better approximation algorithms. For Coloring, as was previously observed [5], a PTAS is ruled out by the NP-hardness of deciding if a planar graph is 3-colorable or 4-chromatic, since planar graphs have twin-width at most 9 and a 9-sequence can be found in linear time [20].

Due to space restrictions, only Theorems 1–3 and half of Theorem 4 are presented in the short version of the paper. All other results as well as some deferred proofs, marked with a $\star$, can be found in the long version, in appendix.

2 Preliminaries

For i and j two integers, we denote by $[i, j]$ the set of integers that are at least i and at most j. For every integer i, $[i]$ is a shorthand for $[1, i]$.

2.1 The contraction and partition viewpoints of twin-width

A *trigraph* G has vertex set $V(G)$, black edge set $E(G)$, red edge set $R(G)$ such that $E(G) \cap R(G) = \emptyset$ (and $E(G), R(G) \subseteq \binom{V(G)}{2}$). A *contraction* in a trigraph G replaces a pair of (non-necessarily adjacent) vertices $u, v \in V(G)$ by one vertex w that is linked to $G - \{u, v\}$ in the following way to form a new trigraph G'. For every $z \in V(G) \setminus \{u, v\}$, $wz \in E(G')$ whenever $uz, vz \in E(G)$, $wz \notin E(G') \cup R(G')$ whenever $uz, vz \notin E(G) \cup R(G)$, and $wz \in R(G')$, otherwise. The *red graph* $(V(G), R(G))$ will be denoted by $\mathcal{R}(G)$. We denote by $\mathcal{T}(G)$ the *total graph* of G defined as $(V(G), E(G) \cup R(G))$. An *induced subtrigraph* of a trigraph G is obtained by removing vertices (but no edges) to G, analogously to induced subgraphs. A partial contraction sequence of an n-vertex (tri)graph G (to a trigraph H) is a sequence of trigraphs $G = G_n, \cdots, G_t = H$ for some $t \in [n]$ such that G_i is obtained from G_{i+1} by performing one contraction. A (complete) contraction sequence is such that $t = 1$, that is, H is the 1-vertex trigraph. A *d-sequence* $\mathcal{S}$ of G is a contraction sequence of G in which the red graph of every trigraph of $\mathcal{S}$ has maximum degree at most d.

Assume that there is a partial contraction sequence from a (tri)graph G to a trigraph H. If u is a vertex of H, then $u(G) \subseteq V(G)$ denotes the set of vertices eventually contracted into u in H. We denote by $\mathcal{P}(H)$ the partition $\{u(G) : u \in V(H)\}$ of $V(G)$. If G is clear from the context, we may refer to a *part* of H as any set in $\{u(G) : u \in V(H)\}$. We will mostly see d-sequences as sequences of partitions, that is, $\mathcal{P}_n, \ldots, \mathcal{P}_t$ with $\mathcal{P}_i := \{u(G) : u \in V(G_i)\}$ when $G_n, \ldots, G_t$ is a partial (contraction) d-sequence.

Given a graph G and a partition $\mathcal{P}$ of $V(G)$, the *quotient graph* of G with respect to $\mathcal{P}$ is the graph with vertex set $\mathcal{P}$, where PP' is an edge if there is $u \in P$ and $v \in P'$ such that $uv \in E(G)$. Given a (tri)graph G and a partition $\mathcal{P}$ of $V(G)$, the *quotient trigraph* $G/\mathcal{P}$ is the trigraph with vertex set $\mathcal{P}$, where PP' is a black edge if these two parts are fully adjacent – for every $u \in P$ and every $v \in P'$, $uv \in E(G)$ –, and a red edge if either there is $u \in P$ and $v \in P'$ such that $uv \in R(G)$, or there is $u_1, u_2 \in P$ and $v_1, v_2 \in P'$ such that $u_1v_1 \in E(G)$ and $u_2v_2 \notin E(G)$.

A trigraph H is a *cleanup* of another trigraph G if $V(H) = V(G)$, $R(H) \subseteq R(G)$, and $E(G) \subseteq E(H) \subseteq E(G) \cup R(G)$. That is, H is obtained from G by turning some of its red edges into black edges or non-edges. We further say that H is *full cleanup* of G if H has no red edge, and thus, is considered as a graph. Note that the total graph $\mathcal{T}(G)$ and the *black graph* $(V(G), E(G))$ of a trigraph G are extreme examples of full cleanups of G.

2.2 Balanced partition sequences

It was shown that twin-width and *versatile* twin-width (we will not need a definition here; see long version) are functionally equivalent [4]. The relevant consequence for our purposes is that every graph G with a d'-sequence admits a *balanced* d-sequence, where $d = h(d')$ depends only on d', i.e., one for which the partitions $\mathcal{P}_n, \dots, \mathcal{P}_1$ are such that for every $i \in [n]$ and $P \in \mathcal{P}_i$, $|P| \leqslant d \cdot \frac{n}{i}$. As we will resort to recursion on induced subtrigraphs and quotient trigraphs, we need to keep more information on those subinstances that the mere fact that they have twin-width at most d (otherwise the twin-width bound could quickly diverge).

This will be done by opening up the proof in [4], and handling divided $0, 1, r$-matrices with some specific properties. Thus we need to recall the relevant definitions.

Given two partitions $\mathcal{P}, \mathcal{P}'$ of the same set, we say that $\mathcal{P}'$ is a *coarsening* of $\mathcal{P}$ if every part of $\mathcal{P}$ is contained in a part of $\mathcal{P}'$, and $\mathcal{P}, \mathcal{P}'$ are distinct. Given a matrix M, we call *row division* (resp. *column division*) a partition of the rows (resp. columns) of M into parts of consecutive rows (resp. columns). A (k, ℓ)*-division*, or simply *division*, of a matrix M is a pair $(\mathcal{R} = \{R_1, \dots, R_k\}, \mathcal{C} = \{C_1, \dots, C_\ell\})$ where $\mathcal{R}$ is a row division and $\mathcal{C}$ is a column division. In a matrix division $(\mathcal{R}, \mathcal{C})$, each part $R \in \mathcal{R}$ is called a *row part*, and each part $C \in \mathcal{C}$ is called a *column part*. Given a subset R of rows and a subset C of columns in a matrix M, the *zone* $M[R, C]$ denotes the submatrix of all entries of M at the intersection between a row of R and a column of C. A *zone* of a matrix partitioned by $(\mathcal{R}, \mathcal{C}) = (\{R_1, \dots, R_k\}, \{C_1, \dots, C_\ell\})$ is any $M[R_i, C_j]$ for $i \in [k]$ and $j \in [\ell]$. A zone is *constant* if all its entries are identical, *horizontal* if all its columns are equal, and *vertical* if all its rows are equal. A *0,1-corner* is a 2×2 $0, 1$-matrix which is neither horizontal nor vertical.

Unsurprisingly, $0, 1, r$*-matrices* are such that each entry is in $\{0, 1, r\}$ where r is an error symbol that should be understood as a red edge. A *neat division* of a $0, 1, r$-matrix is a division for which every zone either contains only r entries or contains no r entry and is horizontal or vertical (or both, i.e., constant). Zones filled with r entries are called *mixed*. A *neatly divided matrix* is a pair $(M, (\mathcal{R}, \mathcal{C}))$ where M is a $0, 1, r$-matrix and $(\mathcal{R}, \mathcal{C})$ is a neat division of M. A t*-mixed minor* in a neatly divided matrix is a (t, t)-division which coarsens the neat subdivision, and contains in each of its t^2 zones at least one mixed zone (i.e., filled with r entries) or a 0,1-corner. A neatly divided matrix is said t*-mixed free* if it does not admit a t-mixed minor.

A *mixed cut of a row part* $R \in \mathcal{R}$ of a neatly divided matrix $(M, (\mathcal{R}, \mathcal{C} = \{C_1, C_2, \dots\}))$ is an index i such that both $M[R, C_i]$ and $M[R, C_{i+1}]$ are not mixed, and there is a $0, 1$-corner in the 2-by-$|R|$ zone defined by the last column of C_i, the first column of C_{i+1}, and R. The *mixed value of a row part* $R \in \mathcal{R}$ of a neatly divided matrix $(M, (\mathcal{R}, \mathcal{C} = \{C_1, C_2, \dots\}))$ is the number of mixed zones $M[R, C_j]$ plus the number of mixed cuts of R. We similarly define the mixed value of a column part $C \in \mathcal{C}$. The *mixed value of a neat division* of a $0, 1, r$-matrix is the maximum of the mixed values taken over every part. The *part size* of a division $(\mathcal{R}, \mathcal{C})$ is defined as $\max(\max_{R \in \mathcal{R}} |R|, \max_{C \in \mathcal{C}} |C|)$. A division is *symmetric* if the largest row index of each row part and the largest column index of each column part define the same set of integers. We call *symmetric fusion* of a symmetric division the contraction of two consecutive parts in $\mathcal{C}$ and of the two corresponding parts in $\mathcal{R}$. A symmetric fusion on a symmetric division yields another symmetric division. A matrix $A := (a_{i,j})_{i,j}$ is said *symmetric* in the usual sense, namely, for every entry $a_{i,j}$ of A, $a_{i,j} = a_{j,i}$.

In what follows, we set $c_d := 8/3(d+1)^2 2^{4d}$.

▶ **Definition 11.** *Let $\mathcal{M}_{n,d}$ be the class of the neatly divided $n \times n$ symmetric $0,1,r$-matrices $(M,(\mathcal{R},\mathcal{C}))$, such that $(\mathcal{R},\mathcal{C})$ is symmetric and has:*
- *mixed value at most $4c_d$,*
- *part size at most 2^{4c_d+2}, and*
- *no d-mixed minor.*

We show in the long version the next couple of key lemmas.

▶ **Lemma 12** (⋆). *Let d be a natural, $s := 2^{4c_d+4}$, and $d' := c_d \cdot 2^{4c_d+4}$. Given an n-vertex graph G with a d-sequence, one can compute in time $n^{O(1)}$ a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_{\lfloor\sqrt{n}\rfloor}\}$ of $V(G)$ satisfying*
- *for every integer $1 \leqslant i \leqslant \lfloor\sqrt{n}\rfloor$, $|P_i| \leqslant s\sqrt{n} \leqslant d'\sqrt{n}$, and*
- *the red graph of $G/\mathcal{P}$ has maximum degree at most d'.*

A neatly divided matrix $(M,(\mathcal{R},\mathcal{C}))$ is said *conform* to a trigraph G if M is the adjacency matrix of a trigraph G' such that G is a cleanup of G'.

▶ **Lemma 13** (⋆). *Let $\hat{d}$ be a natural, $d = 2\hat{d}+2$, and set $s := 2^{4c_d+4}$, and $d' := c_d \cdot 2^{4c_d+4}$. Given an n-vertex graph G with a $\hat{d}$-sequence, or an n-vertex trigraph G with a neatly divided matrix $(M,(\mathcal{R},\mathcal{C})) \in \mathcal{M}_{n,d}$ such that M is conform to G, one can compute in time $n^{O(1)}$ a partition $\mathcal{P} = \{P_1, P_2, \ldots, P_{\lfloor\sqrt{n}\rfloor}\}$ of $V(G)$ with maximum red degree at most d' satisfying that, for every integer $1 \leqslant i \leqslant \lfloor\sqrt{n}\rfloor$, $|P_i| \leqslant s\sqrt{n} \leqslant d'\sqrt{n}$, and for any trigraph H that is*
- *a cleanup of an induced subtrigraph of $G/\mathcal{P}$, or*
- *an induced subtrigraph $G[\bigcup_{i\in J\subseteq[\lfloor\sqrt{n}\rfloor]} P_i]$,*

a neatly divided matrix $(M',(\mathcal{R}',\mathcal{C}')) \in \mathcal{M}_{|V(H)|,d}$ conform to H can be computed in time $n^{O(1)}$.

3 Approximation algorithms for Max Independent Set

We naturally start our study with Max Independent Set, a central problem that is very inapproximable [19, 27], and yet constitutes the textbook example of our approach.

3.1 Subexponential-time constant-approximation algorithm

We present a subexponential-time $O_d(1)$-approximation for WMIS on graphs given with a d-sequence, which we recall, is unlikely to exist in general graphs [10].

▶ **Lemma 14.** *Let d' be a natural, $s := 2^{4c_{d'}+4}$, and $d := c_{d'} \cdot 2^{4c_{d'}+4}$. Assume n-vertex inputs G, vertex-weighted by w, are given with a d'-sequence. Weighted Max Independent Set can be $(d+1)$-approximated in time $2^{O_d(\sqrt{n})}$ on these inputs.*

Proof. By Lemma 12, we compute in polynomial time a partition $\mathcal{P} = \{P_1, \ldots, P_{\lfloor\sqrt{n}\rfloor}\}$ of $V(G)$ whose parts have size at most $s\sqrt{n}$ and such that $\mathcal{R}(G/\mathcal{P})$ has maximum degree at most d.

For every integer $1 \leqslant i \leqslant \lfloor\sqrt{n}\rfloor$, we compute a heaviest independent set in $G[P_i]$, say S_i. Even with an exhaustive algorithm, this takes time $\sqrt{n} \cdot s^2 n \cdot 2^{s\sqrt{n}} = 2^{O_d(\sqrt{n})}$. We then $(d+1)$-color (in linear time) $\mathcal{R}(G/\mathcal{P})$, which is possible since this graph has maximum degree at most d. This defines a coarsening of $\mathcal{P}$ in $d+1$ parts $\mathcal{Q} = \{C_1, \ldots, C_{d+1}\}$. Thus, $\mathcal{Q}$ is a partition of $V(G)$ such that C_j consists of all the parts $P_i \in \mathcal{P}$ receiving color j in the $(d+1)$-coloring of $\mathcal{R}(G/\mathcal{P})$.

For every $j \in [d+1]$, let H_j be the *graph* $(G/\mathcal{P})[C_j]$[4] vertex-weighted by $P_i \subseteq C_j \mapsto w(S_i)$. Note that $(G/\mathcal{P})[C_j]$ can indeed be assimilated to a graph, since it has, by design, no red edge. We compute a heaviest independent set in H_j, say R_j. This takes time $(d+1)\cdot n \cdot 2^{\sqrt{n}} = 2^{O_d(\sqrt{n})}$. We output $\bigcup_{P_i \subseteq R_j} S_i$ for the index $j \in [d+1]$ maximizing $\sum_{P_i \subseteq R_j} w(S_i)$.

This finishes the description of the algorithm. We already argued that its running time is $2^{O_d(\sqrt{n})}$. We shall justify that it does output an independent set of weight at least a $\frac{1}{d+1}$ fraction of the optimum $\alpha(G)$. Let I be the output of the algorithm.

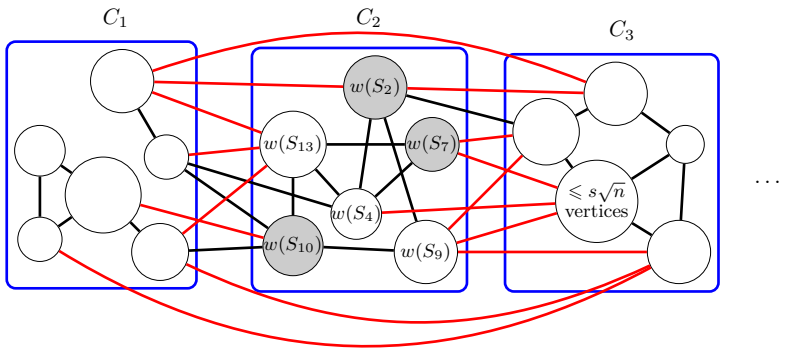


Figure 2 The trigraph $G/\mathcal{P}$ with its $\lfloor\sqrt{n}\rfloor$ vertices, each corresponding to a subset of at most $s\sqrt{n}$ vertices of G. The weights $w(S_i)$ of heaviest independent sets S_i of $G[P_i]$ for each part P_i of the color class C_2 of the $d+1$-coloring of $\mathcal{R}(G/\mathcal{P})$. A heaviest independent set in the so-weighted $(G/\mathcal{P})[C_2]$ (shaded) corresponds to an optimum solution in $G[\bigcup_{P_i \subseteq C_2} P_i]$. One of these $d+1$ independent sets is a $d+1$-approximation.

***I* is indeed an independent set.** For any $j \in [d+1]$, consider two vertices $x, y \in \bigcup_{P_i \subseteq R_j} S_i$. If $\{x, y\} \in S_i$ for some i, then x and y are non-adjacent since S_i is an independent set of $G[P_i]$. Else $x \in S_i$ and $y \in S_{i'}$ for some $i \neq i'$. P_i and $P_{i'}$ are not linked by a black edge in $(G/\mathcal{P})[C_j]$ since R_j is an independent set in H_j, nor they can be linked by a red edge (there are none in $(G/\mathcal{P})[C_j]$). Thus again, x and y are non-adjacent in G.

***I* has weight at least $\frac{\alpha(G)}{d+1}$.** We claim that $\bigcup_{P_i \subseteq R_j} S_i$ is a heaviest independent set of $G[C_j]$. Note that the P_is that are included in C_j (and partition it) form a module partition of $G[C_j]$. In particular, any heaviest independent set intersecting some $P_i \subseteq C_j$ has to contain a heaviest independent of $G[P_i]$. This is precisely what the algorithm computes. Then a heaviest independent set in $G[C_j]$ packs such subsolutions to maximize the total weight, which is what is computed in H_j.

We conclude by the pigeonhole principle, since a heaviest independent set X of G is such that $w(X \cap C_j) \geqslant \frac{\alpha(G)}{d+1}$ for some $j \in [d+1]$. ◀

3.2 Time-approximation trade-offs

Lemma 14 runs exhaustive algorithms on induced subgraphs of size $O_d(\sqrt{n})$. As such, the latter inputs keep the same twin-width upper bound. To speed up the algorithm (admittedly while worsening the approximation factor) it is tempting to recursively call our very algorithm. We show that this leads to a time-approximation trade-off parameterized

[4] We use this notation as a slight abuse of notation for $(G/\mathcal{P})[\{P_i \ : \ P_i \subseteq C_j\}]$.

by an integer $q = 0, \dots, O_d(\log \log n)$. At one end of this discrete curve, one finds the exact exponential algorithm ($q = 0$), and more interestingly the $d+1$-approximation in time $2^{O_d(\sqrt{n})}$ ($q = 1$), while at the other end lies a polynomial-time algorithm with approximation factor n^ε, where $\varepsilon > 0$ can be made as small as desired.

As we will deal with the same kind of recursions for several problems, we show the following generic abstraction.

▶ **Lemma 15.** *Let $\hat{d}$ be a natural, $d' = 2\hat{d}+2$, and $d := c_{d'} \cdot 2^{4c_{d'}+4}$. Let Π be an optimization graph problem where inputs come with a $\hat{d}$-sequence of their n-vertex graph G, or with a neatly divided matrix $(M, (\mathcal{R}, \mathcal{C})) \in \mathcal{M}_{n,d'}$ conform to G. Let $\mathcal{P}$ be the partition of $V(G)$ given by Lemma 13. Assume that*

1. *Π can be exactly solved in time $2^{O(n)}$, and there are constants c_1, c_2, c_3, and a function $f \geqslant 1$ such that*
2. *a $d^{c_3} r^2$-approximation of Π on G can be built in time n^{c_2} by using at most n^{c_1} calls to an r-approximation of Π –or another optimization problem Π' already satisfying the conclusion of the lemma– on an induced subgraph of G with at most $f(d)\sqrt{n}$ vertices or a full cleanup of an induced subtrigraph of $G/\mathcal{P}$ (on at most $\sqrt{n}$ vertices).*

Then Π can be $d^{c_3(2^q-1)}$-approximated in time

$$(f(d)^q n)^{(2-2^{-q})(c_1+c_2)} \cdot 2^{f(d)^{2(1-2^{-q})} n^{2^{-q}}},$$

for any non-negative integer q.

Proof. The proof is by induction on q. The case $q = 0$ is implied by Item 1. The case $q = 1$, and the induction step in general, is nothing more than an abstraction of Lemma 14, where exhaustive algorithms are replaced by recursive calls.

For any $q \geqslant 0$, we assume that Π can $d^{c_3(2^q-1)}$-approximated in the claimed running time, and show the same statement for the value $q+1$. Following Item 2, we run this algorithm –or one for another optimization problem Π' satisfying the conclusion of the lemma– at most n^{c_1} times on $f(d)\sqrt{n}$-vertex induced subgraphs of the input graph G or on full cleanups of induced subtrigraphs of $G/\mathcal{P}$. The latter graphs have at most $\sqrt{n} \leqslant f(d)\sqrt{n}$ vertices. By Lemma 13, we can compute in polynomial time a neatly divided matrix $(M', (\mathcal{R}', \mathcal{C}')) \in \mathcal{M}_{|V(H)|,d'}$ conform to H, for each graph H of a recursive call; hence the induction applies.

Overall this takes time at most

$$n^{c_1} + n^{c_2} \cdot \left((f(d)^q \cdot f(d)\sqrt{n})^{(2-2^{-q})(c_1+c_2)} \cdot 2^{f(d)^{2(1-2^{-q})} (f(d)\sqrt{n})^{2^{-q}}} \right)$$

$$\leqslant (f(d)^{q+1} n)^{c_1+c_2+\frac{1}{2}(2-2^{-q})(c_1+c_2)} \cdot 2^{f(d)^{2(1-2^{-q})+2^{-q}} n^{\frac{2^{-q}}{2}}}$$

$$= (f(d)^{q+1} n)^{(2-\frac{2^{-q}}{2})(c_1+c_2)} \cdot 2^{f(d)^{2-2^{-q+1}+2^{-q}} n^{2^{-(q+1)}}}$$

$$= (f(d)^{q+1} n)^{(2-2^{-(q+1)})(c_1+c_2)} \cdot 2^{f(d)^{2(1-2^{-(q+1)})} n^{2^{-(q+1)}}}.$$

For the first inequality, we assume that the two summands are larger than 2, so their sum can be bounded by their product.

Besides we get an approximation of factor at most $(d^{c_3(2^q-1)})^2 d^{c_3} = d^{c_3(2^{q+1}-1)}$. ◀

In more legible terms we have proved that:

▶ **Lemma 16.** *Problems Π satisfying the assumptions of Lemma 15 can be $d^{O(1)(2^q-1)}$-approximated in time $2^{O_{d,q}(\sqrt[2^q]{n})}$, for any non-negative integer q.*

While most graph problems admit single-exponential algorithms, we will deal with such a problem that is only known to be solvable in time $2^{O(n \log n)}$. Therefore we prove a variant of Lemma 15 with a slightly worse running time.

▶ **Lemma 17** (⋆). *Let Π be solvable in time $2^{O(n \log n)}$ and satisfy the second item of Lemma 15. Then Π can be $d^{c_3(2^q-1)}$-approximated in time*

$$2^{\left((c_1+c_2)(2-2^{-q})\log f(d)+f(d)^{2(1-2^{-q})}n^{2^{-q}}\right)\log n},$$

for any non-negative integer q.

Again the previous lemma can be rewritten as:

▶ **Lemma 18.** *Problems Π satisfying the assumptions of Lemma 17 can be $d^{O(1)(2^q-1))}$-approximated in time $2^{O_{d,q}(\sqrt[2^q]{n}\log n)}$, for any non-negative integer q.*

We derive from Lemma 17 the following notable regimes.

▶ **Theorem 19** (⋆). *Problems Π satisfying the assumptions of Lemma 17 admit polynomial-time n^{ε}-approximation algorithms, for any $\varepsilon > 0$.*

▶ **Theorem 20** (⋆). *Problems Π satisfying the assumptions of Lemma 15, resp. Lemma 17, admit a $\log n$-approximation algorithm running in time $2^{O_d(n^{\frac{1}{\log\log n}})}$, resp. $2^{O_d(n^{\frac{1}{\log\log n}}\log n)}$.*

We derive the following for Weighted Max Independent Set.

▶ **Theorem 21** (⋆). *Weighted Max Independent Set on n-vertex graphs G (vertex-weighted by w) given with a d'-sequence satisfies the assumptions of Lemma 15. In particular, this problem admits*

- *a $(d+1)^{2^q-1}$-approximation in time $2^{O_{d,q}(n^{2^{-q}})}$, for every integer $q \geqslant 0$,*
- *an n^{ε}-approximation in polynomial-time $O_{d,\varepsilon}(1)\log^{O_d(1)} n \cdot n^{O(1)}$, for any $\varepsilon > 0$, and*
- *a $\log n$-approximation in time $2^{O_d(n^{\frac{1}{\log\log n}})}$,*

with $d := c_{2d'+2} \cdot 2^{4c_{2d'+2}+4}$.

4 Finding the suitable generalization: the case of Coloring

In this section, we deal with the Coloring problem. Unlike for WMIS, we cannot solely resort to recursively calling our Coloring algorithm on smaller graphs. The right problem generalization needs to be found for the inductive calls to work through, and it happens to be Set Coloring.

In the Set Coloring problem, the input is a couple (G, b) where G is a graph, and b is a function assigning a positive integer to each vertex of G. The goal is to find, for each $v \in V(G)$, a set S_v of at least $b(v)$ colors such that $S_u \cap S_v = \emptyset$ whenever $uv \in E(G)$, and minimizing $|\cup_{v\in V(G)} S_v|$. Let $\chi_b(G)$ be the optimal value of Set Coloring for (G, b). Observe that Coloring corresponds to the case where $b(v) = 1$ for every $v \in V(G)$.

▶ **Theorem 22.** *Set Coloring (and hence Coloring) on n-vertex graphs G given with a d'-sequence satisfies the assumptions of Lemma 17. In particular, this problem admits*

- *a $(d+1)^{2^q-1}$-approximation in time $2^{O_{d,q}(n^{2^{-q}}\log n)}$, for every integer $q \geqslant 0$, and*
- *an n^{ε}-approximation in polynomial-time for any $\varepsilon > 0$.*

with $d := c_{2d'+2} \cdot 2^{4c_{2d'+2}+4}$.

Proof. It is known [25] that Set Coloring can be solved using the inclusion-exclusion principle in time $O^*(\max_{v\in V(G)} b(v)^n) = 2^{O(n\log n)}$. We now prove that it satisfies the second item of Lemma 15. We denote by $\mathcal{A}$ the r-approximation algorithm of the statement, which we will use on instances of Set Coloring. In particular, we will call it at most $\sqrt{n}+1$ times, and will obtain at the end a $(d+1)r^2$-approximation on our input (G,b) in polynomial time.

We first apply Lemma 13 to get, in polynomial-time, a partition $\mathcal{P} = \{P_1,\dots,P_{\lfloor\sqrt{n}\rfloor}\}$ of $V(G)$ whose parts have size at most $d\sqrt{n}$ and such that $R(G/\mathcal{P})$ has maximum degree at most d. For every $i \in [\lfloor\sqrt{n}\rfloor]$, we use $\mathcal{A}$ to compute an r-approximated solution c_{P_i} of $(G[P_i], b_{|P_i})$. We denote by b' the function which assigns, to each P_i, the number of colors of c_{P_i}. We now compute, in polynomial-time, a proper $(d+1)$-coloring of $R(G/\mathcal{P})$, which defines the sets C_1, …, C_{d+1}. For each $j \in [d+1]$, we construct another Set Coloring instance consisting of the graph $H_j = (G/\mathcal{P})[C_j]$ (recall that this trigraph has no red edge, and can thus be seen as a graph), together with the function $b'_{|C_j}$. Again we use $\mathcal{A}$ to compute an r-approximated solution on $(H_j, b'_{|C_j})$. We denote by c_H this solution. Let G_j be the subgraph of G induced by $\cup_{P_i\in C_j} P_i$, and b_j the restriction of b to $V(G_j)$. We now show how to construct a solution c_j of Set Coloring to (G_j, b_j) from c_H and all c_{P_i}. Recall that for every $P_i \in C_j$, every $v \in P_i$, we have that $c_{P_i}(v)$ is a subset of $\{1,\dots,b'(P_i)\}$ of size at least $b(v)$, and that $c_H(P_i)$ is a subset of size at least $b'(P_i)$. Hence, for each $P_i \in C_j$, one can choose an arbitrary bijection τ from $\{1,\dots,b'(P_i)\}$ to $c_H(P_i)$, and define to each vertex $v \in P_i$ the set $c_j(v)$ as $\{\tau(x) : x \in c_{P_i}(v)\}$.

By construction, this solution is a feasible one for the instance (G_j, b_j). Let us prove that it is an r^2-approximation of $\chi_{b_j}(G_j)$. First, by definition of c_H, our solution uses at most $r\cdot\chi_{b'_{|C_j}}(H_j)$ colors. Then, by definition of c_{P_i} for every $P_i \in C_j$, we have $b'_{C_j}(P_i) \leqslant r\cdot\chi_{b_{|P_i}}(G[P_i])$. Now, denote by Γ the function which assigns to each $P_i \in C_j$ the number $\chi_{b_{|P_i}}(G[P_i])$. We now use the following claim, whose proof is left to the reader.

▷ Claim 23. Let (G,b) be an instance of Set Coloring, and $r \in \mathbb{R}_+$. It holds that $\chi_{r\cdot b}(G) \leqslant r\cdot\chi_b(G)$, where $r\cdot b$ is the function which assigns $r\cdot b(v)$ to each $v \in V(G)$.

This implies $\chi_{b'_{|C_j}}(H_j) \leqslant r\cdot\chi_\Gamma(H_j)$, and thus our solution uses at most $r^2\cdot\chi_\Gamma(H_j)$ colors. We now prove the following claim.

▷ Claim 24. $\chi_\Gamma(H_j) \leqslant \chi_{b_j}(G_j)$.

Proof of the claim. Let c be an optimal solution for (G_j, b_j). For every distinct P_i, $P_{i'} \in C_j$ such that $P_iP_{i'}$ is an edge of H_j, it holds that there are all possible edges between P_i and $P_{i'}$ in G_j (by definition of the coloring C_1, ⋯, C_{d+1}), hence it holds that $\bigcup_{v\in P_i} c(v)$ and $\bigcup_{v\in P_{i'}} c(v)$ have empty intersection. Moreover, by definition of Γ, we have that $\bigcup_{v\in P_i} c(v)$ is of size at least $\Gamma(P_i)$, hence the function which assigns $\bigcup_{v\in P_i} c(v)$ to each P_i is a feasible solution for (H_j, Γ) using at most $\chi_{b_j}(G_j)$ colors. ◁

We now have in hand an r^2-approximated solution of (G_j, b_j) for every $j \in [d+1]$, which can be turned into a $(d+1)r^2$-approximated solution of (G,b), as desired. ◀

References

1 Nikhil Bansal, Parinya Chalermsook, Bundit Laekhanukit, Danupon Nanongkai, and Jesper Nederlof. New tools and connections for exponential-time approximation. *Algorithmica*, 81(10):3993–4009, 2019. doi:10.1007/s00453-018-0512-8.

2 Pierre Bergé, Édouard Bonnet, and Hugues Déprés. Deciding twin-width at most 4 is NP-complete. 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 18:1–18:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.18.

3 Édouard Bonnet, Dibyayan Chakraborty, Eun Jung Kim, Noleen Köhler, Raul Lopes, and Stéphan Thomassé. Twin-width VIII: delineation and win-wins. *CoRR*, abs/2204.00722, 2022. doi:10.48550/arXiv.2204.00722.

4 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.

5 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.35.

6 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20–24, 2022*, pages 924–937. ACM, 2022. doi:10.1145/3519935.3520037.

7 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

8 Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of max independent set, min vertex cover and related problems by moderately exponential algorithms. *Discret. Appl. Math.*, 159(17):1954–1970, 2011. doi:10.1016/j.dam.2011.07.009.

9 Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Graph products revisited: Tight approximation hardness of induced matching, poset dimension and more. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1557–1576. SIAM, 2013. doi:10.1137/1.9781611973105.112.

10 Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 370–379, 2013. doi:10.1109/FOCS.2013.47.

11 Marek Cygan, Lukasz Kowalik, Marcin Pilipczuk, and Mateusz Wykurz. Exponential-time approximation of hard problems. *CoRR*, abs/0810.4934, 2008. arXiv:0810.4934.

12 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 624–633. ACM, 2014. doi:10.1145/2591796.2591884.

13 Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost NP-complete (preliminary version). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 2–12. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185341.

14 Harold N. Gabow and Shuxin Nie. Finding a long directed cycle. *ACM Trans. Algorithms*, 4(1):7:1–7:21, 2008. doi:10.1145/1328911.1328918.

15 Jakub Gajarský, Michal Pilipczuk, Wojciech Przybyszewski, and Szymon Torunczyk. Twin-width and types. 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 123:1–123:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.123.

16 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

17 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

18 Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Inf. Process. Lett.*, 46(4):169–172, 1993. doi:10.1016/0020-0190(93)90022-2.

19 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 627–636, 1996. doi:10.1109/SFCS.1996.548522.

20 Petr Hliněný. Twin-width of planar graphs is at most 9, 2022. doi:10.48550/arXiv.2205.05378.

21 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.

22 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.

23 Stefan Kratsch, Florian Nelles, and Alexandre Simon. On triangle counting parameterized by twin-width. *CoRR*, abs/2202.06708, 2022. arXiv:2202.06708.

24 Carsten Lund and Mihalis Yannakakis. The approximation of maximum subgraph problems. In Andrzej Lingas, Rolf G. Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming, 20nd International Colloquium, ICALP93, Lund, Sweden, July 5-9, 1993, Proceedings*, volume 700 of *Lecture Notes in Computer Science*, pages 40–51. Springer, 1993. doi:10.1007/3-540-56939-1_60.

25 Jesper Nederlof. Inclusion exclusion for hard problems, 2008.

26 Michal Pilipczuk, Marek Sokolowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. 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 52:1–52:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.52.

27 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

Tight Lower Bounds for Problems Parameterized by Rank-Width

Benjamin Bergougnoux
University of Warsaw, Poland

Tuukka Korhonen
University of Bergen, Norway

Jesper Nederlof
Utrecht University, The Netherlands

Abstract

We show that there is no $2^{o(k^2)}n^{O(1)}$ time algorithm for Independent Set on n-vertex graphs with rank-width k, unless the Exponential Time Hypothesis (ETH) fails. Our lower bound matches the $2^{O(k^2)}n^{O(1)}$ time algorithm given by Bui-Xuan, Telle, and Vatshelle [Discret. Appl. Math., 2010] and it answers the open question of Bergougnoux and Kanté [SIAM J. Discret. Math., 2021]. We also show that the known $2^{O(k^2)}n^{O(1)}$ time algorithms for Weighted Dominating Set, Maximum Induced Matching and Feedback Vertex Set parameterized by rank-width k are optimal assuming ETH. Our results are the first tight ETH lower bounds parameterized by rank-width that do not follow directly from lower bounds for n-vertex graphs.

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

Keywords and phrases rank-width, exponential time hypothesis, Boolean-width, parameterized algorithms, independent set, dominating set, maximum induced matching, feedback vertex set

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.11

Related Version *Full Version*: https://arxiv.org/abs/2210.02117

Funding *Tuukka Korhonen*: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
Jesper Nederlof: Supported by the project CRACKNP that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 853234).

Acknowledgements This work was initiated while the authors attended the "2022 Advances in Parameterized Graph Algorithms" workshop.

1 Introduction

Decompositions of graphs and their associated width parameters are a popular approach for solving NP-hard graphs problems. Several fundamental graph problems, like Independent Set, are known to be fixed-parameter tractable parameterized by width parameters like the treewidth (tw) of the input graph [1]. While treewidth is the most prominent width parameter, it has limited applicability as it can be bounded only for sparse graphs. To capture the tractability of problems on structured dense graph classes, like cographs and distance-hereditary graphs, the parameter clique-width (cw) was introduced by Courcelle, Engelfriet, and Rozenberg [9]. Every graph with treewidth tw has clique-width $\mathsf{cw} \leq O(2^{\mathsf{tw}})$, and for example complete graphs have unbounded treewidth but constant clique-width. Many fixed-parameter tractability results parameterized by treewidth generalize to parameterization by clique-width [10], and in particular Independent Set can be solved in time $2^{\mathsf{cw}}n^{O(1)}$ given a decomposition witnessing clique-width at most cw.

© Benjamin Bergougnoux, Tuukka Korhonen, and Jesper Nederlof;
licensed under Creative Commons License CC-BY 4.0

The width parameter rank-width (rw) was introduced by Oum and Seymour [29] in order to obtain a fixed-parameter approximation algorithm for computing the clique-width. In particular, they showed that $\mathsf{rw} \leq \mathsf{cw} \leq 2^{\mathsf{rw}+1} - 1$ and rank-width can be 3-approximated in time $8^{\mathsf{rw}}n^{O(1)}$, implying an exponential $2^{O(\mathsf{cw})}$-approximation for clique-width within the same time complexity. Even though multiple improvements for computing rank-width have been given [17, 21, 24, 27], the only known way to approximate clique-width remains via computing rank-width.

In the past decade, the focus of the study of algorithms on graph decompositions has shifted from complexity classification into establishing fine-grained bounds on the time complexity as a function of the parameter [4, 5, 11, 12, 15, 16, 20, 23, 25, 26, 28], under either the Exponential Time Hypothesis (ETH) or the Strong Exponential Time Hypothesis (SETH) [22]. For example, Lokshtanov, Marx, and Saurabh [26] showed that assuming SETH, Independent Set cannot be solved in time $(2-\varepsilon)^{\mathsf{pw}}n^{O(1)}$ parameterized by path-width (pw) for any constant $\varepsilon > 0$, which also translates into a tight lower bound of $(2-\varepsilon)^{\mathsf{cw}}n^{O(1)}$ parameterized by clique-width because of the relation $\mathsf{cw} \leq \mathsf{pw} + 2$ [14]. Lampis [25] showed that for every constant $k \geq 3$, the optimal time complexity of k-coloring parameterized by clique-width is $(2^k-2)^{\mathsf{cw}}n^{O(1)}$ assuming SETH.

Even though fine-grained lower bounds parameterized by clique-width have been intensively studied [4, 5, 15, 16, 25], less attention has been given to fine-grained lower bounds parameterized by rank-width. As the only known way to compute clique-width is via computing rank-width and using the constructive version of the inequalities $\mathsf{rw} \leq \mathsf{cw} \leq 2^{\mathsf{rw}+1} - 1$, it could be argued that fine-grained bounds parameterized by rank-width have more significance than bounds parameterized by clique-width: In the end, the only known way to use clique-width in its full generality is to actually use rank-width.

The lack of fine-grained lower bounds for parameterizations by rank-width in the literature could be explained by the fact that the best known upper bounds appear to require more complicated arguments than for other width parameters. In particular, while the translation to clique-width or a straightforward dynamic programming leads to a double-exponential $2^{2^{O(\mathsf{rw})}}n^{O(1)}$ time algorithm for Independent Set parameterized by rank-width, Bui-Xuan, Telle, and Vatshelle showed in 2010 that surprisingly this is not optimal, giving a $2^{O(\mathsf{rw}^2)}n^{O(1)}$ time algorithm by exploiting the algebraic properties of rank-width [6]. It was asked by Bergougnoux and Kanté [3] whether this algorithm could be shown to be optimal assuming ETH, and by Vatshelle [30] whether a $2^{O(\mathsf{rw})}n^{O(1)}$ time algorithm exists.

In this paper, we show that assuming ETH, the $2^{O(\mathsf{rw}^2)}n^{O(1)}$ time algorithm for Independent Set by Bui-Xuan, Telle, and Vatshelle is optimal. We show in fact a slightly more general result, using parameterization by linear rank-width (lrw), which is a path-like version of rank-width whose value is at least the rank-width, i.e., $\mathsf{rw} \leq \mathsf{lrw}$.

▶ **Theorem 1.1.** *Unless ETH fails, there is no $2^{o(\mathsf{lrw}^2)}n^{O(1)}$ time algorithm for Independent Set, where lrw is the linear rank-width of the input graphs.*

Unlike for the fine-grained lower bounds parameterized by clique-width, for our result the matching upper bound holds even without the assumption that the decomposition is given because rank-width can be 3-approximated in time $O(8^{\mathsf{rw}}n^4)$ [27].

Theorem 1.1 is the first ETH-tight lower bound parameterized by rank-width that does not follow directly from a lower bound for n-vertex graphs and the relation $\mathsf{rw} \leq n$. Tight bounds of the latter type are known for the problems of finding a largest induced subgraph with odd vertex degrees and for partitioning a graph into a constant number of such induced subgraphs. In particular, these problems admit $2^{O(\mathsf{rw})}n^{O(1)}$ time algorithms but cannot be solved in time $2^{o(n)}$ unless ETH fails [2].

Algorithms with time complexity $2^{O(\mathsf{rw}^2)}n^{O(1)}$ were given for Dominating Set and Maximum Induced Matching by Bui-Xuan, Telle, and Vatshelle [6, 8] and for Feedback Vertex Set by Ganian and Hlinený [18]. We extend our lower bound construction to show that these algorithms for Maximum Induced Matching and Feedback Vertex Set and a weighted variant of the algorithm for Dominating Set are optimal assuming ETH.

▶ **Theorem 1.2.** *Unless ETH fails, there is no $2^{o(\mathsf{lrw}^2)}n^{O(1)}$ time algorithm for Weighted Dominating Set, Maximum Induced Matching, or Feedback Vertex Set, where* lrw *is the linear rank-width of the input graphs.*

Boolean-width. Boolean-width (boolw) is a width-parameter introduced by Bui-Xuan, Telle, and Vatshelle [7]. It is defined on branch decompositions over the vertex set $V(G)$ with the cut function $\mathsf{boolw}(A, B) = \log_2 |\{N(X) \cap B : X \subseteq A\}|$, which naturally leads to algorithms with time complexity $2^{O(\mathsf{boolw})}n^{O(1)}$ for local problems when a branch decomposition with Boolean-width boolw is given. Bui-Xuan, Telle, and Vatshelle proved that $\log_2 \mathsf{rw} \leqslant \mathsf{boolw} \leqslant O(\mathsf{rw}^2)$ and asked as an open question whether the upper bound $\mathsf{boolw} \leqslant O(\mathsf{rw}^2)$ is tight [7]. Since Independent Set can be solved in time $2^{O(\mathsf{boolw})}n^{O(1)}$ given a branch decomposition with Boolean-width boolw, Theorem 1.1 gives evidence that there are graphs whose Boolean-width is quadratic in rank-width. A small variant of our construction for Theorem 1.1 can be used to give a quadratic separation between Boolean-width and rank-width, answering the question of Bui-Xuan, Telle, and Vatshelle:

▶ **Theorem 1.3.** *There are graphs with rank-width k and Boolean-width $\Omega(k^2)$ for arbitrarily large k.*

The proof of this theorem is given in the full version of this paper.

Our Method. We briefly describe our main technical ideas for Theorem 1.1, as the other ideas are similar.

Our starting point is the $2^{O(\mathsf{rw}^2)}n^{O(1)}$ time algorithm for Independent Set from [6]. Intuitively, the algorithm can be thought of as normal dynamic programming over tree decompositions where we have table entries for each subset of a bag of the tree-decomposition (which is a separator in the graph). The twist however is that the size of the separator may be large, but instead we are only guaranteed that the rank (over $\mathbb{F}_2$) of the incidence matrix of the cut between the separator and the remainder of the graph is small. The crucial observation from [6] is that we do not need to know the exact subset of the separator of vertices selected in the independent set, but only its set of neighbors across the cut, and that such a neighborhood can be described by a (row- or column-) *subspace* of the mentioned incidence matrix. Since there are only $2^{O(\mathsf{rw}^2)}$ such subspaces, the runtime follows.

To turn this encoding idea into a reduction from 3-CNF-SAT to Independent Set on graphs of rank-width rw (and get an ETH-tight lower bound), we first show this description cannot be further shortened: Two vertex sets of the separator that describe different subspaces in fact will have different sets of neighbors accross the cut. This allows us to design a "copy gadget": Once locally a certain vertex subset is chosen to be in the independent set, this is has to be copied in various places throughout the graph (such that we can check a clause of the 3-CNF per one such location). Furthermore, there is a simple inductive construction of subspaces that enables us to directly encode assignments of $\Omega(\mathsf{rw}^2)$ variables of an 3-CNF formula into a subspace of $\mathbb{F}_2^{\mathsf{rw}}$. This allows us to get a tight bound.

Organization. The rest of this paper is organized as follows. In Section 2 we set up notation. In Section 3 we present structural results on the maximal unique cut with rank-width k (that we call "the universal k-rank cut"). Subsequently, Section 4 builds upon these results to prove Theorem 1.1. In Section 5 we prove the lower bounds for Maximum Induced Matching and Feedback Vertex Set. We prove the lower bound for Weighted Dominating Set in Section 6. We provide a brief conclusion in Section 7.

2 Notation

Given two integers i, j such that $1 \leqslant i \leqslant j$, we denote by $[i, j]$ the set of integer $\{i, i+1, \ldots, j\}$ and by $[i]$ the set $\{1, 2, \ldots, i\}$. The size of a set V is denoted by $|V|$ and its power set is denoted by 2^V.

Graphs. Our graph terminology is standard and we refer to [13]. The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$. For every vertex set $X \subseteq V(G)$, when the underlying graph is clear from context, we denote by $\overline{X}$ the set $V(G) \setminus X$. An edge between two vertices x and y is denoted by xy or yx. The set of vertices that are adjacent to x is denoted by $N_G(x)$. For a set $U \subseteq V(G)$, we define $N_G(U) := \bigcup_{x \in U} N_G(x) \setminus U$. If the underlying graph is clear, then we may remove G from the subscript. Two distinct vertices $u, v \in V(G)$ are twins if $N(v) \setminus \{u\} = N(u) \setminus \{v\}$. They are true twins if $uv \in E(G)$ and false twins if $uv \notin E(G)$.

The subgraph of G induced by a subset X of its vertex set is denoted by $G[X]$. For two disjoint subsets X and Y of $V(G)$, we denote by $G[X, Y]$ the bipartite graph with vertex set $X \cup Y$ and edge set $\{xy \in E(G) : x \in X \text{ and } y \in Y\}$.

Problem Statements. An *independent set* is a set of vertices that induces an edgeless graph. A *matching* is a set of edges having no common endpoint and an *induced matching* is a matching M where every pair of edges of M do not have a common adjacent edge in G. Given a graph, the problems Independent Set and Maximum Induced Matching ask for respectively an independent set and an induced matching of maximum size.

A *feedback vertex set* is the complement of a set of vertices inducing a forest (i.e. acyclic graph). A set $D \subseteq V(G)$ *dominates* a set $U \subseteq V(G)$ if every vertex in U is either in D or is adjacent to a vertex in D. A *dominating set* of G is a set that dominates $V(G)$. Given a graph, the problems Dominating Set and Feedback Vertex Set ask respectively for a dominating set and a feedback vertex set of minimum size.

Given a graph G with a weight function $w : V(G) \to \mathbb{N}$, the problem Weighted Independent Set (resp. Weighted Dominating Set) asks for an independent set of maximum weight (resp. dominating set of minimum weight), where the weight of a set $X \subseteq V(G)$ is $\sum_{x \in X} w(x)$.

Width parameters. Let V be a finite set with $|V| \geq 3$, and f a function $f : 2^V \to \mathbb{Z}_{\geq 0}$ so that $f(\emptyset) = 0$ and $f(A) = f(\overline{A})$ for all $A \subseteq V$. A branch decomposition of f is a tree whose all internal nodes have degree 3 and whose leaves are bijectively mapped to V. Observe that every edge of a branch decomposition of f corresponds to a bipartition $(A, \overline{A})$ of V given by the leaves on different sides of the edge. The width of the decomposition is the maximum value of $f(A)$ over the bipartitions $(A, \overline{A})$ corresponding to the edges, and the branch-width of f is the minimum width of a branch decomposition of f. The width of a permutation σ of V is the maximum value of $f(A)$ over prefixes A of the permutation. The linear branch-width

of f is the minimum width of a permutation of V. Notice that the branch-width of f is always at most the linear branch-width of f, in particular linear branch-width corresponds to a restriction of branch-width where we demand the tree to be a caterpillar.

Let G be a graph and $A, B \subseteq V(G)$ two disjoint sets of vertices. We define $M_G(A,B)$ to be the $|A| \times |B|$ 0-1 matrix representing the bipartite graph $G[A,B]$. The cut-rank $\mathsf{rw}(A,B)$ between A and B is defined as the $\mathbb{F}_2$-rank of $M_G(A,B)$. For a set of vertices $A \subseteq V(G)$ we define $\mathsf{rw}(A) = \mathsf{rw}(A, \overline{A})$. The rank-width of a graph G is the branch-width of the cut-rank function rw and the linear rank-width of G is the linear branch-width of rw. The Boolean-rank between A and B is defined as $\mathsf{boolw}(A,B) = \log_2 |\{N(X) \cap B : X \subseteq A\}|$, and we define $\mathsf{boolw}(A) = \mathsf{boolw}(A, \overline{A})$. The Boolean-width of a graph G is the branch-width of boolw.

We will use the following lemma about the cut-rank.

▶ **Lemma 2.1.** *Let A, B be disjoint subsets of $V(G)$ and $S \subseteq V(G)$. It holds that $\mathsf{rw}(A,B) \leq \mathsf{rw}(A \cap S, B) + \mathsf{rw}(A \setminus S, B)$.*

Proof. The edges of $G[A \cap S, B]$ and $G[A \setminus S, B]$ are disjoint, and therefore $M_G(A,B)$ can be written as the sum of (appropriately permuted) $M_G(A \cap S, B)$ and $M_G(A \setminus S, B)$. The lemma follows from the fact that the rank of a sum of matrices is at most the sum of the ranks of the matrices. ◀

3 Structural Results on the Universal k-Rank Cut

In this section, we study the *universal k-rank cut* which is the unique inclusion-wise maximal cut of a given rank k with no twin vertices. This cut was used in [6] to give a non-algebraic definition of rank-width and in [7] to prove that some cuts can have rank-width k and Boolean-width $\Omega(k^2)$ for arbitrary large k.

▶ **Definition 3.1.** *For any integer $k \geq 1$, the universal k-rank cut R_k is defined as the bipartite graph with $2 \cdot 2^k$ vertices with color classes $A^k := \{a_s : s \subseteq [k]\}$ and $B^k := \{b_t : t \subseteq [k]\}$ such that a_s and b_t are adjacent if and only if $|s \cap t|$ is odd. Given $\mathcal{S} = \{s_1, \ldots, s_\ell\} \subseteq 2^{[k]}$, we define $A^k[\mathcal{S}] := \{a_{s_1}, \ldots, a_{s_\ell}\}$ and $B^k[\mathcal{S}] := \{b_{s_1}, \ldots, b_{s_\ell}\}$. We may omit k from the superscript when it is clear from the context.*

The following lemma characterizes the cut-rank in terms of R_k.

▶ **Lemma 3.2** ([6])**.** *Let G be a graph and A, B disjoint subsets of $V(G)$. It holds that $\mathsf{rw}(A,B) \leq k$ if and only if $G[A,B]$, after removing twins, is an induced subgraph of R_k.*

In particular, it follows that in the graph R_k we have $\mathsf{rw}(A^k, B^k) = k$.

In the following, we define a family $\mathscr{F}_k$ of subsets of $2^{[k]}$ that will play a crucial role in our reductions. One major property of this family will be that for every distinct $\mathcal{S}_1, \mathcal{S}_2$ in $\mathscr{F}_k$, the sets of vertices $A^k[\mathcal{S}_1]$ and $A^k[\mathcal{S}_2]$ have different neighborhoods in R_k.

▶ **Definition 3.3.** *Given any integer $k \geq 1$ and $\mathcal{S} = \{s_1, \ldots, s_\ell\} \subseteq 2^{[k]}$, we define $\mathcal{S} \otimes k+1$ as the collection consisting of $\{s_1, \ldots, s_\ell, \{k+1\}\}$ and all the sets $\{s'_1, \ldots, s'_\ell\}$ such that for every $i \in [\ell]$, we have $s'_i \in \{s_i, s_i \cup \{k+1\}\}$. We define $\mathscr{F}_1$ as the family containing $\emptyset$ and $\{\{1\}\}$. For every $k \geq 1$, we define $\mathscr{F}_{k+1} = \bigcup_{\mathcal{S} \in \mathscr{F}_k} \mathcal{S} \otimes k+1$.*

For example, the collection $\mathscr{F}_2$ is the union of $\emptyset \otimes 2$ and $\{\{1\}\} \otimes 2$ where $\emptyset \otimes 2 = \{\emptyset, \{\{2\}\}\}$ and $\{\{1\}\} \otimes 2$ is the collection containing $\{\{1\}\}$, $\{\{1,2\}\}$ and $\{\{1\},\{2\}\}$.

It is natural to view $\mathcal{S} = \{s_1, \ldots, s_\ell\} \subseteq 2^{[k]}$ as a binary $\ell \times k$ matrix (so the s_i's are interpreted as k-dimensional binary vectors). Then the $\otimes$ operation can be thought of as transforming a single matrix into a set of matrices as displayed in Figure 1, and the family $\mathcal{F}_i$ can be thought of as the family of all different binary matrices with k columns that are in row-reduced echelon form: The steps in which we add a row are exactly the pivotal columns.

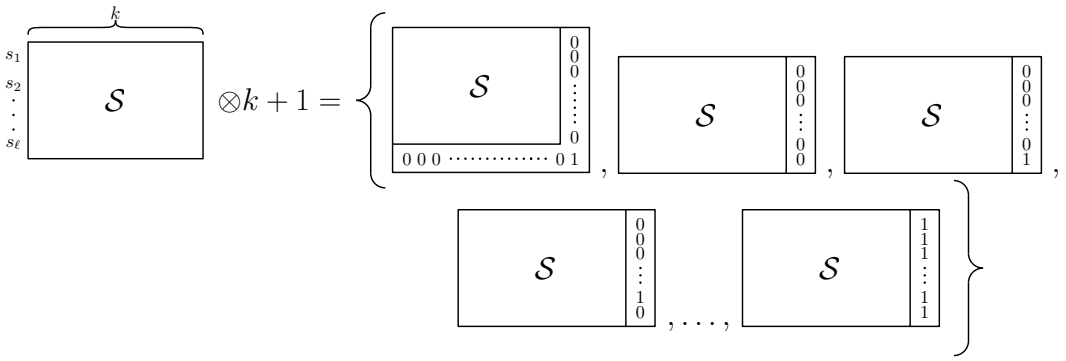


Figure 1 Illustration of the interpretation of the $\otimes$ operation as an operation on binary matrices.

The following observation gives an alternative definition of $\mathscr{F}_k$.

▶ **Observation 3.4.** *For every $k \geq 1$ and $\mathcal{S} \subseteq 2^{[k]}$, we have $\mathcal{S} = \{s_1, \ldots, s_{|\mathcal{S}|}\} \in \mathscr{F}_k$ if and only if there exist pairwise distinct integers $\alpha_1, \ldots, \alpha_{|\mathcal{S}|} \in [k]$ such that for every $i \in [|\mathcal{S}|]$, we have $s_i \cap \{\alpha_1, \ldots, \alpha_{|\mathcal{S}|}\} = \{\alpha_i\}$ and $s_i \subseteq \{\beta \in [k] : \alpha_i \leqslant \beta\}$.*

Note that the distinct integers $\alpha_1, \ldots, \alpha_{|\mathcal{S}|}$ in Observation 3.4 are the column indices of the leading coefficients in the matrix formulation in Figure 1.

We are ready to prove the two properties on the collections in $\mathscr{F}_k$ that we will use in our reductions. We first prove the distinct neighborhoods property.

▶ **Lemma 3.5.** *For every $k \geq 1$ and every pair $\mathcal{S}_1, \mathcal{S}_2 \in \mathscr{F}_k$ with $\mathcal{S}_1 \neq \mathcal{S}_2$ it holds that $N_{R_k}(A^k[\mathcal{S}_1]) \neq N_{R_k}(A^k[\mathcal{S}_2])$.*

Proof. We start with the following claim.

▷ Claim 3.6. For every $k \geq 1$ and $\mathcal{S}, \mathcal{X} \subseteq 2^{[k]}$ with $\mathcal{X} \subseteq \mathcal{S} \in \mathscr{F}_k$, there exists $t \subseteq [k]$ such that for every $s \in \mathcal{S}$, $|s \cap t|$ is odd if and only if $s \in \mathcal{X}$ (i.e. $N(b_t) \cap A^k[\mathcal{S}] = A^k[\mathcal{X}]$).

Proof. Let $\mathcal{S} = \{s_1, \ldots, s_\ell\} \in \mathscr{F}_k$ and $\mathcal{X} \subseteq \mathcal{S}$. By Observation 3.4, there exists $\alpha_1, \ldots, \alpha_\ell \in [k]$ such that for every $i \in [\ell]$, we have $s_i \cap \{\alpha_1, \ldots, \alpha_\ell\} = \{\alpha_i\}$. Let $t = \{\alpha_i : i \in [\ell] \wedge s_i \in \mathcal{X}\}$. Observe that, for every $i \in [\ell]$, we have $s_i \cap t = \{\alpha_i\}$ iff $s_i \in \mathcal{X}$. Hence, $|s_i \cap t|$ is odd iff $s_i \in \mathcal{X}$ for every $i \in [\ell]$. ◁

We are now ready to prove the lemma by induction on k. It is obviously true for $\mathscr{F}_1$. Let $k \geq 1$ and suppose that the lemma holds for $\mathscr{F}_k$. Let $\mathcal{S}'_1, \mathcal{S}'_2 \in \mathscr{F}_{k+1}$ such that $\mathcal{S}'_1 \neq \mathcal{S}'_2$. By construction of $\mathscr{F}_{k+1}$, for every $i \in \{1, 2\}$, there exists a unique $\mathcal{S}_i \in \mathscr{F}_k$ such that $\mathcal{S}'_i \in \mathcal{S}_i \otimes k + 1$.

If $\mathcal{S}_1 \neq \mathcal{S}_2$, then by induction hypothesis, $N_{R_k}(A^k[\mathcal{S}_1]) \neq N_{R_k}(A^k[\mathcal{S}_2])$. By definition of R_{k+1}, for each $i \in \{1, 2\}$, the sets of vertices $A^{k+1}[\mathcal{S}_i]$ and $A^{k+1}[\mathcal{S}'_i]$ have the same neighborshoods in R_{k+1} when restricted to the subset $B^k \subseteq B^{k+1}$. We conclude that if $\mathcal{S}_1 \neq \mathcal{S}_2$, then $N_{R_{k+1}}(A^{k+1}[\mathcal{S}'_1]) \neq N_{R_{k+1}}(A^{k+1}[\mathcal{S}'_2])$.

It remains to consider the case when $\mathcal{S}_1 = \mathcal{S}_2 = \{s_1, \ldots, s_\ell\}$. For both $i \in \{1,2\}$, we define a set $\mathcal{X}_i := \{s_j : j \in [\ell] \wedge s_j \cup \{k+1\} \in \mathcal{S}'_i\}$. Since $\mathcal{S}'_1 \neq \mathcal{S}'_2$, we have $\mathcal{X}_1 \neq \mathcal{X}_2$. By Claim 3.6, there exists $t_1 \subseteq [k]$ such that $|s_j \cap t_1|$ is odd if and only if $s_j \in \mathcal{X}_1$ for every $j \in [\ell]$. We claim that $b_{t_1 \cup \{k+1\}}$ is adjacent to $A^{k+1}[\mathcal{S}'_2]$ but not to $A^{k+1}[\mathcal{S}'_1]$ in R_{k+1}. For every $j \in [\ell]$, we have $s_j \in \mathcal{X}_1$ if and only if $s_j \cup \{k+1\} \in \mathcal{S}'_1$, we deduce that $|s \cap (t_1 \cup \{k+1\})|$ is even for every $s \in \mathcal{S}'_1$. Consequently, $b_{t_1 \cup \{k+1\}}$ is not adjacent to $A^{k+1}[\mathcal{S}'_1]$ in R_{k+1}. As $\mathcal{X}_1 \neq \mathcal{X}_2$, there exists $j \in [\ell]$ such that $s_j \in \mathcal{X}_1 \triangle \mathcal{X}_2$.

- If $s_j \in \mathcal{X}_1 \setminus \mathcal{X}_2$, then $s_j \in \mathcal{S}'_2$ and $|s_j \cap (t_1 \cup \{k+1\})| = |s_j \cap t_1|$ is odd.
- If $s_j \in \mathcal{X}_2 \setminus \mathcal{X}_1$, then $s_j \cup \{k+1\} \in \mathcal{S}'_2$ and $|s_j \cup \{k+1\} \cap (t_1 \cup \{k+1\})| = |s_j \cap t_1| + 1$ is odd since $|s_j \cap t_1|$ is even.

We deduce that $b_{t_1 \cup \{k+1\}}$ is adjacent to $A^{k+1}[\mathcal{S}'_2]$ and conclude that $N_{R_{k+1}}(A^{k+1}[\mathcal{S}'_1]) \neq N_{R_{k+1}}(A^{k+1}[\mathcal{S}'_2])$. ◀

We then show that the size of the neighborhood of $A^k[\mathcal{S}]$ depends only on $|\mathcal{S}|$.

▶ **Lemma 3.7.** *For every $k \geq 1$ and $\mathcal{S} \in \mathscr{F}_k$, we have $|N_{R_k}(A^k[\mathcal{S}])| = 2^k - 2^{k-|\mathcal{S}|}$.*

Proof. Let $k \geq 1$ and $\mathcal{S} \in \mathscr{F}_k$. In this proof, we view each subset $s \subseteq [k]$ as a vector $(s_1, \ldots, s_k) \in \mathbb{Z}_2^k$ with $s_i = 1$ if and only if $i \in s$ for every $i \in [k]$. Observe that, given $s, t \subseteq [k]$, the inner product $\langle s, t \rangle$ of s and t is the sum $s_1t_1 + s_2t_2 + \cdots + s_kt_k$ over $\mathbb{Z}_2$ and it equals 1 if and only if $|s \cap t|$ is odd.

We denote by $\langle \mathcal{S} \rangle$ the subspace of $\mathbb{Z}_2^k$ generated by the vectors of $\mathcal{S}$ and by $\langle \mathcal{S} \rangle^\perp := \{t \subseteq [k] : \forall s \in \langle \mathcal{S} \rangle, \langle s, t \rangle = 0\}$ its orthogonal subspace.

Observe that, for every $s_1, s_2, t \subseteq [k]$, if $\langle s_1, t \rangle = \langle s_2, t \rangle = 0$, then $\langle s_1 + s_2, t \rangle = 0$. Consequently, we have $t \in \langle \mathcal{S} \rangle^\perp$ if and only if $\langle s, t \rangle = 0$ for every $s \in \mathcal{S}$. We deduce that

$$N_{R_k}(A^k[\mathcal{S}]) = B^k \setminus B^k[\langle \mathcal{S} \rangle^\perp].$$

By the rank-nullity theorem, we have $\dim\langle \mathcal{S} \rangle + \dim\langle \mathcal{S} \rangle^\perp = k$. By Claim 3.6, for every $s \in \mathcal{S}$, there exists $t \subseteq [k]$ such that $\langle s, t \rangle = 1$ and $\langle s', t \rangle = 0$ for every $s' \in \mathcal{S} \setminus \{s\}$. We deduce that $\dim\langle \mathcal{S} \rangle = |\mathcal{S}|$ and thus $\dim\langle \mathcal{S} \rangle^\perp = k - |\mathcal{S}|$. As $N_{R_k}(A^k[\mathcal{S}]) = B^k \setminus B^k[\langle \mathcal{S} \rangle^\perp]$, we conclude that $|N_{R_k}(A^k[\mathcal{S}_1])| = |\mathbb{Z}_2^k| - |\langle \mathcal{S} \rangle^\perp| = 2^k - 2^{k-|\mathcal{S}|}$. ◀

4 Reduction for Independent Set

Our reduction is from the variant of 3-CNF-SAT with a square number of variables. Since, for every $n \in \mathbb{N}$, there is always a square number between n and $2n+1$, we can always add $O(n)$ dummy variables to an instance of 3-CNF-SAT to ensure a square number of variables. Thus, we have the following easy consequence of ETH [22].

▶ **Lemma 4.1.** *Unless ETH fails, there is no $2^{o(k^2)}(k+m)^{O(1)}$ time algorithm for 3-CNF-SAT with m clauses and k^2 variables, where k is an integer.*

We start from an instance φ of 3-CNF-SAT with m clauses $C_1, \ldots, C_m$ and a set of k^2 variables $\mathsf{Var}(\varphi) := \{v_{i,j} : i \in [k], j \in [k+1, 2k]\}$. The main idea of our construction is to use the following bijection between the assignments of $\mathsf{Var}(\varphi)$ and some collections in $\mathscr{F}_{2k}$.

▶ **Definition 4.2.** *Let $\mathcal{S} := \{s \subseteq [2k] : |s \cap [k]| = 1\}$. For each $i \in [k]$, we denote by $\mathcal{S}_i$ the set $\{s \in \mathcal{S} : s \cap [k] = \{i\}\}$. For every assignment $f : \mathsf{Var}(\varphi) \to \{0,1\}$, we denote by $\mathcal{S}_f$ the collection containing the sets $s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2, \ldots, s_k \in \mathcal{S}_k$, where for every $i \in [k]$*

$$s_i = \{i\} \cup \{j \in [k+1, 2k] : f(v_{i,j}) = 1\}.$$

Given a literal $\ell \in \{v_{i,j}, \neg v_{i,j}\}$, we define $\mathcal{S}_\ell$ as the set $\{s \in \mathcal{S}_i : j \in s\}$ if $\ell = v_{i,j}$ and $\{s \in \mathcal{S}_i : j \notin s\}$ if $\ell = \neg v_{i,j}$.

For example with $k = 2$, the interpretation f which sets the variables $v_{1,3}, v_{1,4}$ and $v_{2,4}$ to true is associated with the collection $\mathcal{S}_f$ containing $\{1, 3, 4\}$ and $\{2, 4\}$.

▶ **Observation 4.3.** *For every interpretation $f : \mathsf{Var}(\varphi) \to \{0, 1\}$ and literal $\ell \in \{v_{i,j}, \neg v_{i,j} : v_{i,j} \in \mathsf{Var}(\varphi)\}$, we have $f(\ell) = 1$ if and only if $\mathcal{S}_f \cap \mathcal{S}_\ell \neq \emptyset$ if and only if $\mathcal{S}_f \cap \mathcal{S}_{\neg\ell} = \emptyset$.*

High level description of the reduction. We consider a modified version of the universal $2k$-rank cut R_{2k} which is obtained by: (1) removing the vertices that are in A^{2k} but not in $A^{2k}[\mathcal{S}]$ and (2) making $A^{2k}[\mathcal{S}_i]$ a clique for every $i \in [k]$. This way in the graph induced by $A^{2k}[\mathcal{S}]$, every maximal independent set is of the form $A^{2k}[\mathcal{S}_f]$ with f an assignment of $\mathsf{Var}(\varphi)$.

We make m copies $A_1, \ldots, A_m$ of such $A^{2k}[\mathcal{S}]$ and for each $i \in [m]$, we create a simple clause gadget that is adjacent to some vertices of A_i. For each $i \in [m]$, the clause gadget for C_i is simply a triangle whose vertices are associated with the literals of C_i. For each literal ℓ of C_i, the vertex associated with ℓ is adjacent to the vertices in the A_i associated with the sets in $\mathcal{S}_{\neg\ell}$. See Figure 2 for an overview of this construction and Figure 3 for an example of clause gadget.

Finally, we define a vertex-weight function (that can be emulated in the unweighted setting by adding false twins) in such a way that for any independent set I of maximum weight, there exists an interpretation f of $\mathsf{Var}(\varphi)$ such that I contains the copies of $A^{2k}[\mathcal{S}_f]$. To this end, we actively use the fact that for all interpretations f, g of $\mathsf{Var}(\varphi)$, we have $\mathcal{S}_f, \mathcal{S}_g \in \mathscr{F}_{2k}$ and thus $A^{2k}[\mathcal{S}_f]$ and $A^{2k}[\mathcal{S}_g]$ have the same neighborhood in R_{2k} if and only if $f = g$.

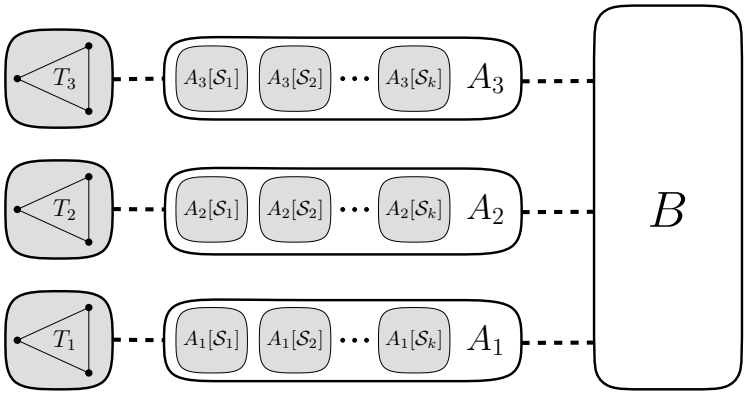


Figure 2 Overview of the reduction for Independent Set with $m = 3$. The gray areas represent cliques and dotted lines indicates the existence of edges between two sets of vertices.

The construction. We construct a graph G as follows. We create m copies $A_1, \ldots, A_m$ of $A^{2k}[\mathcal{S}]$, for each $i \in [m]$, we have $A_i := \{a_s^i : s \in \mathcal{S}\}$. Given $\mathcal{S}' \subseteq \mathcal{S}$ and $i \in [m]$, we denote by $A_i[\mathcal{S}']$ the set $\{a_s^i \in A_i : s \in \mathcal{S}'\}$. For each $i \in [m]$ and $j \in [k]$, we add edges so that $A_i[\mathcal{S}_j]$ induces a clique. Thanks to these k cliques, we have the following relation between the independent sets of $G[A_i]$, the subsets of $\mathcal{S}$ and the interpretations of $\mathsf{Var}(\varphi)$.

▶ **Lemma 4.4.** *Let $i \in [m]$ and $\mathcal{S}' \subseteq \mathcal{S}$. If $A_i[\mathcal{S}']$ is an independent set of $G[A_i]$, then $\mathcal{S}' \in \mathscr{F}_{2k}$. Moreover, $A_i[\mathcal{S}']$ is an independent set of $G[A_i]$ of size k if and only if there exists an interpretation f of $\mathsf{Var}(\varphi)$ such that $\mathcal{S}' = \mathcal{S}_f$.*

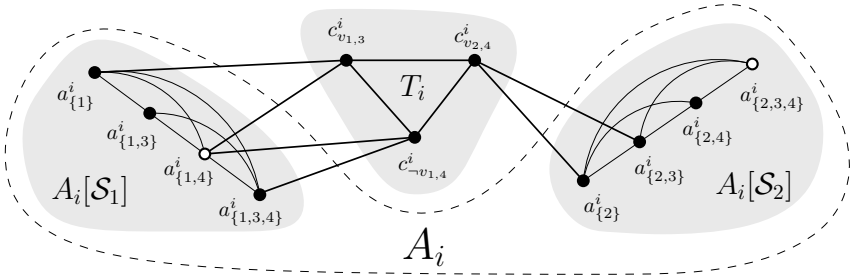


Figure 3 Example of clause gadget with $k = 2$ and a clause $C_i = \{v_{1,3}, \neg v_{1,4}, v_{2,4}\}$. The independent set containing the white-filled vertices is $A_i[\mathcal{S}_f]$ for the interpretation f with $f^{-1}(1) = \{v_{1,4}, v_{2,3}, v_{2,4}\}$ and $f^{-1}(0) = \{v_{1,3}\}$. This independent set can be extended with the vertex $c^i_{v_{2,4}}$.

Proof. Let $i \in [m]$ and $\mathcal{S}' = \{s_1, \ldots, s_r\} \subseteq \mathcal{S}$ such that $A_i[\mathcal{S}']$ is an independent set of G. By construction of $G[A_i]$, we know that $A_i[\mathcal{S}']$ contains at most one vertex from $A_i[\mathcal{S}_j]$ for each $j \in [k]$. Thus, there exist pairwise distinct integers $\alpha_1, \ldots, \alpha_r \in [k]$ such that for every $j \in [r]$, we have $\{\alpha_j\} \subseteq s_j \subseteq \{\alpha_j\} \cup [k+1, 2k]$. By Observation 3.4, we deduce that $\mathcal{S}'$ belongs to the collection $\mathscr{F}_{2k}$.

It is easy to see that $|\mathcal{S}'| = k$ if and only if $\mathcal{S}' = \{s_1, \ldots, s_k\}$ with $\{j\} \subseteq s_j \subseteq \{j\} \cup [k+1, 2k]$ for every $j \in [k]$ and this is equivalent to $\mathcal{S}' = \mathcal{S}_f$ with $f^{-1}(1) = \{v_{i,j} : i \in [k] \wedge j \in s_i \cap [k+1, 2k]\}$. ◀

We create $B = B^{2k} := \{b_s : s \subseteq [2k]\}$. For every $i \in [m]$, $s \in \mathcal{S}$ and $t \subseteq [2k]$, if $|s \cap t|$ is odd, we make b_t adjacent to a^i_s. Consequently, $G[A_i, B_i]$ is isomorphic to $R_{2k}[A^{2k}[\mathcal{S}], B^{2k}]$. We deduce the following observations from Lemma 4.4 and our results on R_{2k}.

▶ **Observation 4.5.** *Let $i, j \in [m]$ and f, g be two assignments of $\mathsf{Var}(\varphi)$. We have $N(A_i[\mathcal{S}_f]) \cap B = N(A_j[\mathcal{S}_g]) \cap B$ if and only if $f = g$.*

▶ **Observation 4.6.** *For every $i \in [m]$ and $\mathcal{S}' \subseteq \mathcal{S}$ such that $A_i[\mathcal{S}']$ is an independent set of G, we have $|N(A_i[\mathcal{S}']) \cap B| = 2^{2k} - 2^{2k-|\mathcal{S}'|}$.*

For every $i \in [m]$ with $C_i = \{\ell_1, \ell_2, \ell_3\}$, we create a triangle induced by a set T_i of three new vertices $c^i_{\ell_1}, c^i_{\ell_2}$ and $c^i_{\ell_3}$. For each $j \in [3]$, we make $c^i_{\ell_j}$ adjacent to all the vertices in $A_i[\mathcal{S}_{\neg \ell_j}]$. Thanks to Observation 4.3 and these edges, each independent sets associated with an interpretation that satisfies C_i can be extended with one of the vertices in T_i.

▶ **Observation 4.7.** *For every assignment f of $\mathsf{Var}(\varphi)$ and $i \in [m]$ with $C_i = \{\ell_1, \ell_2, \ell_3\}$, we have $T_i \setminus N(A_i[\mathcal{S}_f]) \neq \emptyset$ if and only if f satisfies C_i.*

Finally, we define the weight function $w : V(G) \to \mathbb{N}$ such each vertex v in $A_1 \cup \cdots \cup A_m$ have weight $w(v) := 2^{2k} = |B|$ and all the other vertices have weight 1. The purpose of w is to guarantee that maximum independent set of G contains k vertices in each A_i.

We are ready to prove the correctness of our reduction.

▶ **Lemma 4.8.** *If φ is a satisfiable 3-CNF-SAT formula, then G admits an independent set of weight $2^{2k}km + 2^k + m$.*

Proof. Suppose φ admits a satisfying assignment f. For each $i \in [m]$, let ℓ^i be a literal of C_i such that $f(\ell^i) = 1$. Let I be the set of vertices that contains (1) the vertices in $B \setminus N(A_1[\mathcal{S}_f]) = \cdots = B \setminus N(A_m[\mathcal{S}_f])$ and (2) for every $i \in [m]$ the vertices in $A_i[\mathcal{S}_f] \cup \{c^i_{\ell^i}\}$.

Lemmas 4.4 and 4.6 and Observations 4.7 and 4.5 imply that I is an independent set of G. For every $i \in [m]$, we have $|A_i \cap I| = k$ and by Observation 4.6 we have $B \cap I = |B \setminus N(A_1[\mathcal{S}_f])| = 2^k$. Hence, the weight of I is $2^{2k}km + 2^k + m$. ◀

▶ **Lemma 4.9.** *If G admits an independent set of weight at least $2^{2k}km + 2^k + m$, then φ is a satisfiable 3-CNF-SAT formula.*

Proof. Assume that G admits an independent set I of maximum weight with $w(I) \geq 2^{2k}km+2^k+m$. First, we assume towards a contradiction that there exists $i \in [m]$ and $j \in [k]$ such that $I \cap A_i[\mathcal{S}_j] = \emptyset$. Let $s \in \mathcal{S}_j$. By construction of G, we have $N(a^i_s) \subseteq B \cup A_i[\mathcal{S}_j] \cup T_i$. By Observation 4.6, we have $|N(a^i_s) \cap B| = 2^{2k-1}$. Moreover, since T_i induces a triangle in G, I contains at most one of vertex in T_i. We deduce that

$$w(N(a^i_s) \cap I) \leqslant 2^{2k-1} + 1 < w(a^i_s) = 2^{2k}.$$

Consequently, $I' = (I \setminus N(a^i_s)) \cup \{a^i_s\}$ is an independent set of G with $w(I) < w(I')$, yielding a contradiction with I being of maximum weight.

From now, we assume that, for every $i \in [m]$ and $j \in [k]$, I contains exactly one vertex in $A_i[\mathcal{S}_j]$. Since each $A_i[\mathcal{S}_j]$ induces a clique, we deduce that for every $i \in [m]$, we have $|A_i \cap I| = k$. By Lemma 4.4, there exist m interpretations $f_1, \ldots, f_m$ of $\mathsf{Var}(\varphi)$ such that, for every $i \in [m]$, we have $I \cap A_i = A_i[\mathcal{S}_{f_i}]$. By Observation 4.6, for every $i \in [m]$, we have

$$|N(A_i[\mathcal{S}_{f_i}]) \cap B| = 2^{2k} - 2^k. \tag{1}$$

Hence, we have $w(I \cap B) = |I \cap B| \leqslant 2^k$. Since each T_i induces a triangle in G, we have $w(I \cap T_i) = |I \cap T_i| \leqslant 1$. As $w(I) \geq 2^{2k}km + 2^k + m$, we deduce that I has exactly 2^k vertices in B and 1 in each T_j for $j \in [m]$.

Equation 1 and $|I \cap B_i| = 2^k$ imply that the neighborhoods of $I \cap A_1, I \cap A_2, \ldots, I \cap A_m$ in B is the same. From Observation 4.5, we deduce that $f_1 = f_2 = \cdots = f_m$. Since I contains exactly one vertex in each T_i, we conclude from Observation 4.7 that $f_1 = \cdots = f_m$ satisfies every clause of φ. ◀

▶ **Lemma 4.10.** *The linear rank-width of G is at most $2k + 4$.*

Proof. For each $i \in [m]$ and $j \in [k]$, let $\sigma(A_i[\mathcal{S}_j])$ be an arbitrary permutation of $A_i[\mathcal{S}_j]$ and $\sigma(A_i)$ be the concatenation of $\sigma(A_i[\mathcal{S}_1]), \ldots, \sigma(A_i[\mathcal{S}_k])$. For each $X \in \{T_1, \ldots, T_m\} \cup \{B\}$, let $\sigma(X)$ be an arbitrary permutation of X. We define the permutation σ of $V(G)$ as the concatenation of $\sigma(B), \sigma(A_1), \sigma(T_1), \sigma(A_2)$, $\sigma(T_2), \ldots, \sigma(A_m)$ and $\sigma(T_m)$. We claim that $\mathsf{rw}(\sigma) \leqslant 2k + 4$.

Let $(X, \overline{X})$ be a cut of G induced by σ. If $X \cap (A_1 \cup \cdots \cup A_m) = 0$, then $\mathsf{rw}(X, \overline{X})$ is at most $\mathsf{rw}(B, A_1 \cup \cdots \cup A_m)$. Now, observe that $G[B, A_1 \cup \cdots \cup A_m]$ is obtained from the universal $2k$-rank cut R_{2k} by removing some vertices A^{2k} and making copies of the ones we do not remove. Consequently, we have $\mathsf{rw}(B, A_1 \cup \cdots \cup A_m) \leqslant 2k$.

Suppose now that $X \cap (A_1 \cup \cdots \cup A_m) \neq \emptyset$. Let $i \in [m]$ and $j \in [k]$ be maximum such that $X \cap A_i[\mathcal{S}_j] \neq \emptyset$. Observe that the only edges of $G[X, \overline{X}]$ are (1) between $X \cap B$ and $\overline{X} \cap (A_{j+1} \cup \cdots \cup A_m)$, (2) between $A_i[\mathcal{S}_j] \cap X$ and $A_i[\mathcal{S}_j] \cap \overline{X}$, (3) between $X \cap (A_i \cup T_i)$ and $\overline{X} \cap T_i$. As $G[A_i]$ is a clique, we have $\mathsf{rw}(A_i[\mathcal{S}_j] \cap X, A_i[\mathcal{S}_j] \cap \overline{X}) \leqslant 1$. Since $|\overline{X} \cap T_i| \leqslant 3$, we deduce that $\mathsf{rw}(X, \overline{X})$ is at most $2k + 4$. As it holds for any prefix X of σ, we conclude that $\mathsf{lrw}(G) \leqslant 2k + 4$. ◀

▶ **Theorem 4.11.** *There is no algorithm solving Weighted Independent Set in time $2^{o(\mathsf{lrw}(G)^2)}n^{O(1)}$ unless ETH fails.*

Proof. Assume that there exists a $2^{o(\mathsf{lrw}(G)^2)}n^{O(1)}$ time algorithm for Weighted Independent Set. We prove that it implies the existence of a $2^{o(k^2)}n^{O(1)}$ time algorithm for 3-SAT where k^2 is the number of variables. This will contradict ETH.

Suppose that we are given a 3-SAT formula φ with k^2 variables and m clauses. We construct the graph G described above. As G has $2^{2k} + (2^k k + 3)m$ vertices, we deduce that we can construct G in time $2^{O(k)}m$.

From Lemmas 4.8 and 4.9, we know that G admits an independent set of weight at least $2^{2k}km + 2^k + m$ if and only if φ is satisfiable. By assumption, we can compute an independent set of G in time $2^{o(\mathsf{lrw}(G)^2)}n^{O(1)}$. By Lemma 4.10, the linear rank-width of G is at most $2k + 4$. Hence, we can decide whether φ is satisfiable in time $2^{o(k^2)}n^{O(1)}$. This contradicts ETH by Lemma 4.1. ◀

▶ **Lemma 4.12.** *Let G be a graph with a weight function $w : V(G) \to \mathbb{N}$ and σ be a linear decomposition of rank-width w. We can construct in time $O(|V(G)| \max_{v \in V(G)} w(v))$ a graph G' and a linear decomposition σ' of G' with rank-width at most $w + 1$ such that G admits an independent set I of weight at least W iff G' admits a independent set of size at least W.*

Proof. We assume without loss of generality that G has no vertex of weight 0 (we can always delete them without changing the weights of the independents sets of G). The graph G' is obtained from G by adding iteratively $w(v) - 1$ false twins to each vertex $v \in V(G)$. Formally, G' is the graph with vertex set $V(G') = \{v_i : v \in V(G) \wedge i \in [w(v)]\}$ and edge set $\{u_i v_j : uv \in E(G) \wedge i \in [w(u)] \wedge j \in [w(v)]\}$. We construct σ' from σ by replacing every vertex $v \in V(G)$ by the sequence $(v_1, \ldots, v_{w(v)})$. Obviously, G' and σ' can be constructed in time $O(|V(G)| \max_{v \in V(G)} w(v))$

Given an independent set I of G, it is easy to see that $\{v_i : v \in I \wedge i \in [w(v)]\}$ is an independent set of G' of size $w(I)$. On the other hand, for every independent set I' of G', we have $|I'| \leqslant w(\{v \in V(G) : I \cap \{v_1, \ldots, v_{w(v)}\} \neq \emptyset\})$.

Let (A', B') be a cut induced by σ', $A = \{v \in V(G) : v_1 \in A'\}$ and $B = \{v \in V(G) : v_{w(v)} \in B'\}$. By construction, $(A, V(G) \setminus A)$ is a cut of G induced by σ and B is either $V(G) \setminus A$ or $(V(G) \setminus A) \cup \{v\}$ for some vertex $v \in A$ if $v_1 \in A'$ and $v_{w(v)} \in B'$. Moreover, the adjacency matrix between A' and B' in G' can be obtained from the one between A and B in G by adding copies of rows and columns. Since adding a copy of a row or a column does not increase the rank of a matrix, we conclude that $\mathsf{rw}(A', B') \leqslant \mathsf{rw}(A, B) + \mathsf{rw}(A, \{v\}) \leqslant \mathsf{rw}(A, V(G) \setminus A) + 1$. As $\mathsf{rw}(A, V(G)) \leqslant w$, we deduce that $\mathsf{rw}(A', B') \leqslant w + 1$. We conclude that the rank-width of σ' is at most $w + 1$. ◀

Theorem 1.1 is now a direct consequence of Theorem 4.11 and Lemma 4.12.

5 Maximum Induced Matching and Feedback Vertex Set

In this subsection, we prove that our lower bound for Independent Set holds also for Maximum Induced Matching and Feedback Vertex Set. To prove this, we provide a single reduction from Independent Set that works for both problems.

▶ **Lemma 5.1.** *Let G be a graph, σ be a linear decomposition of G of rank-width w. We can construct in polynomial time a graph G' and a linear decomposition of G' of rank-width at most $w + 1$ such that, for every $k \in \mathbb{N}$, the following properties are equivalent:*

1. *G admits an independent set of size k.*
2. *G' admits an induced matching on k edges.*
3. *G admits an induced forest with $2k$ vertices.*

Proof. Let G' be the graph with vertex set $\{v, \widehat{v} : v \in V(G)\}$ and edge set

$$\{v\widehat{v} : v \in V(G)\} \cup \{uv, \widehat{u}v, u\widehat{v}, \widehat{u}\widehat{v} : uv \in E(G)\}.$$

We construct a linear decomposition σ' of G' from σ by inserting $\widehat{v}$ after v in σ for each vertex $v \in V(G)$. Obviously, G' and σ' can be constructed in polynomial time. Since $\widehat{v}$ is a true twin of v in G' for every $v \in V(G)$, we deduce that the rank-width of σ' is at most $w + 1$.

($1 \Rightarrow 2 \wedge 3$). Given an independent set I of G, the set of vertices $\{v\widehat{v} : v \in I\}$ induces a matching (and a forest) of G'.

($2 \Rightarrow 1$). For M an induced matching M of G', we obtain an independent set of G of size $|M|$ by considering any set of vertices that contains exactly one endpoint in $V(G)$ of each edge in M.

($3 \Rightarrow 1$). Let F be a forest of G' of maximum size with a maximum number of edges in $\widehat{E} := \{v\widehat{v} : v \in V(G)\}$. Assume towards a contradiction that F admits a connected component C with an edge not in $\widehat{E}$. By construction, we deduce that $V(C)$ contain at most one vertices in $\{v, \widehat{v}\}$ for each $v \in V(G)$. As C is a tree, there exists $I_C \subseteq C$ such that $2|I_C| \geq |C|$ and $G'[I_C]$ is an independent set. Observe that $M = \{v, \widehat{v} : v \in I_C \vee \widehat{v} \in I_C\}$ induces a matching of size at least $|C|$. It follows that $(V(F) \setminus V(C)) \cup M$ induces a forest with at least as much vertices as F and with more edges in $\widehat{E}$, yielding a contradiction. Consequently, we have $E(F) \subseteq \widehat{E}$ and we conclude that $V(F) \cap V(G)$ is an independent set of size $|V(F)/2|$. ◀

The following corollary is a direct consequence of Theorem 1.1 and Lemma 5.1.

▶ **Corollary 5.2.** *There is no algorithm solving* Maximum Induced Matching *or* Feedback Vertex Set *in time* $2^{o(\mathsf{rw}(G)^2)}n^{O(1)}$ *unless ETH fails.*

6 Weighted Dominating Set

As for Independent Set, the starting point is an instance φ of 3-CNF-SAT with m clauses $C_1, \ldots, C_m$ and a set of k^2 variables $\mathsf{Var}(\varphi) := \{v_{i,j} : i \in [k], j \in [k+1, 2k]\}$. Similarly to Independent Set, we construct a graph G with m copies $A_1, \ldots, A_m$ of $A^{2k}[\mathcal{S}]$ and we make sure that for any dominating set of minimum weight D, the intersection of D with $A_1, \ldots, A_m$ corresponds to $A_1[\mathcal{S}_f], \ldots, A_m[\mathcal{S}_f]$ for some interpretation f of $\mathsf{Var}(\varphi)$. The clause gadget associated with C_i consists of a single vertex adjacent to the vertices of the i-th copy of $A^{2k}[\mathcal{S}]$ that represent the partial interpretations of $\mathsf{Var}(\varphi)$ satisfying C_i. Thus, if φ is satisfied by an interpretation f, a dominating set including $A_1[\mathcal{S}_f], \ldots, A_m[\mathcal{S}_f]$ would dominate the vertices associated with the clause gadgets.

The main difference with our reduction for Independent Set is that we need $2(m-1)$ copies $B_1, \widehat{B}_1, \ldots, B_{m-1}, \widehat{B}_{m-1}$ of B^{2k} to guarantee our equivalence between minimum dominating sets and interpretations of $\mathsf{Var}(\varphi)$. Briefly, for each $i \in [m-1]$, $G[A_i, B_i]$ and $G[\widehat{B}_i, A_{i+1}]$ are isomorphic to $R_{2k}[A^{2k}[\mathcal{S}], B^{2k}]$ and $G[B_i, \widehat{B}_i]$ is an induced matching such that for each set $s \subseteq [2k]$, the two vertices in $B_i \cup \widehat{B}_i$ associated with s are adjacent. See Figure 4 for an overview of the reduction.

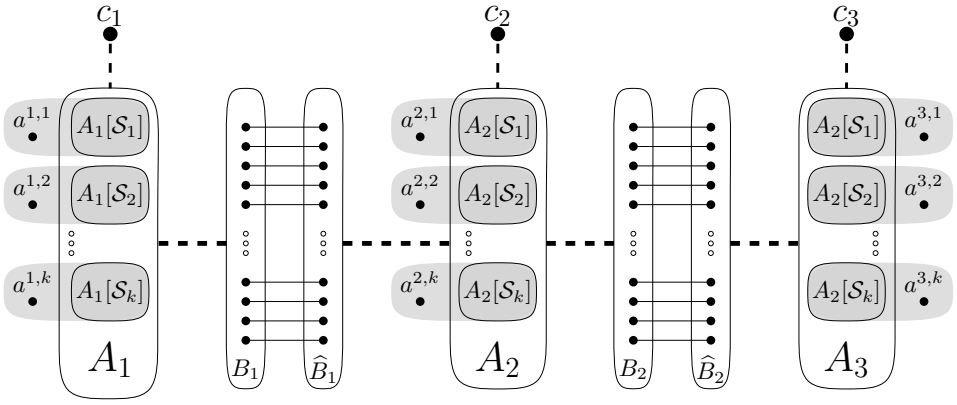


Figure 4 Overview of the reduction for Weighted Dominating Set with $m = 3$. The gray areas represents cliques and dotted lines indicates the existence of edges between two sets of vertices.

This path-shaped construction and prohibitive weights on some vertices ensure that φ is satisfiable by an interpretation f iff the set containing $A_1[\mathcal{S}_f], \ldots, A_m[\mathcal{S}_f]$ and $B_1 \setminus N(A_1[\mathcal{S}_f]), \ldots, B_{m-1} \setminus N(A_m[\mathcal{S}_f])$ is a dominating set of minimum weight. Thanks to the induced matchings between B_i and $\widehat{B}_i$, the vertices of $B_i \setminus N(A_i[\mathcal{S}_f])$ dominate the vertices in $\widehat{B}_i$ that are not dominated by $A_{i+1}[\mathcal{S}_f]$.

The construction. We construct a graph G as follows. We create m copies $A_1, \ldots, A_m$ of $A^{2k}[\mathcal{S}]$, for each $i \in [m]$, we have $A_i := \{a^i_s : s \in \mathcal{S}\}$. For each $i \in [m]$ and $j \in [k]$, we create a vertex $a^{i,j}$ and we add edges so that $A_i[\mathcal{S}_j] \cup \{a^{i,j}\}$ induces a clique denoted by $K_{i,j}$. As the vertex $a^{i,j}$ will be only adjacent to the vertex in $A_i[\mathcal{S}_j]$. This guarantee that any dominating set of G contains at least one vertex in $K_{i,j}$.

We create $2(m-1)$ copies $B_1, \widehat{B}_1, \ldots, B_{m-1}, \widehat{B}_{m-1}$ of B^{2k}, for each $i \in [m-1]$, we have $B_i := \{b^i_s : s \subset [2k]\}$ and $\widehat{B}_i := \{\widehat{b}^i_s : s \subseteq [2k]\}$. For every $i \in [m-1]$, $s \in \mathcal{S}$ and $t \subseteq [2k]$ such that $|s \cap t|$ is odd, we make (1) a^i_s adjacent to b^i_t, (2) $\widehat{b}^i_t$ adjacent to a^{i+1}_s and (3) b^i_t adjacent to $\widehat{b}^i_t$. Consequently, $G[A_i, B_i]$ and $G[\widehat{B}_i, A_{i+1}]$ are both isomorphic to $R_{2k}[A^{2k}[\mathcal{S}], B^{2k}]$ and $G[B_i, \widehat{B}_i]$ is an induced perfect matching.

For every $i \in [m]$ with $\widehat{B}_i = \{\ell_1, \ell_2, \ell_3\}$, we create a vertex c_i adjacent to the vertices in $A_i[\mathcal{S}_{\ell_1} \cup \mathcal{S}_{\ell_2} \cup \mathcal{S}_{\ell_3}]$.

Finally, we define the weight function $w : V(G) \to \mathbb{N}$ such each vertex $v \in B_1 \cup \cdots \cup B_{m-1} \cup \{c_i : i \in [m]\}$ has weight $w(v) := 1$, each vertex $u \in A_1 \cup \cdots \cup A_m$ has weight $w(u) := 2^{2k} + 2$ and every vertex $x \in \widehat{B}_1 \cup \cdots \cup \widehat{B}_{m-1} \cup \{a^{i,j} : i \in [m] \wedge j \in [k]\}$ has weight $w(x) := +\infty$. The purpose of w is to guarantee that every minimum dominating set of G contains at most 1 vertices in each $A_i[\mathcal{S}_j]$ and no vertex in $\widehat{B}_1 \ldots \widehat{B}_{m-1} \cup \{a^{i,j} : i \in [m] \wedge j \in [k]\}$.

▶ **Lemma 6.1.** *If φ is satisfiable, then G admits a dominating set of weight $(2^{2k}+2)km + 2^k(m-1)$.*

Proof. Suppose that φ is satisfied by an interpretation f. Let D be the union of $A_i[\mathcal{S}_f]$ and $B_j \setminus N(A_j[\mathcal{S}_f])$ for every $i \in [m]$ and $j \in [m-1]$. We claim that D is a dominating set of weight $(2^{2k}+2)km + 2^k(m-1)$. The weight of D is deductible from the following observations:

- For each $i \in [m]$, the weight of each vertex in A_i is $2^{2k}+2$ and $|A_i[\mathcal{S}_f]| = k$.
- The weight of each vertex in $B_1 \cup \cdots \cup B_{m-1}$ is 1, and by Observation 3.4, we have $\mathcal{S}_f \in \mathscr{F}_{2k}$ which implies with Lemma 3.7 that $|B_j \setminus N(A_j[\mathcal{S}_f])| = 2^k$ for each $j \in [m-1]$.

We conclude that D is a dominating set of G from the following arguments:

- Let $i \in [m]$. By definition, $A_i[\mathcal{S}_f]$ contains one vertex in $A_i[\mathcal{S}_j]$ for every $j \in [k]$. As $K_{i,j} = A_i[\mathcal{S}_j] \cup a^{i,j}$ is a clique, we deduce that D dominates A_i and $\{a^{i,j} : j \in [k]\}$. Moreover, f interprets at least one literal ℓ of C_i as true. Thus, $A_i[\mathcal{S}_f] \cap A_i[\mathcal{S}_\ell] \neq \emptyset$ and c_i – the vertex representing C_i – has a neighbor in D. So, D dominates $A_i \cup \{c_i\} \cup \{a^{i,j} : j \in [k]\}$ for every $i \in [m]$.
- Let $j \in [m-1]$. As $A_j[\mathcal{S}_f]$ and $B_j \setminus N(A_j[\mathcal{S}_f])$ are included in D, we know that D dominates B_j. Let $\widehat{b}^j_s$ be a vertex in $\widehat{B}_j$. If $b^j_s \in B_j$ is in D, then $\widehat{b}^j_s$ is dominated by D as $b^j_s\widehat{b}^j_s$ is an edge of G. Otherwise, if b^j_s is not in D, then b^j_s is adjacent to a vertex a^j_t in $A_j[\mathcal{S}_f] \subseteq D$. In this later case, the vertex a^{j+1}_t belongs to $A_{j+1}[\mathcal{S}_f] \subseteq D$ and a^{j+1}_t is adjacent to $\widehat{b}^j_s$. It follows that D dominates $\widehat{B}_i$. ◀

▶ **Lemma 6.2.** *If G admits a dominating set of weight at most $(2^{2k}+2)km + 2^k(m-1)$, then $\mathsf{Var}(\varphi)$ is satisfiable.*

Proof. Let D be a dominating set of weight at most $(2^{2k}+2)km + 2^k(m-1)$. We use the following claim prove that there exists some interpretation f such that D is the union of $A_1[\mathcal{S}_f], \ldots, A_m[\mathcal{S}_f]$ and $B_1 \setminus N(A_1[\mathcal{S}_f]), \ldots, B_m \setminus N(A_m[\mathcal{S}_f])$.

▷ Claim 6.3. For every $i \in [m]$, there exists an interpretation f_i of $\mathsf{Var}(\varphi)$ such that $A_i \cap D = A_i[\mathcal{S}_f]$.

Proof. By Definition 4.2, it is sufficient to prove that D contains exactly one vertex in $A_i[\mathcal{S}_j]$ for every $i \in [m]$ and $j \in [k]$. Let $i \in [m]$ and $j \in [k]$. By construction, we have $N(a^{i,j}) = A_i[\mathcal{S}_j]$. Since $a^{i,j}$ must be dominated by D and $w(a^{i,j}) = +\infty$, D contain at least one vertex in $A_i[\mathcal{S}_j]$.

Assume towards a contradiction that D contains two different vertices u, v in $A_i[\mathcal{S}_j]$. By construction, we have $N(u) \setminus N(v) \subseteq \widehat{B}_{i-1} \cup B_i \cup \{c_i\}$ (we consider that $B_0 = \widehat{B}_0 = B_m = \emptyset$) and the vertices in $\{c_i\} \cup \widehat{B}_{i-1} \cup B_i$ are dominated by the set $X := B_{i-1} \cup B_i \cup \{c_i\}$. Thus, $(D \setminus \{u\}) \cup X$ is a dominating set of G. But, as $w(X) = |X| \leqslant 2^{2k}+1 < w(u) = 2^{2k}+2$, this contradicts D being a dominating set of minimum weight. We conclude that $D \cap A_i = A_i[\mathcal{S}_{f_i}]$ for some interpretation f_i of $\mathsf{Var}(\varphi)$. ◁

For every $i \in [m-1]$, we denote by $B^\star_i$ the vertices in B_i that are not dominated by $A_i[\mathcal{S}_{f_i}]$, i.e. $B^\star_i := B_i \setminus N(A_i[\mathcal{S}_{f_i}])$. Similarly, we denote by $\widehat{B}^\star_i$ the vertices in $\widehat{B}_i$ not dominated by $A_{i+1}[\mathcal{S}_{f_{i+1}}]$. By Observation 3.4, we have $\mathcal{S}_{f_1}, \ldots, \mathcal{S}_{f_m} \in \mathscr{F}_{2k}$ and from Lemma 3.7, we deduce that $|B^\star_i| = |\widehat{B}^\star_i| = 2^k$. By construction, each $B^\star_i$ is an independent set and $N(B^\star_i)$ is included in $A_i \cup \widehat{B}_i$. As $A_i \cap D = A_i[\mathcal{S}_{f_i}]$ and $w(v) = +\infty$ for every $v \in \widehat{B}_i$, we deduce that D contains $B^\star_i$ for every $i \in [m-1]$.

Observe that $w(A_i[\mathcal{S}_{f_i}]) = (2^{2k}+2)k$ for every $i \in [m]$ and $w(B^\star_j) = |B^\star_j| = 2^k$ for each $j \in [m-1]$. As $w(D)$ is at most $(2^{2k}+2)km + 2^k(m-1)$, we deduce that D is exactly the union of $A_i[\mathcal{S}_i]$ and $B^\star_j$ for $i \in [m]$ and $j \in [m-1]$.

By definition, for each $i \in [m-1]$, the vertices in $\widehat{B}^\star_i$ are not dominated by $A_{i+1}[\mathcal{S}_{f_{i+1}}]$ and thus they must be dominated by $B^\star_i$. As $G[B_i, \widehat{B}_i]$ is an induced perfect matching with set of edges $\{b^i_s\widehat{b}^i_s : s \subseteq [2k]\}$, we deduce that $\widehat{B}^\star_i = \{\widehat{b}^i_s : b^i_s \in B^\star_i\}$. This implies that $N(A_i[\mathcal{S}_{f_i}]) = N(A_i[\mathcal{S}_{f_{i+1}}])$. Since f_i and f_{i+1} belong to $\mathscr{F}_{2k}$, by Lemma 3.5, it follows that $f_i = f_{i+1}$. We deduce that $f_1 = f_2 = \cdots = f_m$. We conclude that the interpretation $f_1 = \cdots = f_m$ satisfies φ because for every $i \in [m]$, the vertex c_i representing the clause C_i is dominated by $A_i[\mathcal{S}_{f_i}]$ and by Observation 4.3 it implies that f_i satisfies C_i. ◀

▶ **Lemma 6.4.** *We can compute in polynomial time a linear decomposition of G with rank-width at most $4k+2$.*

Proof. For every $X \in \{K_{i,j} : i \in [m] \wedge j \in [k]\}$, let $\sigma(X)$ be an arbitrary permutation of X. For every $i \in [m]$, we define the permutation $\sigma(A_i)$ of $A_i \cup \{c_i\} \cup \{a^{i,j} : j \in [k]\}$ as the concatenation of the permutation $(c_i), \sigma(K_{i,1}), \sigma(K_{i,2}), \ldots, \sigma(K_{i,k-1})$ and $\sigma(K_{i,k})$. Let $(s_1, \ldots, s_t)$ be a permutation of $2^{[2k]}$, for every $i \in [m-1]$, we define $\sigma(B_i \cup \widehat{B}_i)$ as the permutation $(b^i_{s_1}, \widehat{b}^i_{s_1}, b^i_{s_2}, \ldots, \widehat{b}^i_{s_{t-1}}, b^i_{s_t}, \widehat{b}^i_{s_t})$. Let σ be the concatenation of $\sigma(A_1), \sigma(B_1 \cup \widehat{B}_1), \sigma(A_2), \ldots, \sigma(B_{m-1} \cup \widehat{B}_{m-1})$ and $\sigma(A_m)$.

Obviously, σ is a linear decomposition of G that can be computed in polynomial time. We claim that the rank-width of σ is at most $4k+2$. Let $(X, \overline{X})$ be a cut of G induced by σ. We distinguish the following cases:

- Suppose that there exists $i \in [m]$ such that X intersect $A_i \cup \{c_i\} \cup \{a^{i,j} : j \in [k]\}$ but not B_i (we consider that $B_m = \emptyset$). The edges of $G[X, \overline{X}]$ belong to the following cuts: (1) the cut between $\widehat{B}_{i-1}$ and $A_i \cap \overline{X}$, (2) the cut between $A_i \cap X$ and B_i, (3) the cut between c_i and $A_i \cap \overline{X}$ and (4) the cut between $K_{i,j} \cap X$ and $K_{i,j} \cap \overline{X}$ with $j = \max\{\ell \in [k] : K_{i,\ell} \cap X \neq \emptyset\}$.
  The ranks of the first two cuts are upper bounded by $\mathsf{rw}(A_i, \widehat{B}_{i-1})$ and $\mathsf{rw}(A_i, B_i)$ respectively, since $G[A_i, \widehat{B}_{i-1}]$ and $G[A_i, B_i]$ are isomorphic to R_{2k}, these ranks are at most $2k$. The rank of the third cut is upper bounded by 1 since one side consists of a single vertex. Since the fourth cut is a biclique, its rank is at most 1. We deduce that $\mathsf{rw}(X, \overline{X}) \leqslant 4k+2$.
- Suppose now that there exists $i \in [m-1]$ such that X intersect $B_i \cup \widehat{B}_i$ but not A_{i+1}. Let $b^i_s \in B_i$ be the rightmost vertex in σ that belongs to X. The edges of $G[X, \overline{X}]$ belong to the following cuts: (1) the cut between A_i and $B_i \cap \overline{X}$, (2) the cut between $\widehat{B}_i \cap X$ and A_{i+1} and (3) the cut between b^i_s and $\{\widehat{b}^i_s\} \cap \overline{X}$. As argued for the previous case, the ranks of the first two cuts are upper bounded by $2k$ and the rank of the third is upper bounded by one. We conclude that $\mathsf{rw}(X, \overline{X}) \leqslant 4k+1$. ◀

▶ **Theorem 6.5.** *There is no algorithm solving* Weighted Dominating Set *in time $2^{o(\mathsf{lrw}(G)^2)} n^{O(1)}$ unless ETH fails.*

Proof. Assume that there exists a $2^{o(\mathsf{lrw}(G)^2)} n^{O(1)}$ time algorithm for Weighted Dominating Set. We prove that it implies the existence of a $2^{o(k^2)} n^{O(1)}$ time algorithm for 3-SAT where k^2 is the number of variables. This will contradict ETH.

Suppose that we are given a 3-SAT formula φ with k^2 variables and m clauses. We construct the graph G described above. As G has $(2^k k + k + 1)m + 2^{2k}(m-1)$ vertices, we deduce that we can construct G in time $2^{O(k)} m$.

From Lemmas 6.1 and 6.2, we know that G admits a dominating set of weight at most $(2^{2k}+2)km + 2^k(m-1)$ iff φ is satisfiable. Thanks to Lemma 6.4, we can compute in polynomial time a linear decomposition of G of rank-width at most $4k+2$. By assumption, we can compute a dominating set of minimum weight of G in time $2^{o(k^2)} n^{O(1)}$. Hence, we can decide whether φ is satisfiable in time $2^{o(k^2)} n^{O(1)}$. This contradicts ETH by Lemma 4.1. ◀

7 Concluding Remarks

We showed the first ETH-tight lower bounds for problems with time complexity $2^{O(\mathsf{rw}^2)} n^{O(1)}$ parameterized by rank-width rw. In particular, we showed that algorithms with such time complexity are optimal for Independent Set, Weighted Dominating Set, Maximum Induced Matching, and Feedback Vertex Set.

We hope the tools designed in this paper could be used to design tight lower bounds for more problems parameterized by rank-width. In particular, is the $2^{O(\mathsf{rw}^2)}n^{O(1)}$ time algorithm in [6] for (unweighted) Dominating Set optimal under ETH? What about the $2^{O(q\cdot\mathsf{rw}^2)}n^{O(1)}$ time algorithm in [6] for q-Coloring (even when $q = 3$)? Finally, one could also explore the optimality of XP algorithms parameterized by rank-width such as the $n^{2^{O(\mathsf{rw})}}$ time algorithm in [19] for Chromatic Number. For the clique-width parameterization this was solved in [16].

References

1. Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discret. Appl. Math.*, 23(1):11–24, 1989. doi:10.1016/0166-218X(89)90031-0.
2. Rémy Belmonte and Ignasi Sau. On the complexity of finding large odd induced subgraphs and odd colorings. *Algorithmica*, 83(8):2351–2373, 2021. doi:10.1007/s00453-021-00830-x.
3. 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.
4. 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.
5. Hajo Broersma, Petr A. Golovach, and Viresh Patel. Tight complexity bounds for FPT subgraph problems parameterized by the clique-width. *Theor. Comput. Sci.*, 485:69–84, 2013. doi:10.1016/j.tcs.2013.03.008.
6. Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discret. Appl. Math.*, 158(7):809–819, 2010. doi:10.1016/j.dam.2009.09.009.
7. Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theor. Comput. Sci.*, 412(39):5187–5204, 2011. doi:10.1016/j.tcs.2011.05.022.
8. 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.
9. Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.*, 46(2):218–270, 1993. doi:10.1016/0022-0000(93)90004-G.
10. 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.
11. 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.
12. 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.
13. Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
14. Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is NP-complete. *SIAM J. Discret. Math.*, 23(2):909–939, 2009. doi:10.1137/070687256.
15. 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.
16. Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Clique-width III: Hamiltonian cycle and the odd case of graph coloring. *ACM Trans. Algorithms*, 15(1):9:1–9:27, 2019. doi:10.1145/3280824.

17 Fedor V. Fomin and Tuukka Korhonen. Fast FPT-approximation of branchwidth. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 886–899. ACM, 2022. doi:10.1145/3519935.3519996.

18 Robert Ganian and Petr Hlinený. On parse trees and myhill-nerode-type tools for handling graphs of bounded rank-width. *Discret. Appl. Math.*, 158(7):851–867, 2010. doi:10.1016/j.dam.2009.10.018.

19 Robert Ganian, Petr Hlinený, and Jan Obdrzálek. A unified approach to polynomial algorithms on graphs of bounded (bi-)rank-width. *Eur. J. Comb.*, 34(3):680–701, 2013. doi:10.1016/j.ejc.2012.07.024.

20 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.

21 Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

22 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.

23 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.

24 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Finding branch-decompositions of matroids, hypergraphs, and more. *SIAM J. Discret. Math.*, 35(4):2544–2617, 2021. doi:10.1137/19M1285895.

25 Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

26 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.

27 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.

28 Sang-il Oum, Sigve Hortemo Sæther, and Martin Vatshelle. Faster algorithms for vertex partitioning problems parameterized by clique-width. *Theor. Comput. Sci.*, 535:16–24, 2014. doi:10.1016/j.tcs.2014.03.024.

29 Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

30 Martin Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, Norway, 2012.

On the Multilinear Complexity of Associative Algebras

Markus Bläser
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany

Hendrik Mayer[1]
Massachusetts Institute of Technology, Cambridge, MA, USA

Devansh Shringi[1]
University of Toronto, Canada

Abstract

Christandl and Zuiddam [12] study the multilinear complexity of d-fold matrix multiplication in the context of quantum communication complexity. Bshouty [8] investigates the multilinear complexity of d-fold multiplication in commutative algebras to understand the size of so-called testers. The study of bilinear complexity is a classical topic in algebraic complexity theory, starting with the work by Strassen. However, there has been no systematic study of the multilinear complexity of multilinear maps.

In the present work, we systematically investigate the multilinear complexity of d-fold multiplication in arbitrary associative algebras. We prove a multilinear generalization of the famous Alder-Strassen theorem, which is a lower bound for the bilinear complexity of the (2-fold) multiplication in an associative algebra. We show that the multilinear complexity of the d-fold multiplication has a lower bound of $d \cdot \dim A - (d-1)t$, where t is the number of maximal twosided ideals in A. This is optimal in the sense that there are algebras for which this lower bound is tight. Furthermore, we prove the following dichotomy that the quotient algebra $A/\operatorname{rad} A$ determines the complexity of the d-fold multiplication in A: When the semisimple algebra $A/\operatorname{rad} A$ is commutative, then the multilinear complexity of the d-fold multiplication in A is polynomial in d. On the other hand, when $A/\operatorname{rad} A$ is noncommutative, then the multilinear complexity of the d-fold multiplication in A is exponential in d.

2012 ACM Subject Classification Theory of computation → Algebraic complexity theory; Mathematics of computing

Keywords and phrases Multilinear computations, associative algebras, matrix multiplication, Alder–Strassen theorem

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.12

Acknowledgements Hendrik Mayer would like to thank the MIT MISTI-Germany program for funding his internship.

1 Introduction

A fundamental problem in algebraic complexity theory is the question about the costs of multiplication, for instance, of matrices, triangular matrices, or polynomials. To be more specific, let $\mathbb{F}$ be a field and let A be a finite dimensional associative $\mathbb{F}$-algebra with unity 1. By fixing a basis of A, say $v_1, \ldots, v_N$, we can define a set of bilinear forms corresponding to the multiplication in A. If $v_\mu v_\nu = \sum_{\kappa=1}^{N} \alpha_{\mu,\nu,\kappa} v_\kappa$ for $1 \leq \mu, \nu \leq N$ with *structural constants* $\alpha_{\mu,\nu,\kappa} \in \mathbb{F}$, then these constants and the identity

[1] Work done during an internship at Saarland University.

© Markus Bläser, Hendrik Mayer, and Devansh Shringi[1];
licensed under Creative Commons License CC-BY 4.0
40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 12; pp. 12:1–12:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\left(\sum_{\mu=1}^{N} X_\mu v_\mu\right)\left(\sum_{\nu=1}^{N} Y_\nu v_\nu\right) = \sum_{\kappa=1}^{N} b_\kappa(X,Y)v_\kappa$$

define the desired bilinear forms $b_1,\dots,b_N$. The *bilinear complexity* or *rank* of $b_1,\dots,b_N$ is the smallest number of *bilinear* multiplications necessary and sufficient to compute $b_1,\dots,b_N$ from the indeterminates $X_1,\dots,X_N$ and $Y_1,\dots,Y_N$. A bilinear multiplication is a multiplication of the form $u_\rho(X_1,\dots,X_N)v_\rho(Y_1,\dots,Y_N)$, where u_ρ and v_ρ are linear forms. Additions and multiplications with scalars from $\mathbb{F}$ are free of costs.

It is easy to see that the bilinear complexity of $b_1,\dots,b_N$ does not depend on the choice of $v_1,\dots,v_N$, thus we may speak about the bilinear complexity (or rank) of (the multiplication in) A. Equivalently, we can formulate the problem as a tensor rank problem. Given the structure tensor $(\alpha_{\mu,\nu,\kappa})$ of the algebra, which you can think of as a three-dimensional matrix, the bilinear complexity of an algebra is exactly the number of rank-one tensors that are needed to write the structure tensor of an algebra as a sum of rank-one tensors. For an introduction to algebraic complexity theory and for further background on tensor rank, we recommend [11, 18].

The best general lower bound for the bilinear complexity of an associative algebra A is due to Alder and Strassen [1], they show

$$R(A) \geq 2\dim A - t\,, \tag{1}$$

where t is the number of maximal twosided ideals in A. (See Section 2 for more background on algebras.) This bound has been improved for a large class of so-called semisimple algebras to $\frac{5}{2}\dim A - 3n$, where n is the sum of the sizes of the matrices in the decomposition of A into simple algebras [3]. The Alder–Strassen theorem itself even holds for a more general complexity measure, the so-called multiplicative complexity. Algebras for which the Alder–Strassen bound is tight are called algbras of minimal rank or minimal multiplicative complexity. They have been characterised in terms of their algebraic structure in [4] and [6].

The most prominent algebra is the algebra of $n \times n$-matrices. Strassen [23] proved that the rank of 2×2-matrix multiplication is upper bounded by 7, giving rise to his famous matrix multiplication algorithm. Winograd [25] proved that the bound of 7 is optimal, which can be considered as a first instance of the Alder–Strassen theorem. Over the past decades, an exciting development of fast matrix multiplication algorithms has taken place, culminating in the current fastest algorithms with running time $O(n^{2.373})$ [14, 22, 24, 16, 2], see [5] for an overview. The best lower bound is due to Landsberg [17] and is $R(\mathbb{F}^{n\times n}) \geq 3n^2 - o(n^2)$.

When an algebra A is associative, the d-fold multiplication is a multilinear map $A \times \cdots \times A \to A$. The notion of bilinear complexity naturally generalizes: Instead of bilinear products, we have d-linear products. Equivalently, we can study the tensor rank of tensors of higher orders. Christandl and Zuiddam [12] recently studied the multilinear complexity of d-fold matrix multiplication, which is a so-called graph tensor on the cycle graph of length d. The multilinear complexity of such graph tensors plays a particular role in quantum communication complexity, see e.g. [10]. Prior to this, Nisan and Wigderson studied depth-three circuits for iterated matrix multiplication using the partial derivative method [19].

Bshouty [9] invented the concept of testers. Testers are a useful tool in algebraic algorithms to reduce identity testing from large domains to smaller ones. In the case of the class of multilinear forms of degree d over an algebra A, he proves that the size of the optimal tester is exactly equal to the multilinear complexity of the d-fold multiplication in the algebra A [8]. Thus, it is interesting to understand the multilinear complexity of d-fold multiplication of arbitrary algebras.

Related models have been studied in algebraic complexity theory, like (set-multilinear) depth-3-circuits, higher order tensor rank, or Waring rank, see e.g. the surveys [21, 7]. But there has been no systematic study of the multilinear complexity of d-fold multiplication maps.

1.1 Our work

We initiate the systematic study of the multilinear complexity of d-fold multiplication in an associative algebra. Our motivation comes from the work mentioned above that relies on bounds for the multilinear complexity in certain algebras.

We prove a multilinear generalization of the Alder–Strassen theorem (Theoreom 19), namely that

$$R(A, d) \geq d \cdot \dim A - (d-1)t,$$

where t is the number of maximal twosided ideals in A. Here $R(A, d)$ denotes the rank of d-fold multiplication in A (see Section 3). This bound is tight in the sense that there are algebras for which equality holds, for instance, products of simply generated division algebras (see Section 5.3). For $d = 2$, we exactly recover the Alder–Strassen theorem.

Moreover, an interesting phenomenon arises. When we keep the algebra A fixed and consider the multilinear complexity $R(A, d)$ as a function of d, then the growth is either polynomial (where the exponent might depend on A) or exponential. When $A/\operatorname{rad} A$ is commutative, then $R(A, d)$ is polynomial, more precisely $R(A, d) \leq d^{(s-1)(D+1)} \cdot D^{(D+3)s}$, where D is the dimension of A and s is the so-called index of nilpotency, which can be upper bounded by D (Theorem 18). This result holds over large enough perfect fields. Note that most fields are perfect, like fields of characteristic zero, finite fields, or algebraically closed fields. On the other hand, when $A/\operatorname{rad} A$ is noncommutative, then we prove an exponential lower bound (Theorem 13 together with Lemma 5). So we obtain a dichotomy.

We would like to stress once again that the motivation for this work are the above mentioned applications in quantum communication and the study of testers. Multilinear computations are in general not suited to actually evaluate d-fold multiplication maps in practice. While for instance Christandl and Zuiddam [12] prove that the multilinear complexity of d-fold matrix multiplication is exponential in d, we can of course multiply d matrices in polynomial time. The reason is that in practice, we first multiply two matrices, then multiply the result with the third one and so on. In multilinear computations, we cannot reuse results. It is essentially the same as the difference between formula and circuit size, which can be exponential, too. Nevertheless, bilinear algorithms have been successfully used to construct algorithms for the multiplication of two matrices, because in the case $d = 2$, bilinear computations and circuit size only differ by a factor of 2, see e.g. [5].

1.2 Organisation of the paper

Section 2 provides some basic facts about associative algebras. In Section 3, we introduce the model of computation and prove some basic facts about it. Then we prove how to remove the radical in Section 4, similar to the original proof of the Alder–Strassen theorem. In Sections 5 and 6, we study semisimple algebras, first semisimple algebras that are in addition basic and then arbitrary semisimple algebras. Then we go on with a study of the algebra of upper triangular matrices in Section 7, which is an example of a noncommutative algebra such that $A/\operatorname{rad} A$ is commutative. In constrast to general matrices, for which we have an exponential lower bound, we prove a polynomial upper bound for the multilinear complexity (as a function

of d). In Section 8, we generalize this result to arbitrary, potentially noncommutative algebras A for which $A/\operatorname{rad} A$ is commutative. Finally, we prove the generalized Alder–Strassen theorem in Section 9.

2 Structure of associative algebras

We collect some elementary properties of associative algebras. The term *algebra* here always means a finite dimensional associative algebra with identity 1 over some field $\mathbb{F}$. The term *left module* and *right module* always means a finitely generated left module and right module over some algebra A, respectively. By the embedding $\alpha \mapsto \alpha \cdot 1$, $\mathbb{F}$ becomes a subalgebra of A. Hence, every A-left module or A-right module is also a finite dimensional $\mathbb{F}$-vector space. If we speak of a basis of an algebra or a module, we always mean a basis of the underlying vector space. Further material as well as proofs of the mentioned properties can be found in [20, 13, 15].

A left ideal I (and in the same way, a right ideal or twosided ideal) is called *nilpotent*, if $I^n = \{0\}$ for some positive integer n.

▶ **Fact 1.** *For all finite dimensional algebras A, the following holds:*

1. *The sum of all nilpotent left ideals of A is a nilpotent twosided ideal, which contains every nilpotent right ideal of A. This twosided ideal is called the* radical *of A and is denoted by* $\operatorname{rad} A$.
2. *The quotient algebra $A/\operatorname{rad} A$ contains no nilpotent ideals other than the zero ideal.*
3. *The radical of A is contained in every maximal twosided ideal of A.*
4. *The algebras A and $A/\operatorname{rad} A$ have the same number of maximal twosided ideals.*

We call an algebra A *semisimple*, if $\operatorname{rad} A = \{0\}$. By the above fact, $A/\operatorname{rad} A$ is semisimple. An algebra A is called *simple*, if there are no twosided ideals in A, except the zero ideal and A itself.

We now describe some of the most important ways to construct new algebras from given ones: If A and B are $\mathbb{F}$-algebras, then the direct product $A \times B$ with componentwise addition and multiplication is again an $\mathbb{F}$-algebra. The set of all $n \times n$–matrices with entries from A forms an $\mathbb{F}$-algebra (with the usual definition of addition and multiplication of matrices). This algebra is denoted by $M_n(A)$ or $A^{n\times n}$.

We denote the set of all units of an algebra A, that is, the set of all invertible elements, by $A^\times$. An algebra D is called a *division algebra*, if $D^\times = D \setminus \{0\}$. An algebra A is called *local*, if $A/\operatorname{rad} A$ is a division algebra, and A is called *basic*, if $A/\operatorname{rad} A$ is a direct product of division algebras.

If $x \in A$, we denote by AxA the ideal generated by x. If A is commutative, we will also write (x) for short. Furthermore, $\mathbb{F}[x]$ denotes the smallest subalgebra of A that contains x. If $x_1, \ldots, x_m \in A$ mutually commute, then $\mathbb{F}[x_1, \ldots, x_m]$ denotes the smallest subalgebra of A that contains $x_1, \ldots, x_m$. For elements $v_1, \ldots, v_n$ of some vector space, $\langle v_1, \ldots, v_n \rangle$ denotes their linear span.

The following fundamental theorem describes the structure of semisimple algebras.

▶ **Theorem 2** (Wedderburn). *Every finite dimensional semisimple algebra is isomorphic to a finite direct product of simple algebras. Every finite dimensional simple $\mathbb{F}$-algebra A is isomorphic to an algebra $M_n(D)$ for an integer $n \geq 1$ and an $\mathbb{F}$-division algebra D. The integer n and the algebra D are uniquely determined by A (the latter one up to isomorphism).*

3 Multilinear computations

For a vector space V, V^* denotes the dual space of V, that is, the vector space of all linear forms on V.

We define multilinear complexity in a coordinate-free way. The d-fold multiplication in an algebra is a multilinear map $\phi : A^d \to A$. A multilinear computation consists of linear forms $F_{i,\delta} \in A^*$, $1 \leq \delta \leq d$, and elements $w_i \in A$, $1 \leq i \leq r$, such that

$$x_1 \cdot x_2 \cdot \ldots \cdot x_d = \sum_{i=1}^{r} F_{i,1}(x_1)F_{i,2}(x_2)\ldots F_{i,d}(x_d) \cdot w_i$$

for all $x_1, \ldots, x_d \in A$. r is called the length of the computation. The length of a shortest computation for the d-fold multiplication is the d-linear complexity of A. We denote this quantity by $R(A, d)$.

In the remainder of this section, we collect some useful properties of multilinear complexity. If we choose vector spaces $V_i \subseteq A$, $1 \leq \delta \leq d$, we get a multilinear map $\phi' : V_1 \times \ldots, \times V_d \to A$ in the canonical way. By restricting each $F_{i,\delta}$ to V_δ, the computation above turns into a computation for ϕ' of the same length. Obviously, $R(\phi') \leq R(\phi)$.

If I is a twosided ideal of A, then A/I is an algebra again. Each $F_{i,\delta}$ induces a linear form on A/I in the canonical way. By replacing each $F_{i,\delta}$ with this linear form and mapping each w_i to its image in A/I, the computation turns into a computation for the multiplication in A/I.

We can define an equivalence relation on the set of all d-linear computations for an algebra. Let $a_0, \ldots, a_d \in A$ be invertible. We have the identity

$$a_0^{-1}a_0x_1a_1^{-1}a_1x_2a_2^{-1}a_2 \ldots a_{d-1}x_da_d{}^{-1}a_d = x_1 \ldots x_d.$$

Therefore, the computation given by $\hat{F}_{i,\delta}(x) = F_{i,\delta}(a_{i-1}xa_i^{-1})$, $1 \leq \delta \leq d$, and $\hat{w}_i = a_0^{-1}w_ia_d$ is again a computation for ϕ, the multiplication in A. The action of $(A^\times)^{d+1}$ defines an equivalence relation on the set of all computations of length r.

The following claim will turn out useful in our lower bound proofs:

▷ Claim 3. Consider a computation for an algebra A of dimension N as above. For every j, $F_{1,j}, F_{2,j}, \ldots, F_{r,j}$ span A^*. That is, we can have $F_{1,j}, \ldots, F_{N,j}$ as a basis of A^* after reordering.

Proof. Assume they do not span A^* for some j, then there is an element $y \in A \setminus \{0\}$ such that $F_{1,j}(y) = F_{2,j}(y) = \ldots = F_{r,j}(y) = 0$. This means that $x_1 \cdot x_2 \cdot \ldots \cdot x_{j-1} \cdot y \cdot x_{j+1} \cdot \ldots \cdot x_d = 0$ for all $x_1, \ldots, x_d$. By setting $x_i = 1$ for $i \neq j$, we get that $y = 0$, which is a contradiction. We can reorder the $F_{i,j}$'s such that $F_{1,j}, \ldots, F_{N,j}$ span A^*. ◁

The first item of the following fact follows from the trivial decomposition. The second by setting $x_{d+1} = 1$.

▶ **Fact 4.**

1. $R(A, d) \leq (\dim A)^d$
2. $R(A, d) \leq R(A, d + 1)$

The d-fold multiplication in an algebra corresponds to a tensor in $t_{A,d} \in A^* \otimes \cdots \otimes A^* \otimes A$. The rank of $t_{A,d}$ is the minimum number r of rank-one tensors $u_{\rho,1} \otimes \cdots \otimes u_{\rho,d} \otimes v_\rho \in A^* \otimes \cdots \otimes A^* \otimes A$ such that

$$t_{A,d} = \sum_{\rho=1}^{r} u_{\rho,1} \otimes \cdots \otimes u_{\rho,d} \otimes v_\rho.$$

Tensor rank and multilinear complexity coincide, that is, $R(t_{A,d}) = R(A, d)$.

4 Removing the radical $\mathrm{rad}(A)$

We start by generalizing the first lemma of the proof by Alder-Strassen to multilinear complexity, allowing us to work with the semisimple algebra $A/\operatorname{rad}(A)$, which has a nicer structure than a general algebra.

▶ **Lemma 5.**

$$R(A,d) \geq R(A/\operatorname{rad}(A),d) + d \cdot \dim(\operatorname{rad}(A)).$$

Proof. Consider an algebra A over the field $\mathbb{F}$. Let ϕ be a length r computation with $F_{i,j} \in A^*$ and $w_i \in A$ such that the d-fold multiplication of A is computed by the following equation:

$$x_1 \cdot x_2 \cdot \ldots \cdot x_d = \sum_{i=1}^{r} F_{i,1}(x_1)F_{i,2}(x_2)\ldots F_{i,d}(x_d) \cdot w_i.$$

We will inductively construct vector spaces $V_1, \ldots, V_d$ such that $V_\delta \oplus \operatorname{rad}(A) = A$ and $1 \in V_\delta$. Let ϕ_j be the multiplication map restricted to $V_1 \times \cdots \times V_j \times A \times \cdots \times A$. We will now prove that $R(A,d) \geq R(\phi_j) + j \cdot \dim(\operatorname{rad}(A))$. The base case $j = 0$ is clear.
Induction Hypothesis: $R(A,d) \geq R(\phi_{j-1}) + (j-1) \cdot \dim(\operatorname{rad}(A))$.
Induction Step: We obtain a basis of A^* using the following claim similar to Claim 3, which we prove later.

▷ Claim 6. For all j, we have that in a computation for ϕ_{j-1}, $F_{1,j}, F_{2,j}, \ldots, F_{r,j}$ span A^*. That is, we can have $F_{1,j}, \ldots, F_{N,j}$ as a basis of A^* after reordering. Here, $N = \dim A$.

Now for the basis $F_{1,j}, \ldots, F_{N,j}$ of A^*, we calculate the dual basis $a_{1,j}, \ldots, a_{N,j}$, which is a basis of A as $A^{**} = A$ (A is finite dimensional). Note that from the definition of dual basis, $F_{i,j}(a_{k,j}) = \delta_{ik}$, where δ_{ik} is Kronecker's delta.

Consider the canonical projection $P : A \to A/\operatorname{rad}(A)$. Let $\dim(\operatorname{rad}(A)) = \rho$. Then $\dim(A/\operatorname{rad}(A)) = N - \rho$. After rearrangement, we can have that $P(a_{1,j}), \ldots, P(a_{N-\rho,j})$ form a basis of $A/\operatorname{rad}(A)$. So we can assume w.l.o.g. that $P(a_{i,j})$ for $i \in [N-\rho]$ span $A/\operatorname{rad}(A)$. In particular, with $V_j := \langle a_{1,j}, \ldots, a_{N-\rho,j}\rangle$, we have that $V_j \cap \operatorname{rad} A = \{0\}$.

We also observe that $1 \notin \operatorname{rad}(A)$. Now, we want that $a_{1,j}, \ldots, a_{N-\rho,j}$ span 1. If they don't, then there must exist $z \in \operatorname{rad}(A)$ such that $1 - z$ is spanned by $a_{1,j}, \ldots, a_{N-\rho,j}$, i.e., there are $\beta_1, \ldots, \beta_{N-\rho} \in \mathbb{F}$ such that

$$1 - z = \sum_{i=1}^{N-\rho} \beta_i a_{i,j}.$$

As $z \in \operatorname{rad}(A)$, there exists an s such that $z^s = 0$. Therefore, $1 - z$ is invertible and has inverse $1 + z + z^2 + \ldots + z^{s-1}$. We can write

$$x_1 \ldots x_{j-1}x_jx_{j+1}\ldots x_d = x_1 \ldots x_{j-1}(x_j(1-z))((1-z)^{-1}x_{j+1})\ldots x_d,$$

which changes the computation as follows:

$$\forall i \in [r]: \quad \hat{F}_{i,j}(x_j) = F_{i,j}(x_j(1-z)), \text{ and } \hat{F}_{i,j+1}(x_{j+1}) = F_{i,j+1}((1-z)^{-1}x_{j+1}).$$

All other $F_{i,h}$ are not changed. If $j = d$, then w_i is changed instead of $F_{i,d+1}$ (which does not exist). Compare also Section 3.

The dual basis will change as $\hat{a}_{i,j} = a_{i,j}(1-z)^{-1}$. As we have $1-z$ in the span of $a_{1,j}, \ldots, a_{N-\rho,j}$, we will have that 1 is in the span of $\hat{a}_{1,j}, \ldots, \hat{a}_{N-\rho,j}$. Note that this does not change the spaces $V_1, \ldots, V_{j-1}$. So, we have $1 = \sum_{i=1}^{N-\rho} \beta_i \hat{a}_{i,j}$.

Set $V_j = \langle \hat{a}_{1,j}, \ldots, \hat{a}_{N-\rho,j} \rangle$. By the choice of V_j, $F_{i,j}|_{V_j} = 0$ for $i \in \{N-\rho+1, \ldots N\}$, as $\hat{F}_{i,j}(\hat{a}_{k,j}) = F_{i,j}(a_{k,j}) = 0$ if $i \neq k$. Thus when restricting to V_j, we can remove ρ products from the computation. By the induction hypothesis, we get that

$$R(A,d) \geq R(\phi_{j-1}) + (j-1)\dim(\operatorname{rad} A)) \geq R(\phi_j) + \dim(\operatorname{rad} A) + (j-1)\dim(\operatorname{rad} A)).$$

This finishes the proof of the induction step.

We now have a multilinear map $\phi_d : V_1 \times \cdots \times V_d \to A$ with $R(A,d) \geq R(\phi_d) + d \cdot \dim(\operatorname{rad} A))$. To finish the proof of the lemma, it suffices to prove $R(\phi_d) \geq R(A/\operatorname{rad} A, d)$. Since $V_\delta \cap \operatorname{rad} A = \{0\}$ for $\delta \in [d]$, the restriction of the projection P to V_δ is an isomorphism. The following diagram commutes:

$$\begin{array}{ccccccc} V_1 & \times & \ldots & \times & V_d & \longrightarrow & A \\ \downarrow & & \ldots & & \downarrow & & \downarrow \\ A/\operatorname{rad} A & \times & \ldots & \times & A/\operatorname{rad} A & \longrightarrow & A/\operatorname{rad} A \end{array}$$

Thus a computation for ϕ_d can be turned into a computation for the d-fold multiplication in $A/\operatorname{rad} A$ (compare Section 3). ◀

We also give a proof of Claim 6, which we made in the proof above.

Proof of Claim 6. Assume they do not span A^* for some j, then there is an element $y \in A \setminus \{0\}$ such that $F_{1,j}(y) = F_{2,j}(y) = \ldots = F_{r,j}(y) = 0$. This means that $x_1 \cdot x_2 \cdot \ldots \cdot x_{j-1} \cdot y \cdot x_{j+1} \cdot \ldots \cdot x_d = 0$ for all $x_1, \ldots, x_d$. Since $1 \in V_\delta$ for every δ, this means $y = 0$, which is a contradiction. We can reorder the $F_{i,j}$'s such that $F_{1,j}, \ldots, F_{N,j}$ span A^*. ◁

4.1 A tight example

An example we can consider is the algebra $A = \mathbb{F}[x]/(x^n)$ of univariate polynomials modulo x^n. We clearly have $\dim(A) = n$. We see that all polynomials in the algebra with zero constant term are nilpotent. Therefore, $\operatorname{rad}(A) = \langle x, x^2, \ldots, x^{n-1} \rangle$ and $\dim(\operatorname{rad}(A)) = n-1$.

From Lemma 5, we get that $R(A,d) \geq R(A/\operatorname{rad}(A), d) + d \cdot \dim(\operatorname{rad}(A))$, i.e. for this case $R(A,d) \geq d \cdot (n-1) + 1$ as $A/\operatorname{rad}(A)$ is just $\mathbb{F}$.

For the upper bound, we see that the d-fold product (without reduction modulo x^n) $f = f_1 \cdot f_2 \cdot \ldots \cdot f_d$ will actually be a polynomial of degree $d(n-1)$. We can evaluate each f_i at $d(n-1)+1$ points (which is free of costs in our model) and multiply the evaluations with $d(n-1)+1$ costing multiplication operations to get an evaluation of f at $d(n-1)+1$ points. (This assumes that our field $\mathbb{F}$ is large enough.) Using $d(n-1)+1$ evaluations of f, we can obtain f using interpolation, which is again free of costs. We ignore the terms with degree higher than n, thus obtaining the final result f. This gives an upper bound of $d \cdot (n-1) + 1$, proving that our lower bound is tight in this case.

5 Products of division algebras

Now that we are to work on $R(A/\operatorname{rad}(A), d)$, we see that $A/\operatorname{rad}(A)$ is a semisimple algebra. Using Theorem 2, we have

$$A/\operatorname{rad}(A) = A_1 \times A_2 \times \ldots \times A_k$$

where $A_i = M_{n_i}(D_i)$, i.e., $n_i \times n_i$ matrices with entries from a division algebra D_i.

In this section, we will mainly focus on case when $n_i = 1$, i.e., all A_i are division algebras.

5.1 Single division algebra ($k = 1$)

For simplicity, we will first take a look at the following lemma, for the case when $k = 1$, i.e., $A = D$ where D is any division algebra. Let $N = \dim(A)$.

▶ **Lemma 7.** *Let A be a division algebra. Then*

$$R(A, d) \geq d \cdot \dim(A) - (d - 1).$$

Proof. Let $r = R(A, d)$. We consider an optimal length r computation for the d-fold multiplication of A. By Claim 3, $F_{1,1}, \ldots, F_{N,1}$ is a basis of A^*. Let $a_{1,1}, \ldots, a_{N,1}$ be the dual basis, that is, $F_{i,1}(a_{j,1}) = \delta_{i,j}$. We know that the $a_{1,1}, \ldots, a_{N,1}$ are all invertible. Let us define a new basis $\hat{a}_{i,1} = a_{i,1}{a_{1,1}}^{-1}$, $i \in [N]$. Let

$$\forall i \in [r] : \hat{F}_{i,1}(x_1) = F_{i,1}(x_1 a_{1,1}), \;\; \hat{F}_{i,2}(x_2) = F_{i,2}({a_{1,1}}^{-1} x_2), \text{ and } \hat{F}_{i,j}(x_j) = F_{i,j}(x_j),\ j \geq 3.$$

We note that $\hat{a}_{1,1} = 1$ and $\hat{F}_{i,1}(1) = F_{i,1}(a_{1,1}) = \delta_{i,1}$. We have that:

$$1 \cdot x_2 \cdot \ldots \cdot x_d = \hat{F}_{1,1}(\hat{a}_{1,1})\hat{F}_{1,2}(x_2) \ldots \hat{F}_{1,d}(x_d) \cdot w_1 + \sum_{i=N+1}^{r} \hat{F}_{i,1}(\hat{a}_{1,1})\hat{F}_{i,2}(x_2) \ldots \hat{F}_{i,d}(x_d) \cdot w_i.$$

As there are $r - N + 1$ terms to sum, $R(A, d-1) \leq r - N + 1 = R(A, d) - N + 1$. Since $R(A, 1) = N$, we have by induction that

$$R(A, d) \geq N + (N - 1)(d - 1) = d \cdot \dim(A) - (d - 1). \qquad \blacktriangleleft$$

5.2 Arbitrary products of division algebras

Now $A = D_1 \times \cdots \times D_k$ is a product of division algebras.

▶ **Theorem 8.**

$$R(A, d) \geq d \cdot \dim(A) - (d - 1)k.$$

Proof. As in the single division algebra case, we consider an optimal length r computation for the d-fold multiplication in A:

$$x_1 \cdot x_2 \cdot \ldots \cdot x_d = \sum_{i=1}^{r} F_{i,1}(x_1)F_{i,2}(x_2) \ldots F_{i,d}(x_d) \cdot w_i.$$

Similarly to the single division algebra case, $F_{1,1}, F_{2,1}, \ldots, F_{N,1}$ is w.l.o.g. a basis of A^*. Let $a_1, a_2, \ldots, a_N$ be the dual basis, that is, $F_{i,1}(a_j) = \delta_{i,j}$. Each a_i can be written as $a_i = (a_{i,1}, a_{i,2}, \ldots, a_{i,k})$, where each $a_{i,\ell} \in D_\ell$, $1 \leq \ell \leq k$.

We can assume w.l.o.g. that the span of $a_1, \ldots, a_k$ contains an element $b = (b_1, \ldots, b_k)$ that is nonzero in every component of $D_1 \times \cdots \times D_k$. Since each b_ℓ is nonzero, it is invertible, because D_ℓ is a division algebra. Thus b is invertible, too, and $b^{-1} = (b_1^{-1}, \ldots, b_k^{-1})$.

Now, we define $\hat{a}_j = a_j b^{-1}$. Also, let $\hat{F}_{i,1}(x_1) = F_{i,1}(x_1 b)$ and $\hat{F}_{i,2} = F_{i,2}(b^{-1}x_2)$. By construction, $1 = (1, \ldots, 1)$ is contained in the linear span of $\hat{a}_1, \ldots, \hat{a}_k$. Thus

$$1 \cdot x_2 \cdot \ldots \cdot x_d = \sum_{\ell=1}^{k} \hat{F}_{\ell,1}(1)\hat{F}_{\ell,2}(x_2) \ldots \hat{F}_{\ell,d}(x_d) \cdot w_\ell + \sum_{i=N+1}^{r} \hat{F}_{i,1}(1)\hat{F}_{i,2}(x_2) \ldots \hat{F}_{i,d}(x_d) \cdot w_\ell.$$

Similarly to the $k = 1$ case, we have that $R(A, d-1) \leq r - N + k = R(A, d) - N + k$. Using induction, we get $R(A, d) \geq N + (d-1)N + (d-1)k$, because $R(A, 1) = N$. $\blacktriangleleft$

5.3 Tight example

An example we can consider is the divison algebra $A = \mathbb{F}[x]/(x^n + 1)$, with $x^n + 1$ being irreducible (think of $\mathbb{F} = \mathbb{Q}$ for instance). We clearly have $\dim(A) = n$. This example is similar to the one in Section 4.1.

From Lemma 7, we get that $R(A, d) \geq d \cdot \dim(A) - (d - 1)$. For the matching upper bound, we see that $f = f_1 \cdot f_2 \cdot \ldots \cdot f_d$ will actually be a polynomial of degree $d(n - 1)$. We can evaluate each f_i at $d(n - 1) + 1$ different points and multiply the evaluations to get an evaluation of f at $d(n - 1) + 1$ points. Using these $d(n - 1) + 1$ evaluations of f, we can obtain f using interpolation. Again, $\mathbb{F}$ needs to be large enough. Finally, we calculate $f \bmod (x^n + 1)$, giving the final result in A. Thus, this yields an upper bound of $d \cdot (n - 1) + 1 = d \cdot \dim A - (d - 1)$.

If we take the k-fold product A^k, we also see that the bound of Theorem 8 is tight, too, for arbitrary k.

In general, we can get a polynomial upper bound (in d) on the multilinear complexity of the d-fold multiplication in any commutative algebra, provided that the field $\mathbb{F}$ is large enough. The exponent of the polynomial depends on the dimension of the algebra.

▶ **Fact 9.** *Let A be a finite dimensional commutative algebra. Then there is a polynomial ring R and an ideal $I \subset R$ such that $A = R/I$.*

▶ **Lemma 10.** *Let A be a commutative algebra over a field $\mathbb{F}$ with k generators and highest degree δ of a variable in a basis of the vector space R/I as above. Then $R(A, d) \leq (d\delta + 1)^k$ provided that $|\mathbb{F}| \geq d\delta + 1$.*

Proof. The idea is to use the same construction as in the example above. Each element x_i in the d-fold product can be presented as a polynomial in k variables and degree at most δ in each variable (modulo I). When we multiply these polynomials, we get a polynomial of individual degree $\leq d\delta$ in k variables. By multivariate Lagrange interpolation, it is enough to interpolate on a k-dimensional grid with $(d\delta + 1)$ points in each direction and a total number of $(d\delta + 1)^k$ many points. In this way we get the stated upper bound because reduction modulo I is free of costs in our model. ◀

We can bound k and $\delta - 1$ in the lemma above by $\dim A$. Thus the bound simplifies to $(d \cdot \dim A)^{\dim A}$.

When A is simply generated, then the construction of the lemma becomes the construction of the example above and we get the following tight bound.

▶ **Corollary 11.** *If A is a simply generated commutative algebra, then $R(A, d) = d \cdot \dim A - (d - 1)$.*

Let D be a commutative division algebra, that is, an extension field. If D is separable, then by the primitive element theorem, it is simply generated and the corollary above applies.

6 General semisimple algebras

In this section, we will look at the case of semisimple algebras, that is $A = A_1 \times \cdots \times A_k$ and each A_ℓ is a matrix algebra of format $n_\ell \times n_\ell$ with entries from a divison algebra D_ℓ. We can assume that at least one $n_\ell > 1$, otherwise we have a product of divison algebras. W.l.o.g. $n_1 > 1$.

6.1 Simple algebras ($k = 1$)

Christandl and Zuiddam [12] prove a lower bound for the case of a single matrix algebra with entries from the ground field $\mathbb{F}$ using flattening. When d is odd, then their bound n^{d+1} is optimal. It is rather easy to generalize the flattening approach to semisimple algebras. We start with simple algebras that are not division algebras.

▶ **Theorem 12.** *Let $n > 1$ and let D be a division algebra of dimension ℓ, and let d be odd. Then we have an exponential lower bound of*

$$R(M_n(D), d) \geq \ell \cdot n^{d+1}.$$

Proof. Let $f_1 = 1, f_2, \dots f_\ell$ be a basis for the division algebra D. Let $f_{i,j,\lambda}$ be the matrix in $M_n(D)$ that has the element f_λ in position (i, j) and zeros elsewhere. By the choice of f_1, $f_{i,j,1} = e_{i,j}$, where $e_{i,j}$ is the matrix that has a 1 in position (i, j) and zeros elsewhere. The set of all such $f_{i,j,\lambda}$ forms a basis of $M_n(D)$. The multiplication tensor in this case looks like

$$\sum_{i_1,\dots,i_{d+1}\in[n],\lambda_1,\dots,\lambda_d\in[\ell]} f^*_{i_1,i_2,\lambda_1} \otimes \dots \otimes f^*_{i_d,i_{d+1},\lambda_d} \otimes (f_{\lambda_1} \cdot f_{\lambda_2} \cdot \dots \cdot f_{\lambda_d}) f_{i_{d+1},i_1,1}.$$

The product $f_{\lambda_1} \cdot f_{\lambda_2} \cdot \dots \cdot f_{\lambda_d}$ is an element of D, so it can be written as $\sum_{h=1}^{\ell} \alpha^h_{\lambda_1,\dots,\lambda_d} f_h$ for suitable scalars $\alpha^h_{\lambda_1,\dots,\lambda_d}$. Thus, the tensor can be rewritten as

$$\sum_{i_1,\dots,i_{d+1}\in[n],\lambda_1,\dots,\lambda_d\in[\ell]} f^*_{i_1,i_2,\lambda_1} \otimes \dots \otimes f^*_{i_d,i_{d+1},\lambda_d} \otimes \sum_{h=1}^{\ell} \alpha^h_{\lambda_1,\dots,\lambda_d} f_{i_{d+1},i_1,h}. \tag{2}$$

Now we flatten the multilinear map into a matrix. The tensor defines a multilinear map

$$D_1^{n\times n} \times D_2^{n\times n} \times \dots \times D_d^{n\times n} \to D_{d+1}^{n\times n},$$

where we use the subscript i in D_i just to indicate the position. We will make it into a $((n^2\ell)^{(d+1)/2}) \times ((n^2\ell)^{(d+1)/2})$ matrix by combining the odd and even positions into the same parts, respectively. Therefore, the odd positions will index the rows and even will index the columns of the resulting matrix:

$$D_1^{n\times n} \otimes D_3^{n\times n} \otimes \dots \otimes D_d^{n\times n} \to (D_2^{n\times n})^* \otimes (D_4^{n\times n})^* \otimes \dots \otimes D_{d+1}^{n\times n}.$$

We further restrict this mapping to

$$D_1^{n\times n} \otimes \mathbb{F}^{n\times n} \otimes \dots \otimes \mathbb{F}^{n\times n} \to (\mathbb{F}^{n\times n})^* \otimes (\mathbb{F}^{n\times n})^* \otimes \dots \otimes D_{d+1}^{n\times n}.$$

The matrix is then of size $\ell n^{d+1} \times \ell n^{d+1}$. For every vector $f_{i_1,i_2,\lambda} \otimes f_{i_3,i_4,1} \otimes \dots \otimes f_{i_d,i_{d+1},1} \in D_1^{n\times n} \otimes \mathbb{F}^{n\times n} \otimes \dots \otimes \mathbb{F}^{n\times n}$, there is exactly one element in $(\mathbb{F}^{n\times n})^* \otimes (\mathbb{F}^{n\times n})^* \otimes \dots \otimes D_{d+1}^{n\times n}$ such that the tensor product appears in Equation (2), namely, $f^*_{i_2,i_3,1} \otimes f^*_{i_4,i_5,1} \otimes \dots \otimes f_{i_{d+1},i_1,\lambda}$. (Note that in this case, only one of the coefficients $\alpha^h_{\lambda_1,\dots,\lambda_d}$ is nonzero, namely $\alpha^\lambda_{\lambda,1\dots,1}$.) This means that the matrix is the identity matrix after appropriate permutation of rows and columns, which has rank $\ell \cdot n^{d+1}$.

The rank of the matrix is a lower bound on the rank of the tensor, which in turn is a lower bound on the multilinear complexity of $M_n(D)$. Thus, we have $R(M_n(D)) \geq \dim(D) \cdot n^{d+1}$. ◀

We also note that computing the rank of the flattened matrix for small ℓ, n and d using a computer algebra system gave us values suggesting the matrix rank to be $\ell \cdot n^{d+1}$. Therefore, a better lower bound using the flattening approach is unlikely.

6.2 Semisimple algebras

We can generalize the above proof for the case when the algebra $A = (D_1)^{n_1 \times n_1} \times (D_2)^{n_2 \times n_2} \times \ldots \times (D_k)^{n_k \times n_k}$.

▶ **Theorem 13.** *Let $A \cong \prod_{i=1}^{k} M_{n_i}(D_i)$ for arbitrary $n_i > 1$, arbitrary k, and D_i being a division algebra of dimension ℓ_i, and let d be odd. Then we have an exponential lower bound of*

$$R(A, d) \geq \sum_{i=1}^{k} \ell_i \cdot {n_i}^{d+1}.$$

Proof. The multiplication in A is a direct sum of multiplications in the k factors. The flattening matrix has a block structure when taking bases with respect to the k factors. Therefore, the flattening ranks add up. ◀

If d is even, we get a lower bound by using the fact that $R(A, d-1) \leq R(A, d)$. In the theorem above, all $n_i > 1$. The theorem also works in a similar way when $n_i = 1$ but D_i is noncommutative. In this case, we simply replace $\mathbb{F}$ by its algebraic closure (or just the splitting field of the division algebra). This will transform D_i into a matrix algebra with matrix size > 1.

6.3 Example

Consider the multiplication of d $n \times n$ matrices with elements from $D = \mathbb{F}[x]/(x^\ell + 1)$ with $x^\ell + 1$ being irreducible, i.e., we want to upper bound $R(M_n(D), d)$.

The degree of polynomials in the product matrix will be $d \cdot (\ell - 1)$, which means we will need $d \cdot (\ell - 1) + 1$ points to interpolate them. Thus, the algorithm is simply to evaluate the matrices at $d \cdot (\ell - 1) + 1$ points, calculate the result matrices of the product (complexity $R(M_n(\mathbb{F}), d)$) and interpolate at the points to get the final result matrix.

The number of multiplications will be $(d \cdot (\ell - 1) + 1) \cdot R(M_n(\mathbb{F}), d)$. We can do multiplication of d matrices in complexity n^{d+1} using the straight forward algorithm. Thus, it gives an upper bound of $R(M_n(D), d) \leq (d \cdot (\ell - 1) + 1) \cdot n^{d+1}$. We note that there is still a gap between our lower bound and this upper bound by a factor of d, but it is doubtful that the lower bound can be improved using flattening methods.

In particular, Theorem 8 can be viewed as the case of Theorem 13 when each $n_i = 1$ (the proof also works in this case). The flattening bound gives the lower bound ℓ while Theorem 8 gives $d\ell - (d - 1)$.

7 Upper triangular Matrices

In this section, we will look at the multilinear complexity of upper triangular matrices which are one of the special cases when A is noncommutative, but $A/\operatorname{rad}(A)$ is commutative. This is interesting as we saw in the case when $A/\operatorname{rad}(A)$ is noncommutative, we got exponential lower bounds in d, but for commutative algebras we saw linear lower bounds, and examples with linear upper bounds in d and polynomial upper bounds in general.

The result from Lemma 5 gives us a linear lower bound in d for general A. We will try to get more relevant lower and upper bounds for the special example of upper triangular matrices.

In the following, $U_n(\mathbb{F})$ denotes the algebra of upper triangular matrices. Its radical $\operatorname{rad}(U_n(\mathbb{F}))$ is the set of all upper triangular matrices with all diagonal elements 0. The quotient $U_n(\mathbb{F})/\operatorname{rad}(U_n(\mathbb{F}))$ will therefore be the set of all diagonal matrices, which is a commutative algebra and isomorphic to $\mathbb{F}^n$.

7.1 Lower Bound for $d \leq n$

We will use the flattening approach we used earlier to get a lower bound on the tensor rank for upper triangular matrices.

▶ **Theorem 14.** *For upper-triangular matrices, for $d \leq n$ odd, we have a lower bound of*

$$R(U_n(\mathbb{F}), d) \geq \binom{n+d}{n}.$$

Proof. We will be using flattening arguments similar to the one for general matrices. The multiplication tensor for the d-fold multiplication of upper triangular matrices looks as follows:

$$\mathbf{U}^{\otimes d} = \sum_{i_1, j_1, \ldots, i_{d+1}, j_{d+1} \in [n]} t_{i_1, j_1, \ldots, i_{d+1}, j_{d+1}} e^*_{i_1 j_1} \otimes \ldots \otimes e^*_{i_d j_d} \otimes e_{j_{d+1} i_{d+1}}$$

with $t_{i_1, j_1, \ldots, i_{d+1}, j_{d+1}} = (\prod_{k<d} \delta_{j_k i_{k+1}}) \cdot \delta_{j_d j_{d+1}} \cdot \delta_{i_{d+1} i_1}$ when $i_k \leq j_k$ for all $k \in [d]$, otherwise it will be 0. This means $t_{i_1, j_1, \ldots, i_{d+1}, j_{d+1}} = 1$ when $i_k = j_{k+1}$. From the upper triangular condition, we also have $i_k \leq j_k$.

We again flatten the tensor into a $(n^2)^{(d+1)/2} \times (n^2)^{(d+1)/2}$ matrix. As we saw in the proof of Theorem 12, each row or column has exactly one 1 and the matrix has full rank. With the extra condition of being upper triangular, we see that the rows will be non-zero if $i_1 \leq j_1, \ldots, i_d \leq j_d$, and as the columns are fixed with $j_1 = i_2 \leq j_2 = i_3 \ldots$, we basically get the rows with $i_1 \leq j_1 \leq i_3 \leq j_3 \leq \ldots \leq i_d \leq j_d$ will have exactly one 1. A similar thing can be done for the columns, with $j_{d+1} \leq i_2 \leq j_2 \leq i_4 \leq j_4 \leq \ldots \leq i_{d+1}$ with exactly one element 1. Therefore, the rank of the matrix will be the number of $i_1 \leq j_1 \leq i_3 \leq j_3 \leq \ldots \leq i_d \leq j_d$ sequences with $i, j \in [n]$. This number is $\binom{n+d}{n}$. ◀

7.2 Upper Bound for $d \gg n$

We see that unlike the case where $A/\operatorname{rad}(A)$ is noncommutative, the complexity flattens down as d becomes larger than n, becoming polynomial in d (for fixed n). We have the following upper bound:

▶ **Theorem 15.** *For multiplying upper-triangular matrices, for $d \gg n$, we have an upper bound of*

$$R(U_n(K), d) \leq O\left(\frac{(2d)^n}{\sqrt{n}}\right).$$

Proof. We can express the multiplication of d upper triangular matrices as

$$M_1 \cdot M_2 \cdot \ldots \cdot M_d.$$

Additionally, we can deconstruct any upper triangular matrix M_i as $M_i = D_i + N_i$, where D_i is a diagonal matrix and N_i has zeroes on the diagonal. We note that multiplying n upper triangular matrices which have zeroes on the diagonal yields 0, because they are in the radical. Then,

$$M_1 \cdot M_2 \cdot \ldots \cdot M_d = (D_1 + N_1)(D_2 + N_2), \ldots, (D_d + N_d).$$

We call the D_j's diagonal terms and the N_j's radical terms. We expand the product on the righthand side into a big sum. All the summands that have more than $n-1$ of the radical terms vanish. There are $\binom{d}{k}$ summands that have k radical terms, which in turn uniquely determines the positions of the diagonal terms in the summands. Therefore, there are $\sum_{k=0}^{n-1}\binom{d}{k}$ nonzero summands all together. We decompose each summand into rank-one tensors separately and in the trivial way. Each rank-one tensor in this decomposition corresponds to a subsequence of $(1, 2, \dots n)$ of the form $(i_1, j_1, i_2, j_2, \dots, i_k, j_k)$ such that $i_1 < j_1 = i_2 < j_2 = j_3 < \ldots = i_k < j_k$. The number of such subsequences is $\binom{n}{k+1}$ as choosing $i_1, i_2, \dots, i_k, j_k$ uniquely determines the subsequence. This sequence of indices corresponds to the basis elements e_{i_κ, j_κ} in each of the k radical terms. There is only one diagonal element that can stand between e_{i_κ, j_κ} and $e_{i_{\kappa+1}, j_{\kappa+1}}$, namely, e_{j_κ, j_κ}. (Recall that $j_\kappa = i_{\kappa+1}$.) Thus, each summand has a decomposition into rank-one tensors of length $\binom{n}{k+1}$.

Altogether, we have that

$$R(U_n(K), d) \leq \sum_{k=0}^{n-1} \binom{d}{k}\binom{n}{k+1}.$$

We have $\binom{n}{n/2} \geq \binom{n}{k+1}$ for all $1 \leq k \leq n-1$ and we have that $\sum_{k=0}^{n-1}\binom{d}{k} \leq d^n$. Thus, $R(U_n(K), d) \leq \binom{n}{n/2} \cdot d^n$. Using Stirling's approximation, we have that

$$R(U_n(K), d) \leq O\left(\frac{(2d)^n}{\sqrt{n}}\right). \qquad \blacktriangleleft$$

8 General algebras with commutative semisimple part

We see the above method can be used to give an upper bound on general algebras with commutative semisimple part, similar to the upper triangular matrices. We will only work over perfect fields (see [15] for a definition). Note that most fields are perfect, for instance, fields of characteristic zero, finite fields, and algebraically closed fields.

▶ **Theorem 16** (Wedderburn-Malcev Theorem, see [15]). *An algebra A over a perfect field can be written as $A = B \oplus \operatorname{rad}(A)$ with B being a subalgebra of A and $B \cong A/\operatorname{rad}(A)$.*

We see that in the situation of the theorem, we can write any element $x \in A$ as $x_B + x_{\operatorname{rad}(A)}$ with $x_B \in B$ and $x_{\operatorname{rad}(A)} \in \operatorname{rad}(A)$. This decomposition is unique. There are examples over non-perfect fields where such a B does not exist as a subalgebra of A, see [15].

The case when B is commutative is what is interesting to us, as the noncommutative case has an exponential lower bound anyway and we can remove the radical with Lemma 5. We focus on the case when $d \gg s$, where s is the smallest integer such that $(\operatorname{rad}(A))^s = \{0\}$. This number is also called the index of nilpotency. Obviously, $s \leq \dim A$.

When we consider a product $x_1 \cdot x_2 \cdot \ldots \cdot x_d$, we write it as $\prod_{i=1}^{d}(x_{i,B} + x_{i,\operatorname{rad}(A)})$. In this, terms with at least s terms from $\operatorname{rad}(A)$ will be 0. Therefore, when we expand the product as a sum, each nonzero summand contains at most $s-1$ factors from the radical.

Let ϕ^ℓ denote the $(\ell+1)$-linear map $B^\ell \times \operatorname{rad}(A) \to \operatorname{rad}(A)$, which takes $b_1, \dots, b_\ell \in B$ and $r \in \operatorname{rad}(A)$ and maps it to $b_1 \cdot \dots \cdot b_\ell \cdot r$. This is a restriction of the $(\ell+1)$-fold multiplication in A. We start with bounding the complexity of ϕ^ℓ.

▶ **Lemma 17.** *For all ℓ, $R(\phi^\ell) \leq \ell^D D^{D+2}$, where D denotes the dimension of A.*

Proof. ϕ^1 is the bilinear multiplication $B \times \operatorname{rad}(A) \to \operatorname{rad}(A)$. We can simply bound this by $R(A, 2) \leq D^2$, see Fact 4.

By Lemma 10, we have $R(B,\ell) \leq (\ell D)^D$. As seen above, $R(\phi^1) \leq D^2$. Consider a computation for the ℓ-fold multiplication in B, that is,

$$b_1 \cdot b_2 \cdot \ldots \cdot b_\ell = \sum_{i=1}^{r} F_{i,1}(b_1)F_{i,2}(b_2)\ldots F_{i,\ell}(b_\ell)\cdot w_i,$$

where $r \leq (\ell D)^D$. Furthermore, take a computation for ϕ^1,

$$b \cdot m = \sum_{j=1}^{r'} U_j(b)V_j(m)\cdot z_j$$

with $r' \leq D^2$. Plugging the first computation into the second, we get

$$\begin{aligned} b_1 \cdot \ldots \cdot b_\ell \cdot m &= \sum_{j=1}^{r'} U_j \left(\sum_{i=1}^{r} F_{i,1}(b_1)F_{i,2}(b_2)\ldots F_{i,\ell}(b_\ell)\cdot w_i \right) V_j(m)\cdot z_j \\ &= \sum_{j=1}^{r'}\sum_{i=1}^{r} F_{i,1}(b_1)F_{i,2}(b_2)\ldots F_{i,\ell}(b_\ell)U_j(w_i)V_j(m)\cdot z_j \\ &= \sum_{j=1}^{r'}\sum_{i=1}^{r} F_{i,1}(b_1)F_{i,2}(b_2)\ldots F_{i,\ell}(b_\ell)V_j(m)\cdot (U_j(w_i)z_j). \end{aligned}$$

Thus $R(\phi^\ell) \leq rr' = \ell^D D^{D+2}$. ◀

Concatenating k of the mappings ϕ_ℓ, we obtain an upper bound in a similar fashion to the upper triangular matrices.

▶ **Theorem 18.** *Let A be an algebra of dimension D over a perfect field $\mathbb{F}$ such that $A/\operatorname{rad} A$ is commutative. Then $R(A,d) \leq d^{(s-1)(D+1)} \cdot D^{(D+3)s}$, where s denotes the index of nilpotency of $\operatorname{rad} A$.*

Proof. Consider a product of d elements and write it as

$$x_1 \cdot x_2 \cdot \ldots \cdot x_d = \prod_{i=1}^{d}(x_{i,B} + x_{i,\operatorname{rad}(A)}).$$

We expand the product on the righthand side into a large sum. Summands with at least s factors from the radical will be zero by the definition of s. Let $k < s$. There are at most $\binom{d}{k}$ choices of k factors from the radical. As above, we will treat each summand separately. A summand is characterized by the k positions of the factors of the radical; they cut the product into $k+1$ parts of lengths $\ell_1, \ldots, \ell_{k+1}$ with $\ell_1 + \cdots + \ell_{k+1} = d - k$:

$$\begin{aligned} &x_{1,B}\cdots x_{\ell_1,B}\cdot x_{\ell_1+1,\operatorname{rad}(A)}x_{\ell_1+2,B}\cdots x_{\ell_1+\ell_2+2,B}\cdot x_{\ell_1+\ell_2+3,\operatorname{rad}(A)}\cdot x_{\ell_1+\ell_2+4,B}\cdots = \\ &\qquad \phi^{\ell_1}(x_{1,B},\ldots,x_{\ell_1,B},x_{\ell_1+1,\operatorname{rad}(A)})\cdot \phi^{\ell_2}(x_{\ell_1+2,B},\ldots,x_{\ell_1+\ell_2+2,B},x_{\ell_1+\ell_2+3,\operatorname{rad}(A)})\cdots \end{aligned}$$

(There are a total of $k+1$ factors on the righthand side, but we only wrote down the first two for the sake of readability.) For each ϕ^{ℓ_κ}, we can bound $R(\phi^{\ell_\kappa}) \leq \ell_\kappa^D D^{D+2}$ using Lemma 17. The last factor is a product of ℓ_{k+1} elements from B, we can simply bound the rank by $(\ell_{k+1}D)^D \leq \ell_{k+1}^D D^{D+2}$.

We trivially have $R(A, k+1) \leq D^{k+1}$, see Fact 4. By plugging computations for the ϕ^{ℓ_κ} in the computation for the $(k+1)$-fold multiplication in A, we get that there is a multilinear computation for our summand of length

$$\ell_1^D D^{D+2} \dots \ell_{k+1}^D D^{D+2} \cdot D^{k+1} \leq d^{kD} \cdot D^{(D+3)(k+1)}.$$

Altogether we can bound the multilinear complexity by

$$\sum_{k=0}^{s-1} \binom{d}{k} d^{kD} \cdot D^{(D+3)(k+1)} \leq d^{s-1} d^{(s-1)D} \cdot D^{(D+3)s} = d^{(s-1)(D+1)} \cdot D^{(D+3)s}. \qquad \blacktriangleleft$$

In the appendix, we present a more refined technique, that allows us to reduce the exponent of d in the upper bound.

9 Multilinear Alder–Strassen theorem

Finally, we prove the multilinear generalization of the Alder–Strassen theorem.

▶ **Theorem 19.** *Let A be a finite dimensional associative algebra with k maximal twosided ideals. Then $R(A, d) \geq d \cdot \dim A - (d-1)k$.*

Proof. We can assume $d \geq 3$, since the case $d = 1$ is trivial and the case $d = 2$ is the classical Alder–Strassen theorem.

By Lemma 5,

$$R(A, d) \geq R(A/\operatorname{rad} A, d) + d \cdot \dim \operatorname{rad} A. \tag{3}$$

Since A has k maximal twosided ideals, $A/\operatorname{rad} A$ has k maximal twosided ideals, too. and is of the form $A/\operatorname{rad} A = B_1 \times \dots \times B_k$ with B_κ being simple algebras. Each $B_\kappa = M_{n_\kappa}(D_\kappa)$ with D_κ being a division algebra. Assume that $n_1 = \dots n_j = 1$ and $n_{j+1}, \dots, n_k > 1$. That means that $B_1, \dots, B_j$ are division algebras.

We next prove that

$$R(A/\operatorname{rad} A, d) \geq d \cdot \dim(B_1 \times \dots \times B_j) - (d-1)j + R(B_{j+1} \times \dots \times B_k). \tag{4}$$

This is done by showing by induction that for all $1 \leq i \leq j$,

$$R(B_i \times \dots \times B_k) \geq d \cdot \dim B_i - (d-1) + R(B_{i+1} \times \dots \times B_k). \tag{5}$$

To prove Equation (5), we will inductively construct vector spaces $V_1, \dots, V_d$ such that $V_\delta + B_i \times \{0\} \times \dots \times \{0\} = B_i \times \dots \times B_k$ for all $1 \leq \delta \leq d$, $(1, 0, \dots, 0) \in V_\delta$ for $1 \leq \delta \leq d-1$, and

$$\begin{aligned} R(\phi) &\geq \delta \cdot \dim B_i - \delta + R(\phi|_{V_1 \times \dots \times V_\delta \times (B_i \times \dots \times B_k) \times \dots \times (B_i \times \dots \times B_k)}) \quad \text{for } 1 \leq \delta \leq d-1, \\ R(\phi) &\geq d \cdot \dim B_i - d + 1 + R(\phi|_{V_1 \times \dots \times V_d}), \end{aligned} \tag{6}$$

where ϕ is the d-fold multiplication in $B_i \times \dots \times B_k$. The proof is similar to the one of Theorem 8. Consider a multilinear computation for the d-fold multiplication in $B_i \times \dots \times B_k$, i.e.,

$$x_1 \cdot x_2 \cdot \ldots \cdot x_d = \sum_{i=1}^{r} F_{i,1}(x_1) F_{i,2}(x_2) \dots F_{i,d}(x_d) \cdot w_i.$$

Let $N = \dim B_i$. Similar to before, we can achieve that $F_{1,1}, \ldots, F_{N,1}$ restricted to $B_i \times \{0\} \times \cdots \times \{0\}$ is a basis of $(B_i \times \{0\} \times \cdots \times \{0\})^*$. Let $a_1, \ldots, a_N$ be the dual basis. Like before, we can assume that $a_N = (1, 0, \ldots, 0)$ using sandwiching. We set $V_1 = \bigcap_{\nu=1}^{N-1} \ker F_{\nu,1}$. Then V_1 has the desired properties. Restricting to V_1 trivializes $N-1$ of the multilinear products, namely, the products $1, \ldots, N-1$. Since $(1, 0, \ldots, 0) \in V_1$, we can conclude that w.l.o.g. $F_{N-1,2}, \ldots, F_{2N-1,2}$ restricted to $B_i \times \{0\} \times \cdots \times \{0\}$ form a basis and construct V_2 and so on. Since the induction stops at d, the dimension of the space V_d can be by one smaller, since we do not need $(1, 0, \ldots, 0) \in V_d$. Therefore, restricting to V_d trivializes even N products and we obtain Equation (6).

$B_i \times \{0\} \times \cdots \times \{0\}$ is a twosided ideal in $B_i \times \cdots \times B_k$ with the property that $B_{i+1} \times \cdots \times B_k \cong B_i \times \cdots \times B_k / B_i \times \{0\} \times \cdots \times \{0\}$. Thus (5) follows from (6) in the similar way like in the end of the proof of Lemma 5.

Finally, we prove that

$$R(B_{j+1} \times \cdots \times B_k) \geq d \cdot \dim(B_{j+1} \times \cdots \times B_k). \tag{7}$$

This will finish the proof. To prove the last equation, we use Theorem 13 and get $R(B_{j+1} \times \cdots \times B_k) \geq \sum_{i=j+1}^{k} \dim D_i \cdot n_i^d$. (The exponent is d instead of $d+1$, since d might be even.) Since $\dim B_i = n_i^2 \dim D_i$, we are done when we can show that $n_i^d \geq d \cdot n_i^2$. The latter inequality is implied by $n_i^{d-2} \geq d$. Since $n_i \geq 2$, this is true when $d \geq 4$. In the case when $d = 3$, note that $n_i^{d-1} \geq d$ is sufficient, since d is odd. Combining (3), (4), and (7) finishes the proof. ◀

For $d \geq 3$, the proof in fact yields the stronger lower bound $d \cdot \dim A - (d-1)j$, where j is the number of division algebras in the decomposition of $A/\operatorname{rad} A$ into simple parts.

References

1 A. Alder and V. Strassen. On the algorithmic complexity of associative algebras. *Theoret. Comput. Sci.*, 15:201–211, 1981.

2 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

3 Markus Bläser. Lower bounds for the bilinear complexity of associative algebras. *Comput. Complexity*, 9:73–112, 2000.

4 Markus Bläser. A complete characterization of the algebras of minimal bilinear complexity. *SIAM J. Comput.*, 34(2):277–298, 2004. doi:10.1137/S0097539703438277.

5 Markus Bläser. Fast matrix multiplication. *Theory Comput.*, 5:1–60, 2013. doi:10.4086/toc.gs.2013.005.

6 Markus Bläser and Bekhan Chokaev. Algebras of minimal multiplicative complexity. In *Proceedings of the 27th Conference on Computational Complexity, CCC 2012, Porto, Portugal, June 26-29, 2012*, pages 224–234. IEEE Computer Society, 2012. doi:10.1109/CCC.2012.13.

7 Markus Bläser and Christian Ikenmeyer. Introduction to geometric complexity theory. Lecture notes, 2018. URL: https://pcwww.liv.ac.uk/~iken/teaching_sb/summer17/introtogct/gct.pdf.

8 Nader H. Bshouty. Multilinear complexity is equivalent to optimal tester size. *Electron. Colloquium Comput. Complex.*, page 11, 2013. URL: https://eccc.weizmann.ac.il/report/2013/011, arXiv:TR13-011.

9 Nader H. Bshouty. Testers and their applications. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 327–352. ACM, 2014. doi:10.1145/2554797.2554828.

10 Harry Buhrman, Matthias Christandl, and Jeroen Zuiddam. Nondeterministic quantum communication complexity: the cyclic equality game and iterated matrix multiplication. In Christos H. Papadimitriou, editor, *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 24:1–24:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ITCS.2017.24.

11 Peter Bürgisser, Michael Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*. Springer, 1997.

12 Matthias Christandl and Jeroen Zuiddam. Tensor surgery and tensor rank. *Comput. Complex.*, 28(1):27–56, 2019. doi:10.1007/s00037-018-0164-8.

13 P. M. Cohn. *Algebra*, volume 3. Wiley, 1991.

14 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progression. *J. Symbolic Computation*, 9:251–280, 1990.

15 Yurij A. Drozd and Vladimir V. Kirichenko. *Finite Dimensional Algebras*. Springer, 1994.

16 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

17 J. M. Landsberg. New lower bounds for the rank of matrix multiplication. *SIAM J. Comput.*, 43(1):144–149, 2014. doi:10.1137/120880276.

18 J. M. Landsberg. *Geometry and Complexity Theory*. Cambridge University Press, 2017.

19 Noam Nisan and Avi Wigderson. Lower bounds for arithmetic circuits via partial derivatives (preliminary version). In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 16–25. IEEE Computer Society, 1995. doi:10.1109/SFCS.1995.492458.

20 Richard S. Pierce. *Associative Algebras*. Springer, 1982.

21 Ramprasad Saptharishi et al. A selection of lower bounds in arithmetic circuit complexity. Version 2021-07-27. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/tag/v9.0.3.

22 Andrew J. Stothers. *On the complexity of matrix multiplication*. PhD thesis, The University of Edinburgh, 2010.

23 Volker Strassen. Gaussian elimination is not optimal. *Num. Math.*, 13:354–356, 1969.

24 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Ann. ACM Symp. on Theory of Computing*, pages 887–898, 2012.

25 Shmuel Winograd. On multiplication of 2×2–matrices. *Lin. Alg. Appl.*, 4:381–388, 1971.

A Improvements of Theorem 18

Wedderburn's Theorem holds in a similar manner for modules over simple algebras. If A is an algebra, let $A^{n\times m}$ denote the vector space of all $n \times m$–matrices with entries from A.

▶ **Theorem 20** (Wedderburn). *Let A be a simple algebra with $A \cong M_n(D)$ for some division algebra D. For every A-left module $M \neq \{0\}$, there is a (unique) integer $m \geq 1$ such that M is isomorphic to $D^{n\times m}$.*

If C and D are algebras and M is a C-left module that is also a D-right module, then the module M is called a (C, D)-*bimodule*, if in addition $(am)b = a(mb)$ for all $a \in C$, $m \in M$, and $b \in D$. If $C = D$, M is also called a C-bimodule for short.

While the above bound of Theorem 18 is polynomial in d (when the dimension of the algebra is fixed), the algorithm can be improved and the exponent can be reduced. We sketch this approach below, but the actual bounds depend on a lot of parameters of the algebra, captured by the so-called path diagram of the algebra.

We decompose the semisimple algebra $B = B_1 \oplus \cdots \oplus B_n$ into simple algebras. (Here, the decomposition is written additively, and we consider the B_ν to be subspaces of B.) Each B_ν is a commutative division algebra. Let ℓ_ν be its dimension.

Let e_ν be the unit element of the algebra B_ν. It is well known [15], that $1 = e_1 + \cdots + e_n$ and that the $e_1, \ldots, e_n$ annihilate each other, that is, $e_\mu e_\nu = 0$ for all $\mu \neq \nu$. Write

$$A = 1 \cdot A \cdot 1 = (e_1 + \cdots + e_n)A(e_1 + \cdots + e_n) = \sum_{\mu,\nu=1}^{n} e_\mu A e_\nu.$$

Since $e_1, \ldots, e_n$ annihilate each other, the sum of vector spaces on the right hand side is direct. In the same way, we can decompose $\operatorname{rad} A$.

▶ **Fact 21.**

1. $\mathbb{F}[X] \otimes \mathbb{F}[Y] = \mathbb{F}[X, Y]$ *whenever* X *and* Y *are disjoint sets of variables.*
2. *Let* $I \subseteq \mathbb{F}[X]$ *and* $J \subseteq \mathbb{F}[Y]$ *be ideals. Then* $F[X]/I \otimes \mathbb{F}[Y]/J = \mathbb{F}[X, Y](I + J)$.

Since each B_i is commutative, we can write is as $\mathbb{F}[X_i]/I_i$ for some set of variables X_i and an ideal I_i. We assume that the sets of variables are pairwise disjoint.

The opposite algebra A^{opp} is the algebra obtained by "reversing" the multiplication of A. The multiplication $*$ in A^{opp} is defined by $a * b = b \cdot a$. When A is commutative, then A^{opp} and A are isomorphic (as algebras).

Let $R_{i,j} := e_i(\operatorname{rad} A)e_j$. The $R_{i,j}$ are B left-modules and B right-modules, so they are B-bimodules. However, the only nontrivial multiplication from the left is with elements from B_i and the only nontrivial multiplication from the right is with B_j. So it is more natural to view $R_{i,j}$ as a (B_i, B_j)-bimodule. A (B_i, B_j)-bimodule is isomorphic to a $B_i \otimes B_j^{\text{opp}}$-left module. Since B_j is commutative, this is in turn isomorphic to a $B_i \otimes B_j$ module. Take an element $m \in R_{i,j}$. $(B_i \otimes B_j)m$ is a submodule of $R_{i,j}$ and since $B_i \otimes B_j$ is a quotient of a polynomial ring, so is the submodule.

Let $D = \dim A$. There are $\binom{d}{k}$ choices for the factors from the radical, $0 \leq k \leq s$. For each factor of the radical, we choose a subspace R_{i_κ, j_κ}, $1 \leq \kappa \leq k$, such that $i_{\kappa+1} = j_\kappa$. So essentially, we sum over all path in the so-called path diagram of the algebra, see [15], which describes the structure of the radical. Along such a path, by the above consideration, the multiplication can be simulated by a polynomial multiplication, so we can do the multiplication "in one stroke" and do not need to glue different pieces like we did in the proof above.

Strongly Hyperbolic Unit Disk Graphs

Thomas Bläsius
Karlsruhe Institute of Technology, Germany

Tobias Friedrich
Hasso Plattner Institute, University of Potsdam, Germany

Maximilian Katzmann
Karlsruhe Institute of Technology, Germany

Daniel Stephan
GSV Algorithm Consulting UG (haftungsbeschränkt) & Co. KG, Potsdam, Germany

Abstract

The class of Euclidean unit disk graphs is one of the most fundamental and well-studied graph classes with underlying geometry. In this paper, we identify this class as a special case in the broader class of *hyperbolic unit disk graphs* and introduce *strongly hyperbolic unit disk graphs* as a natural counterpart to the Euclidean variant. In contrast to the grid-like structures exhibited by Euclidean unit disk graphs, strongly hyperbolic networks feature hierarchical structures, which are also observed in complex real-world networks.

We investigate basic properties of strongly hyperbolic unit disk graphs, including adjacencies and the formation of cliques, and utilize the derived insights to demonstrate that the class is useful for the development and analysis of graph algorithms. Specifically, we develop a simple greedy routing scheme and analyze its performance on strongly hyperbolic unit disk graphs in order to prove that routing can be performed more efficiently on such networks than in general.

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

Keywords and phrases hyperbolic geometry, unit disk graphs, greedy routing, hyperbolic random graphs, graph classes

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.13

Related Version *Full Version*: `https://arxiv.org/abs/2107.05518` [3]

Funding This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Grant No. 390859508.

1 Introduction

Studying networks in terms of *graph classes* based on certain properties is a fundamental tool in graph theory. Instead of having to consider all possible graphs, we can focus on the ones in a certain class, which allows us to get a more fine-grained understanding of their structural properties and the complexity of graph problems. Additionally, it facilitates the development of more efficient algorithms that are tailored towards the characteristics of the considered networks.

Different classes can be utilized in different contexts. For example, the characteristics of wireless communication networks are captured naturally in *Euclidean unit disk graphs* [6, 15], i.e., graphs where vertices can be identified with disks of equal size in the Euclidean plane and any two are adjacent if and only if their disks intersect. In this paper, we use the following formalization. Let $G = (V, E)$ be an undirected graph. A *(Euclidean) unit disk representation* of G is a mapping $\phi\colon V \to \mathbb{R}^2$ together with a *threshold radius* R such that $\{u, v\} \in E$ if and only if the distance between $\phi(u)$ and $\phi(v)$ is at most R. Then, the graph G

© Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, and Daniel Stephan;
licensed under Creative Commons License CC-BY 4.0

is a *(Euclidean) unit disk graph* if it has a unit disk representation. In such graphs, the generally NP-complete problem of finding a maximum clique can be solved in polynomial time [6, 24], and routing can be performed more efficiently than in general graphs [16].

In this paper, we study a related graph class where the Euclidean ground space is replaced with the hyperbolic plane. The result is a generalization of the Euclidean variant, containing networks with a broader range of structural properties. Formally, a graph G is a *hyperbolic unit disk graph*, if there exists a *hyperbolic unit disk representation* $\phi\colon V \to \mathbb{H}^2$ together with a threshold radius R, such that $\{u, v\} \in E$, if and only if the hyperbolic distance between the vertex representations is at most $d_{\mathbb{H}^2}(\phi(u), \phi(v)) \leq R$. We note that the threshold radius R is part of the representation and can thus depend on the graph. The choice of R does not matter in Euclidean space, as scaling R and all coordinates $\phi(\cdot)$ by the same factor yields the same adjacencies. In contrast, there is no scaling operation in the hyperbolic plane that leaves relative distances intact.[1] As a result, the size of the considered region and the threshold radius *do* have an impact on the structure of the graphs in the hyperbolic setting.

To understand this effect, which is visualized in Figure 1, first consider some region, say a disk D of radius R', in the Euclidean plane and assume we distribute vertices evenly in D. Then the resulting Euclidean unit disk graph resembles a *grid-like* structure (with a density depending on the threshold radius R and the radius R' of D). That is, in the sparse setting, we only find small cliques, while separators and treewidth as well as the diameter are large, and we observe a homogeneity among the vertices, in the sense that all neighborhoods feature similar characteristics. Essentially, as in a grid, the graph looks the same no matter from which vertex it is viewed.

As the hyperbolic plane resembles the Euclidean plane locally, we can achieve the same grid-like structures by choosing a very small radius R' and an even smaller threshold radius R (Figure 1 (left)). In fact, by scaling the Euclidean unit disk representation of a graph into a sufficiently small region, we can realize the same adjacencies in the hyperbolic plane and obtain the following.

▶ **Theorem 1.** *Every Euclidean unit disk graph is a hyperbolic unit disk graph.*

Beyond that, we can increase the radii R' and R. Then, the grid-like structures start to vanish and *hierarchical* structures begin to form (Figure 1 (center)). Eventually, we reach the *strongly hyperbolic* setting where only hierarchical and no grid-like structures remain (Figure 1 (right)). There, vertices are rather heterogeneous, with respect to their degree and what neighborhoods look like. The diameter is small, while large cliques can form. We note that the treewidth is also large, just as in grid-like graphs. However, in hierarchical graphs this is an artifact of the large cliques, while in grid-like graphs we observe large treewidth *despite* the fact that only small cliques form. In hierarchical networks vertices connect via hubs, which connect via larger hubs, and so on. The hubs explicitly exhibit the hierarchy in the graph structure. As a result, the graph looks very different when viewed from vertices on different levels in the hierarchy.

Formally, we say that a graph is a *strongly hyperbolic unit disk graph* if it admits a hyperbolic unit disk representation in which ϕ maps all vertices to points within a disk whose radius matches the threshold ($R' = R$). For better understanding, we recommend using the interactive visualization[2], which lets the user change the size of the ground space, allowing to smoothly transition between Euclidean and strongly hyperbolic unit disk graphs.

[1] Under the common assumption that the curvature is -1, such a scaling operation does not exist. The term "unit disk" is still justified as we could instead fix $R = 1$ and allow for different curvatures.

[2] `https://thobl.github.io/hyperbolic-unit-disk-graph/`

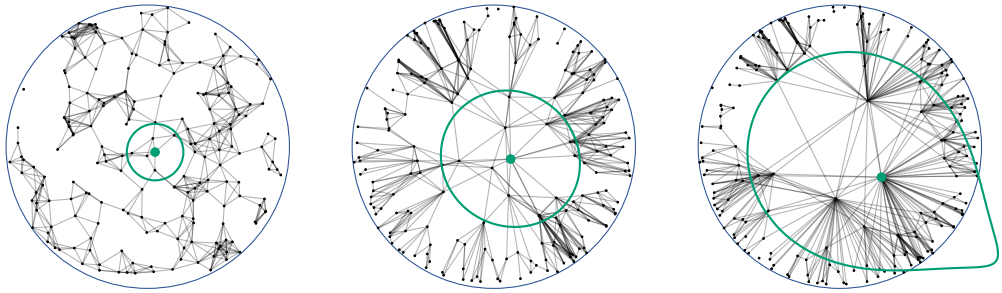


Figure 1 Hyperbolic unit disk graphs with different ground space and threshold radii. The representations have been scaled such that the ground spaces *appear* to have the same size, while their actual sizes are denoted by R'. **(Left)** The ground space is very small and the threshold radius even smaller, leading to grid-like structures. **(Center)** Ground space and threshold radius are increased, hierarchies start form but grid like structures remain. **(Right)** Ground space and threshold have the same large value, leading to hierarchical structures.

To paint the big picture, hyperbolic unit disk graphs comprise two extremes: Euclidean unit disk graphs with grid-like structures on one side and strongly hyperbolic unit disk graphs with hierarchical structures on the other. Therefore, if we want to design algorithms for grid-like structures, it makes sense to analyze them on Euclidean unit disk graphs. For hierarchical structures, strongly hyperbolic unit disk graphs are a good choice.

Related Concepts

To the best of our knowledge, intersection graphs of hyperbolic unit disks, or hyperbolic unit balls, have so far only been considered by Kisfaludi-Bak [17]. There, for every $\rho > 0$, a graph is said to be in the graph class $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$ (*UBG = unit ball graph*) if its vertices can be mapped into $\mathbb{H}^d$ such that vertices have distance at most 2ρ if and only if they are adjacent. There are two core differences compared to our definition of hyperbolic unit disk graphs. First, it allows for higher dimensions. Secondly, it is parameterized by the radius, i.e., $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$ describes an infinite family of graph classes rather than a single class.

This second difference is somewhat subtle but rather important. Consider the class $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$ for a fixed radius ρ. Moreover, assume we want to study graphs in $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$ that are sparse; for the sake of argument, assume constant average degree. Then, for an increasing number of vertices n, the region of $\mathbb{H}^d$ spanned by the vertices has to grow, as otherwise the density of the graph grows with n. Thus, for sufficiently large n, the radius ρ is arbitrarily small compared to the region spanned by the vertices, yielding grid-like structures (see discussion above). Thus, for fixed ρ, large graphs in $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$ are grid-like rather than hierarchical. This means that asymptotic statements for the classes $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$ do not translate to the hierarchical structures in the class of strongly hyperbolic unit disk graphs.

A second related concept are hyperbolic random graphs [18], which are basically random strongly hyperbolic unit disk graphs. A hyperbolic random graph is obtained by assigning each vertex a random point in a disk of radius $R \approx 2\log(n)$ and connecting two vertices if and only if their distance is at most R. This yields graphs that resemble certain real-world networks, as they have small diameter, high clustering, and a power-law degree distribution whose exponent can be adjusted using the probability distribution of the vertex positions.

This is not the case for graphs in $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$ as the connection radius cannot increase with the graph size. Nevertheless, every hyperbolic random graph is a strongly hyperbolic unit disk graph and thus any statement shown for the latter also holds for the former.

Hyperbolic random graphs have also been studied in a noisy setting, where, with some small probability, distant vertices are adjacent and close vertices are not adjacent. Similarly, Kisfaludi-Bak [17] also studies a noisy variant of the class $\mathrm{UBG}_{\mathbb{H}^d}(\rho)$. It would be interesting to also study (strongly) hyperbolic unit disk graphs in a noisy setting. This is, however, beyond the scope of this paper and left for future research.

Contribution

Beyond the generalization of Euclidean unit disk graphs to hyperbolic unit disk graphs, we identify strongly hyperbolic unit disk graphs as a natural counterpart to the Euclidean special case and provide the first insights into their structural and algorithmic properties. In particular, we study fundamental criteria relating the coordinates of vertices to their adjacency and investigate the formation of cliques (Section 2). Using these insights, we follow up on prior empirical efforts towards understanding how an underlying hyperbolic geometry facilitates efficient routing on internet-like networks [4, 21], and utilize strongly hyperbolic unit disk graphs to obtain theoretical performance guarantees, proving that routing in such networks can be performed more efficiently than in general (Section 3). While similar results have been obtained on the grid-like Euclidean unit disk graphs [16], our analysis covers networks with hierarchical structures. In particular, it includes hyperbolic random graphs, which are used to represent real-world complex networks like the internet [4], where routing plays an important role. By developing a simple routing scheme, which is interesting in its own right, we show that greedy routing on such graphs can be performed with a stretch of at most 3, while asymptotically almost surely requiring at most $\mathcal{O}(\log^4 n)$ bits of storage per vertex and taking $\mathcal{O}(\log^2 n)$ time per routing decision. We note that some proofs are deferred to the full version of the paper [3].

2 Strongly Hyperbolic Unit Disk Graphs

Throughout the paper we consider the *polar-coordinate model* of the hyperbolic plane $\mathbb{H}^2$. There, we have a designated *pole* $O \in \mathbb{H}^2$, together with a *polar axis*, i.e., a reference ray starting at O. A point p is identified by its *radius* $r(p)$, denoting the hyperbolic distance to O, and its *angle* $\varphi(p)$, denoting the angular distance between the polar axis and the ray from O through p. In our figures we interpret these values as polar coordinates in the Euclidean plane. The disk of radius R centered at p is denoted by $D_R(p)$. When $p = O$ we simply write D_R. The hyperbolic distance between points p and q is given by

$$d_{\mathbb{H}^2}(p,q) = \operatorname{acosh}\Big(\cosh\big(r(p)\big)\cosh\big(r(q)\big) - \sinh\big(r(p)\big)\sinh\big(r(q)\big)\cos\big(\Delta_\varphi(p,q)\big)\Big), \qquad (1)$$

where $\cosh(x) = (e^x + e^{-x})/2$, $\sinh(x) = (e^x - e^{-x})/2$, and $\Delta_\varphi(p,q) = \pi - |\pi - |\varphi(p) - \varphi(q)||$ denotes the angular distance between p and q. Without loss of generality, we assume that the representation ϕ of a strongly hyperbolic unit disk graph maps the vertices into a disk of radius R that is centered at O. For the sake of readability, we typically associate a vertex v with its mapping $\phi(v)$ and denote the set of vertices lying in a region $A \subseteq D_R$ with $V(A)$.

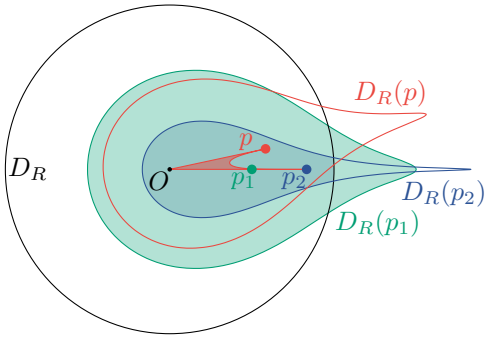


Figure 2 Visualization of the proof of Lemma 2. Point p_1 has a smaller radius than p_2, both having the same angular coordinate. Consequently, $D_R(p_1)$ (green region) is a superset of $D_R(p_2) \cap D_R$ (blue region). The triangle formed by the points p, p_2, and O is contained in $D_R(p)$ (both red).

2.1 Adjacency

Similar results to the ones described in this subsection have been determined on hyperbolic random graphs before (see, e.g., [13]). Here we verify under which requirements they also hold on strongly hyperbolic unit disk graphs. By definition, two vertices in a strongly hyperbolic unit disk graph G are adjacent, if and only if their hyperbolic distance is at most R. Consequently, we can imagine that each vertex v is equipped with a neighborhood disk $D_R(v)$. That is, $N(v) = V(D_R(v))$. The following lemma shows that moving such a neighborhood disk closer to the center of D_R only increases the region of D_R that it covers.

▶ **Lemma 2.** *Let R be a radius and let $p_1, p_2 \in D_R$ be points with $r(p_1) \leq r(p_2)$ and $\varphi(p_1) = \varphi(p_2)$. Then, $D_R(p_1) \supseteq D_R(p_2) \cap D_R$.*

Proof. Let $p \in D_R(p_2) \cap D_R$ be a point and note that $d_{\mathbb{H}^2}(p, p_2) \leq R$. Now consider the triangle spanned by the points p, p_2, and the origin O. This triangle is completely contained in the disk $D_R(p)$, as $d_{\mathbb{H}^2}(p, p_2) \leq R$ and $r(p) \leq R$, as shown in Figure 2. Since disks are convex and p_1 lies on the line from O to p_2, it is part of the triangle and therefore also contained in the disk. Consequently, $d_{\mathbb{H}^2}(p, p_1) \leq R$ and thus $p \in D_R(p_1)$. ◀

Consequently, moving a vertex towards the center does not decrease its neighborhood.

▶ **Corollary 3.** *Let G be a strongly hyperbolic unit disk graph with radius R and let v_1, v_2 be vertices with $r(v_1) \leq r(v_2) \leq R$ and $\varphi(v_1) = \varphi(v_2)$. Then, $N(v_1) \supseteq N(v_2)$.*

In the following, we investigate in greater detail under which circumstances two vertices are adjacent. Consider two vertices v_1 and v_2 in G with radii r_1 and r_2, respectively. Clearly, the two are adjacent if $r_1 + r_2 \leq R$. When $r_1 + r_2 > R$, the hyperbolic distance between them depends on their angular distance $\Delta_\varphi(v_1, v_2)$. More precisely, for vertices of fixed radii, increasing the angular distance also increases the hyperbolic distance. Let $\theta(r_1, r_2)$ denote the angular distance, such that the hyperbolic distance between v_1 and v_2 is exactly R. That is, for $\Delta_\varphi(v_1, v_2) \leq \theta(r_1, r_2)$ we have $d_{\mathbb{H}^2}(v_1, v_2) \leq R$, meaning v_1 and v_2 are adjacent. We can compute $\theta(r_1, r_2)$ by using the hyperbolic distance function in Equation (1), setting the distance equal to R, and solving for the angular distance. That is,

$$\theta(r_1, r_2) = \operatorname{acos}\left(\frac{\cosh(r_1)\cosh(r_2) - \cosh(R)}{\sinh(r_1)\sinh(r_2)}\right). \tag{2}$$

Tight asymptotic bounds on $\theta(r_1, r_2)$ have been derived before [19, Lemma 3.2]. The following lemma holds for all $R > 0$.

▶ **Lemma 4.** *Let $R > 0$ and $r_1, r_2 \in (0, R]$ with $r_1 + r_2 \geq R$ be given. Then,*

$$2\sqrt{e^{R-r_1-r_2} + e^{-R-r_1-r_2} - (e^{-2r_1} + e^{-2r_2})} \leq \theta(r_1, r_2) \leq \pi\sqrt{e^{R-r_1-r_2}}.$$

We note that, while the above bounds are easier to work with than the exact function and are generally applicable due to the few constraints on the considered radii, the lower bound is still a bit tedious to work with. However, by introducing some minor requirements, we can obtain a slightly weaker bound that can be worked with more easily.

▶ **Corollary 5.** *Let $R \geq 1$ and $r_1, r_2 \in (0, R]$ with $r_1 + r_2 \geq R$ and $|r_1 - r_2| \leq R - 1$ be given. Then,*

$$\sqrt{e^{R-r_1-r_2}} \leq \theta(r_1, r_2) \leq \pi\sqrt{e^{R-r_1-r_2}}.$$

Apart from the above bounds, we highlight another property of the function $\theta(r_1, r_2)$, for the special case where $r_1 = r_2$.

▶ **Lemma 6.** *The function $\theta(r, r)$ is monotonically decreasing for $r \geq 0$.*

2.2 Cliques

In the following, we examine how the underlying geometry affects the formation of cliques. We start by showing that the vertices lying in $D_R(p)$, having smaller radius than p, form two cliques. More precisely, we say that a vertex set $S \subseteq V$ can be *covered by k cliques*, if there exists a partitioning $S_1, \ldots, S_k$ of S such that the induced subgraphs $G[S_i]$ for $i \in [k]$ are cliques.

▶ **Lemma 7.** *Let G be a strongly hyperbolic unit disk graph with radius $R > 0$ and let $p \in D_R$ be a point with $r(p) = r$. Then, $V(D_R(p) \cap D_r)$ can be covered by two cliques.*

Proof. Assume without loss of generality that $\varphi(p) = 0$. We divide the region $D_R(p) \cap D_r$ into two halves A and A' containing all points with angles in $[0, \pi)$ and $[\pi, 2\pi)$, respectively, as illustrated in Figure 3 (left). The goal now is to show that the vertices in $V(A)$ and the ones in $V(A')$ induce a clique. More precisely, we show that this is the case for A. For symmetry reasons this then also holds for A'. Consider two vertices $v_1, v_2 \in A$ and assume without loss of generality that $\varphi(v_1) \leq \varphi(v_2)$. Since $v_2 \in A \subseteq D_R(p)$ and since by Lemma 2 moving $D_R(p)$ towards the origin increases the region of D_R that it covers, we know that v_2 is contained in the disk $D_R(p')$ for $p' = (r(v_1), 0)$ (dark green in Figure 3 (left)). It follows that $d_{\mathbb{H}^2}(p', v_2) \leq R$. Note that v_1 has the same radius as p' and that $\Delta_\varphi(p', v_2) \geq \Delta_\varphi(v_1, v_2)$. As established above, decreasing the angular distance between two points with fixed radii decreases their hyperbolic distance. Therefore, $d_{\mathbb{H}^2}(v_1, v_2) \leq d_{\mathbb{H}^2}(p', v_2) \leq R$, meaning v_1 and v_2 are adjacent. ◀

We note that the above lemma implies that for a vertex v, the neighbors with smaller radius than v form two cliques. We continue by investigating the number of cliques required to cover a strongly hyperbolic unit disk graph.

▶ **Lemma 8.** *Let G be a strongly hyperbolic unit disk graph with radius $R > 0$. Then, G can be covered by $\max\{2\pi\sqrt{2}, 2\pi e^{R/2}\}$ cliques.*

Proof. To prove the claim, we utilize the underlying geometry by covering the ground space D_R with a set of k disks $D^1, \ldots, D^k$, such that each $V(D^i)$ for $i \in [k]$ can be covered by two cliques. All of these disks have radius R and their centers lie on the boundary of

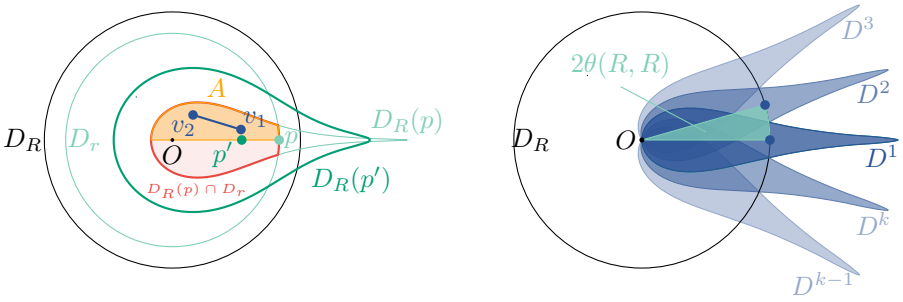


Figure 3 Covering strongly hyperbolic unit disk graphs with cliques. **(Left)** Visualization of the proof of Lemma 7. Vertices v_1, v_2 (blue) are in the half A (orange) of the region $D_R(p) \cap D_r$ (red) and are adjacent. **(Right)** Visualization of the proof of Lemma 8, showing five of the k disks $D^1, \dots, D^k$ (blue) and the angular distance between two consecutive centers (green).

the disk D_R. The center of the first disk has an angular coordinate of 0. All other disks D^i are placed at an angular distance of $2\theta(R,R)$ to their predecessor D^{i-1} in counterclockwise direction. See Figure 3 (right) for an illustration. As a consequence, the boundaries of two consecutive disks intersect on the boundary of D_R, which is therefore covered completely by the k disks. It follows that each vertex is contained in at least one disk D^i.

Since by Lemma 7 each $V(D^i)$ for $i \in [k]$ can be covered by two cliques, it suffices to show that $k \leq \max\{\pi\sqrt{2}, \pi e^{R/2}\}$ in order to finish the proof. To this end, recall that two consecutive disks are placed at an angular distance of $2\theta(R,R)$. Consequently, it takes $k = 2\pi/(2\theta(R,R)) = \pi/\theta(R,R)$ disks to cover the whole disk D_R. Using Lemma 4 we can conclude

$$\theta(R,R) \geq 2\sqrt{e^{-R} + e^{-3R} - 2e^{-2R}} = 2\sqrt{\left(e^{-R/2} - e^{-3/2 \cdot R}\right)^2} = 2e^{-R/2}(1 - e^{-R}).$$

It follows that k can be bounded by

$$k = \frac{\pi}{\theta(R,R)} \leq \frac{\pi}{2e^{-R/2}\left(1 - e^{-R}\right)} = \pi e^{R/2} \cdot \frac{1}{2\left(1 - e^{-R}\right)}. \tag{3}$$

We now distinguish between two cases depending on the size of R and start with $R < \log(2)$. Recall that the function $\theta(R,R)$ is monotonically decreasing in R (see Lemma 6). As a consequence, we have $\theta(R,R) \geq \theta(\log(2), \log(2))$. Then, it follows that

$$k \leq \frac{\pi}{\theta(\log(2), \log(2))} \leq \pi e^{\log(2)/2} \frac{1}{2\left(1 - e^{-\log(2)}\right)} = \pi\sqrt{2},$$

which we account for with the first part of the maximum. When $R \geq \log(2)$, note that we have $(1 - e^{-R}) \geq 1/2$. Consequently, we can bound the last fraction in Equation (3) by 1, which yields the claim. ◀

3 Routing in Strongly Hyperbolic Unit Disk Graphs

Throughout the remainder of the paper, we utilize strongly hyperbolic unit disk graphs to investigate the performance of *routing* on networks with underlying hyperbolic geometry. This is not only interesting since routing is one of the most fundamental graph problems, but is also particularly relevant on complex networks like the internet, which has previously been observed to fit well into the hyperbolic plane [4].

While finding a path between two vertices in an undirected network is typically rather simple, the internet is decentralized and does not allow for the use of a central data structure. Instead each vertex can only use local information to perform a *routing decision*, i.e., the decision to which vertex information is forwarded next such that it eventually reaches the target. This situation is further complicated by the fact that the internet consists of billions of vertices. In order to be able to handle a network of this scale, a routing scheme has to be optimized with respect to three criteria: the *space requirement* (the amount of information that the scheme uses to forward information), the *query time* (the time it takes to make a routing decision), and the *stretch* (how much longer the routed path is compared to a shortest path in the network, where the *length* of a path denotes the number of contained edges). Formally, a path between two vertices has *multiplicative stretch* $c \geq 1$ if it is at most c times longer than a shortest path between them. An *additive stretch* of ℓ denotes that the routed path contains at most ℓ more vertices than a shortest path. A *multiplicative stretch c with additive bound ℓ* denotes that the routed paths have stretch c *or* additive stretch ℓ. Note that this implies a multiplicative stretch of $\max\{c, 1 + \ell\}$.

There is a long line of research on how to obtain efficient routing schemes with respect to these criteria, together with tight bounds on the trade-offs that have to be made to optimize one criterion over another. In particular, it was shown that routing with a multiplicative stretch of $c \geq 1$ requires a total storage of $\Omega(n^{1+1/(2c+4)})$ bits in general graphs [23]. We refer to the full version of the paper for a comprehensive overview of related work [3].

In the following, we combine standard techniques in routing to obtain a simple routing scheme and analyze its performance on strongly hyperbolic unit disk graphs. For the special case of hyperbolic random graphs, a graph model that is used to represent complex networks like the internet [18], it improves below the above mentioned performance lower bounds.

3.1 Greedy Routing Scheme

A standard approach to routing in a decentralized setting is *greedy routing*, where the idea is to always forward a message to a neighbor of a vertex that is closer to the target.[3] When designing a greedy routing scheme, we therefore need to compute distances between vertices and select a suitable neighbor with respect to these distances. For a graph $G = (V, E)$ let $d\colon V \times V \to \mathbb{R}_{\geq 0}$ be a semi-metric on G. That is, for all $s, t \in V$ we have $d(s, t) \geq 0$, $d(s, t) = 0$ if and only if $s = t$, and $d(s, t) = d(t, s)$. We say that a greedy routing scheme routes *with respect to* d, if at s a message to t is forwarded to a neighbor v of s where $d(v, t) < d(s, t)$. Depending on d such a neighbor may not exist and the message cannot be forwarded, which is called *starvation*. In contrast, a routing scheme with guaranteed delivery is called *starvation-free* and it is known that greedy routing is starvation-free, if at every vertex $s \neq t$ there is a neighbor v with $d(v, t) < d(s, t)$ (see e.g. [28]). Moreover, we say that d is *integral* if it maps to the natural numbers, i.e., $d\colon V \times V \to \mathbb{N}$. Note that, if d is integral and routing with respect to d is starvation-free, the distance to the target decreases by at least one in each step. Thus, the length of the routed path between s and t is bounded by $d(s, t)$. When this is the case, we say that routing with respect to d is *d-bounded*.

Given a connected graph G, a natural choice for determining a distance between s and t is to use the length of a shortest path between them, which we denote with $d_G(s, t)$. Routing with respect to d_G is starvation-free and yields perfect stretch. However, d_G cannot be computed while simultaneously keeping the required space and query time low [12]. Therefore,

[3] The term *greedy* often refers to routing to a neighbor *closest* to the target. For us *closer* is sufficient.

we relax the constraint on routing with respect to exact graph distances and use upper bounds instead. This can be achieved by taking a subgraph G' of G and routing on G with respect to $d_{G'}$. The stretch of the resulting routing scheme depends on how well the distances in G' approximate the distances in G. Unfortunately, finding a subgraph with good stretch is hard in general [5, 22]. However, instead of routing with respect to the distances in a single subgraph, we can combine the distances in multiple subgraphs. To obtain a good stretch, it then suffices to find low-stretch subgraphs for small parts of the graph. To formalize this, we use a (c, ℓ, k)-*graph-cover* $\mathcal{C}$ of G, which is a collection of subgraphs of G, such that for all $s, t \in V$ there exists a connected subgraph G' in $\mathcal{C}$ with $d_{G'}(s,t) \leq c \cdot d_G(s,t)$ or $d_{G'}(s,t) \leq d_G(s,t) + \ell$, and every vertex $v \in V$ is contained in at most k graphs in $\mathcal{C}$. We say that $\mathcal{C}$ has multiplicative stretch c with additive bound ℓ. For two vertices s and t we define $d_{\mathcal{C}}(s,t) = \min_{G' \in \mathcal{C}} d_{G'}(s,t)$.

▶ **Lemma 9.** *Let G be a graph and let $\mathcal{C}$ be a (c, ℓ, k)-graph-cover of G. Then, greedy routing on G with respect to $d_{\mathcal{C}}$ has multiplicative stretch c with additive bound ℓ.*

To show that $d_{\mathcal{C}}$ can be computed efficiently, we use *distance labeling schemes* [11]. Such a scheme implements a semi-metric d by assigning each vertex a *distance label*, such that for two vertices s, t we can compute $d(s,t)$ by looking at their distance labels only. The *label size* of a distance labeling scheme denotes the maximum number of bits required to represent the label of a vertex. The *query time* denotes the time it takes to compute d using the labels. Given a graph-cover $\mathcal{C}$, a distance labeling scheme that implements $d_{\mathcal{C}}$ can be obtained by combining distance labeling schemes for the contained subgraphs.

▶ **Lemma 10.** *Let G be a graph and let $\mathcal{C}$ be a (c, ℓ, k)-graph-cover of G such that for every $G' \in \mathcal{C}$ there exists a distance labeling scheme that implements $d_{G'}$ with label size λ and query time q. Then, there exists a distance labeling scheme for G that implements $d_{\mathcal{C}}$ with label size $\mathcal{O}(k(\lambda + \log k + \log n))$ and query time $\mathcal{O}(kq)$.*

In order to perform a routing decision efficiently, we want to avoid performing a linear search over all neighbors. To this end, we need to be able to identify a neighbor directly, which can be done by assigning each neighbor v of s a unique *port* $p_s(v) \colon N(s) \to \{1, \dots, n\}$. Finding a neighbor of s that is closer to a target t with respect to a semi-metric d then boils down to determining the corresponding port. To this end, we can use a *port labeling scheme* that implements d. Such a scheme assigns each vertex in a graph a *port label* such that we can determine the port of a neighbor of s that is closer to t with respect to d, by only looking at the port labels of s and t. The corresponding label sizes and query times are defined analogous to how they are defined for distance labels.

Given a graph-cover $\mathcal{C}$, we can combine distance and port labels of the subgraphs in the cover, to obtain a port labeling scheme that implements $d_{\mathcal{C}}$.

▶ **Lemma 11.** *Let G be a graph and let $\mathcal{C}$ be a (c, ℓ, k)-graph-cover of G such that for every $G' \in \mathcal{C}$ there exist distance and port labeling schemes that implement $d_{G'}$ with label size λ and query time q. Then, there exists a port labeling scheme for G that implements $d_{\mathcal{C}}$ with label size $\mathcal{O}(k(\lambda + \log k + \log n))$ and query time $\mathcal{O}(kq)$.*

We are now ready to combine the above results to obtain our greedy routing scheme. To this end, we need to find (c, ℓ, k)-graph-covers with small values for c, ℓ, and k, as well as distance and port labeling schemes with small label sizes and query times, as all of these properties affect the performance of the routing scheme. While distance labeling schemes require large labels in general graphs [11], better results can be obtained by restricting

the graph-cover to only contain trees as subgraphs. Such a cover is then called *tree-cover*. Tree-covers are standard in routing [1, 2, 8, 9, 14, 26, 27], and while it is known that greedy routing with respect to $d_{\mathcal{C}}$ for a (c, ℓ, k)-tree-cover $\mathcal{C}$ is starvation-free (see e.g., [14]), we also know that the resulting routing scheme has stretch c with additive bound ℓ due to Lemma 9. Moreover, for trees there are distance and port labeling schemes with $\mathcal{O}(\log^2 n)$ bit labels and constant query time [10, 27]. Together with Lemma 11 we obtain the following theorem.

▶ **Theorem 12.** *Given a (c, ℓ, k)-tree-cover of a graph G, greedy routing on G can be implemented such that the routing scheme is starvation-free, has stretch c with additive bound ℓ, stores $\mathcal{O}(k(\log^2 n + \log k))$ bits at each vertex, and takes $\mathcal{O}(k)$ time for a routing decision.*

To complete our scheme, we propose an algorithm that computes a tree-cover with bounded stretch. It is an adaptation of an algorithm previously used to compute graph spanners [7]. We start with the following lemma, describing a situation that is exploited by our algorithm.

▶ **Lemma 13.** *Let $G = (V, E)$ be a graph, $u, v \in V$, and let H be an induced subgraph that contains all vertices on a shortest uv-path P in G. Let T be a partial shortest-path tree in H rooted at t that contains u and v. Then, for every vertex w in T that lies on P, $d_T(u, v) \leq d_G(u, v) + 2d_H(t, w)$.*

Consider the setting as in the above lemma, let w be chosen such that $d_H(t, w)$ is minimal and let $\xi = 2d_H(t, w)/d_G(u, v)$. Then, it holds that $d_T(u, v) \leq (1 + \xi)d_G(u, v)$. That is, T has stretch $(1 + \xi)$. The following algorithm computes a tree-cover with the same stretch.

Let G be the input graph. The algorithm operates in phases, starting with phase 0. For each phase i, we define a radius $r_i = b^i$, for a base $b > 1$. Then, for $a > 0$, we choose a vertex t in the current graph and compute the partial shortest-path tree with root t containing all vertices with distance at most $(1 + a)r_i$ from t. Afterwards, we delete all vertices with distance at most r_i to t from the current graph. This is iterated until all vertices are deleted. Afterwards, phase i is done and we restore the original input graph G before starting phase $i + 1$. This process is stopped, once the whole graph is deleted after processing the first tree in a phase. The output of the algorithm is the set of all trees computed during execution. Since the algorithm **pro**duces **t**ree-covers **o**f **n**etworks, we call it PROTON.

Note that PROTON has several degrees of freedom. We can choose the parameters $a > 0$ and $b > 1$, as well as the order in which the roots of the partial shortest-path trees are selected. The following lemma holds independent of the root selection strategy.

▶ **Lemma 14.** *The tree-cover computed by PROTON has stretch $(1 + 2b/a)$ with additive bound 2.*

Proof. Let $\mathcal{C}$ be the tree-cover computed by PROTON, let $G = (V, E)$ be the input graph, and let $u \neq v \in V$ be two arbitrary vertices. We have to show that $\mathcal{C}$ contains a tree T that includes u and v such that $d_T(u, v) \leq (1 + 2b/a)d_G(u, v)$ or $d_T(u, v) \leq d_G(u, v) + 2$.

Let i be minimal such that $d_G(u, v) \leq ar_i$. Assume for now that PROTON did not stop before phase i; we deal with the other case later. As phase i continues until all vertices are deleted, at one point a vertex w on a shortest uv-path in G is deleted. Let H be the current graph before that happens for the first time and let T be the partial shortest-path tree computed in H rooted at t. To show that T is the desired tree, we aim to apply Lemma 13.

First note that H is an induced subgraph of G that contains all vertices on a shortest uv-path of G. Moreover, T contains u and v for the following reason. As w is deleted, we have that $d_H(t, w) \leq r_i$. Moreover, as w lies on a shortest uv-path, the distance from w

to either u or v cannot exceed $d_H(u,v) = d_G(u,v)$. Thus, by the triangle inequality and the above choice of i, we have $d_H(t,u) \leq d_H(t,w) + d_H(w,u) \leq r_i + ar_i = (1+a)r_i$, which implies that u is a vertex of T. Analogously, v is also contained in T.

With this, we can apply Lemma 13, yielding stretch $(1+\xi)$ for $\xi = 2d_H(t,w)/d_G(u,v)$. To bound ξ, recall that we chose i minimal such that $d_G(u,v) \leq ar_i$. Thus, if $i > 0$, then $d_G(u,v) > ar_{i-1} = \frac{a}{b}r_i$. Together with the fact that $d_H(t,w) \leq r_i$, we obtain $\xi \leq 2\frac{b}{a}$, as desired. In the special case that $i = 0$ we have $r_i = 1$ and therefore $d_H(t,w) \leq 1$. Thus, Lemma 13 directly yields $d_T(u,v) \leq d_G(u,v) + 2$, which is covered by the additive bound 2.

Finally, we assumed above that PROTON did not stop before phase i and it remains to consider the case where it stops in phase $j < i$. In this case, let T be the tree we get in phase j, which includes all vertices of G. Let t be the root of T. As all vertices have distance at most r_j from t, we get $d_T(u,v) \leq 2r_j$. Moreover, as i was chosen minimal such that $d_G(u,v) \leq ar_i$, we have $d_G(u,v) > ar_j$. Together with the previous inequality, this gives a stretch of $2/a$, which is smaller than the desired $(1+2b/a)$, as $b > 1$. ◀

3.2 Performance on Strongly Hyperbolic Unit Disk Graphs

While PROTON computes a (c,ℓ,k)-tree-cover with bounded stretch, the value k, i.e., the maximum number of trees that a vertex is contained in, depends on the structure of the considered graph. In the following, we show that k is small on networks with an underlying hyperbolic geometry. In our analysis, we consider the *radially increasing* root selection strategy that selects the vertices in order of increasing distance to the origin of the hyperbolic plane, and prove the following theorem. There, $\operatorname{diam}(G)$ denotes the *diameter* of G, i.e., the length of the longest shortest path in G.

▶ **Theorem 15.** *Let G be a strongly hyperbolic unit disk graph with radius $R > 0$. Given the disk representation of G, $a > 0$, and $b > 1$, the PROTON algorithm with the radially increasing root selection strategy computes a (c,ℓ,k)-tree-cover of G with*

$$c = 1 + 2b/a,\ \ell = 2\ \text{, and } k = \pi e \left(\frac{1+a}{b-1}(b^2 \operatorname{diam}(G) - 1)R + 2\left(\log_b(\operatorname{diam}(G)) + 2\right)\right).$$

First note that the correctness of the claimed stretch immediately follows from Lemma 14. However, bounding k is more involved. To that end, we first compute an upper bound that holds for a given phase and afterwards sum over all phases.

Consider the roots of the partial shortest-path trees that contain a vertex v in a given phase, which we refer to as the *roots of* v. We partition the hyperbolic disk into radial bands and compute an upper bound on the number of roots of v in each band, see Figure 4 for an illustration. We then utilize two key ingredients. First, since v is contained in the partial shortest-path trees of its roots, the length of the path between v and a root is bounded, and so is the angular distance between them. Consequently, all roots in a band lie in a bounded angular interval (blue areas in Figure 4). Secondly, roots cannot be adjacent as they would otherwise delete each other, which means that the hyperbolic distance between them has to be sufficiently large. For roots in the same band, this can only be achieved if their angular distance is large. Consequently, each root in a band reserves a portion of the angular interval (red areas in Figure 4) that no other root can lie in, from which we can derive an upper bound on the number of roots that lie in the band.

The following lemma bounds the angular distance between a vertex u and another vertex u_k, assuming that there exists a path of length k between them that consists only of vertices whose radius is not smaller than the one of u. In particular, this applies to roots

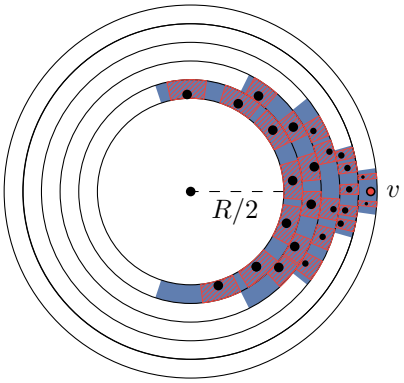


Figure 4 The hyperbolic disk is divided into radial bands. The roots (black vertices) of v (red dot) in a band lie in an angular interval Φ of bounded width (blue). Each root reserves a portion of that interval (red) that no other root can lie in. All vertices with radius at most $R/2$ are removed after processing the first root.

of v: In a given phase, the length of the paths considered in the partial shortest-path trees is bounded. Moreover, when the partial shortest-path tree of a root ρ of v is computed, all vertices of smaller radii than ρ have been deleted (since roots are considered in order of increasing radius), meaning the path from ρ to v cannot contain vertices of smaller radius.

▶ **Lemma 16.** *Let G be a strongly hyperbolic unit disk graph with radius R and let u be a vertex with $r(u) \geq R/2$. Further, let $P = (u, u_1, \dots, u_k)$ be a path with $r(u) \leq r(u_i)$ for all $i \in [k]$. Then, $\Delta_\varphi(u, u_k) \leq k \cdot \pi e^{R/2 - r(u)}$.*

Proof. For convenience, we define $u_0 = u$. Then, $\Delta_\varphi(u, u_k)$ can be bounded by

$$\Delta_\varphi(u, u_k) \leq \sum_{i=1}^{k} \Delta_\varphi(u_{i-1}, u_i).$$

Note that u_{i-1} and u_i are adjacent and recall that $\theta(r(u_{i-1}), r(u_i))$ denotes the maximum angular distance between them, such that this is the case. Thus,

$$\Delta_\varphi(u, u_k) \leq \sum_{i=1}^{k} \theta(r(u_{i-1}), r(u_i)).$$

Since $R/2 \leq r(u) \leq r(u_i)$ for all $i \in [k]$ is a precondition of this lemma, we have $r(u_{i-1}) + r(u_i) \geq R$ for all $i \in [k]$. Consequently, we can apply Lemma 4 to bound $\theta(r(u_{i-1}), r(u_i))$, which yields

$$\Delta_\varphi(u, u_k) \leq \sum_{i=1}^{k} \pi e^{(R - r(u_{i-1}) - r(u_i))/2} \leq \sum_{i=1}^{k} \pi e^{(R - r(u) - r(u))/2} = k \cdot \pi e^{R/2 - r(u)},$$

where the second inequality is valid since $r(u) \leq r(u_i)$ for all $i \in [k]$. ◀

The second key ingredient is a lower bound on the minimum angular distance between two non-adjacent vertices in a radial band of fixed width in the hyperbolic disk. We note that in order to obtain a bound that is easy to work with, we aim to utilize Corollary 5. However, this requires that R is not too small. For now, we assume that this requirement is met and afterwards resolve the constraint in the analysis of the algorithm.

▶ **Lemma 17.** *Let G be a strongly hyperbolic unit disk graph with radius $R \geq 1$ and let $r \geq R/2$ be a radius. Further, let u, v be non-adjacent vertices with $r(u), r(v) \in [r, r+\tau]$ for $\tau \in [0, R-1]$. Then, $\Delta_\varphi(u,v) \geq e^{R/2-(r+\tau)}$.*

Proof. Recall that $\theta(r(u), r(v))$ denotes the maximum angular distance such that u and v are adjacent. Since the two vertices are not adjacent in our case, we can derive that $\Delta_\varphi(u,v) > \theta(r(u), r(v))$. We now aim to apply Corollary 5 in order to obtain a lower bound on $\theta(r(u), r(v))$. To this end, we first validate that its preconditions are met. Since $r(u), r(v) \geq r \geq R/2$, we have $r(u) + r(v) \geq R$. Moreover, by assumption we know that $r(u), r(v) \in [r, r+\tau]$ for $\tau \in [0, R-1]$, which implies that $|r(u) - r(v)| \leq \tau \leq R-1$. Consequently, we can apply Corollary 5 to conclude that

$$\begin{aligned}\theta(r(u), r(v)) &\geq e^{(R-r(u)-r(v))/2} \\ &\geq e^{(R-2r-2\tau)/2} \\ &= e^{R/2-(r+\tau)},\end{aligned}$$

where the second inequality is valid, since by assumption $r(u), r(v) \leq r + \tau$. ◀

We can now combine the two key ingredients to compute an upper bound on the number of the roots of v in a given phase i, which we denote by $\boldsymbol{\rho}_i(v)$.

▶ **Lemma 18.** *Let G be a strongly hyperbolic unit disk graph with radius $R > 0$. Let the disk representation of G, $a > 0$, and $b > 1$ be given and consider phase i of the* Proton *algorithm. Then, for every vertex v it holds that $|\boldsymbol{\rho}_i(v)| \leq \pi e(R(1+a)b^i + 2)$.*

Proof. In the following, we aim to utilize Lemmas 16 and 17, both of which require that the considered vertices have a radius of at least $R/2$ and one additionally assumes that $R \geq 1$ Therefore, we first argue about the case where these conditions are not met. First note that after the first root in a phase is processed, all vertices with radius at most $R/2$ are removed since (if they exist in the first place) they form a clique. Additionally, when $R < 1$, the whole graph can be covered by few cliques. More precisely, by Lemma 8 a strongly hyperbolic unit disk graph with radius R can be covered by $\max\{2\pi\sqrt{2}, 2\pi e^{R/2}\}$ cliques. In particular, for $R < 1$, this yields a bound of $2\pi\sqrt{e}$. Since processing each root removes at least one such clique from the graph, the number of roots in the phase is bounded by the number of cliques. It follows, considering the first clique in $D_{R/2}$ and the remaining cliques when $R < 1$, that we can bound the roots of v in phase i as $|\boldsymbol{\rho}_i(v)| \leq 1 + 2\pi\sqrt{e} \leq 2\pi e$, which we account for with the $+2$ in the lemma statement.

For the remaining roots of v we can now assume that $R \geq 1$ and that all vertices have radius at least $R/2$. Furthermore, it suffices to show that there are at most $\pi e R(1+a)b^i$ such roots. We cover the remainder of the disk with $R/2$ bands of radial width 1, where the jth band (for $j \in \{0, \dots, R/2-1\}$) contains all points with radius in $[R/2+j, R/2+j+1]$, see Figure 4. The claim then follows if we can bound the number of roots of v in a single band by $2\pi e(1+a)b^i$.

Let $\boldsymbol{\rho}_{i,j}(v)$ denote the roots of v that lie in the jth band (see Figure 5). We first bound the angular distance between v and a root in $\boldsymbol{\rho}_{i,j}(v)$, and with that the width of the angular interval Φ (blue region in Figure 5) that contains all of them. Afterwards, we show that each root reserves a portion of Φ that no other root can be in. An upper bound on $|\boldsymbol{\rho}_{i,j}(v)|$ is then obtained by the quotient of the widths of the two intervals.

Consider a root $\rho \in \boldsymbol{\rho}_{i,j}(v)$. Since the roots are processed in order of increasing radius, all vertices of radius at most $r(\rho)$ have been removed before. Consequently, the path from ρ to v in the partial shortest-path tree rooted at ρ consists only of vertices with radius at

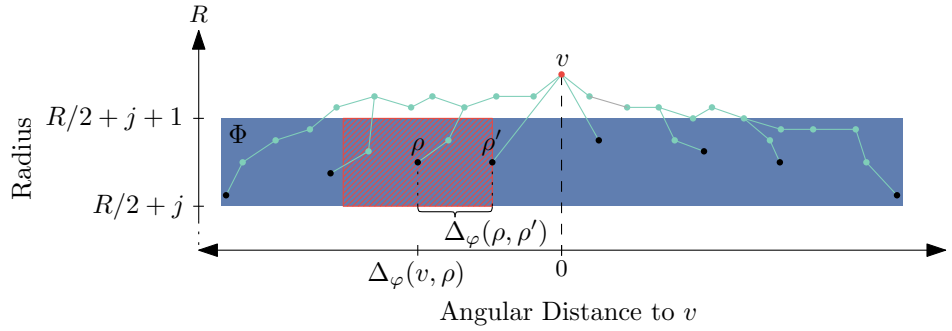


Figure 5 A vertex v (red dot) and the roots (black vertices) that are contained in the jth band and are connected to v (via the green paths). All roots lie in the angular interval Φ (blue region). Other than ρ, no root can lie in the red region.

least $r(\rho)$. Moreover, in phase i the depth of this tree is $(1+a)b^i$, which means that the path between ρ and v is at most this long. Therefore, we can apply Lemma 16 to conclude that the maximum angular distance between v and a root ρ is at most

$$\max_{\rho \in \boldsymbol{\rho}_{i,j}(v)} \Delta_\varphi(v,\rho) \leq \max_{\rho \in \boldsymbol{\rho}_{i,j}(v)} (1+a)b^i \cdot \pi e^{R/2-r(\rho)} \leq (1+a)b^i \cdot \pi e^{-j},$$

where the last inequality stems from the fact that $r(\rho) \geq R/2 + j$ holds for all $\rho \in \boldsymbol{\rho}_{i,j}(v)$. Moreover, since roots cannot be adjacent (as they would otherwise delete each other) and all roots in $\boldsymbol{\rho}_{i,j}(v)$ have their radii in $[R/2+j, R/2+j+1]$, we can apply Lemma 17 to conclude that the minimum angular distance between two roots $\rho, \rho' \in \boldsymbol{\rho}_{i,j}(v)$ is at least

$$\min_{\rho \neq \rho' \in \boldsymbol{\rho}_{i,j}(v)} \Delta_\varphi(\rho,\rho') \geq e^{R/2-(R/2+j+1)} = e^{-(j+1)}.$$

Note that the angular interval Φ extends to both angular directions from v. Therefore, the number of roots in $\boldsymbol{\rho}_{i,j}(v)$ can be bounded by

$$\begin{aligned} |\boldsymbol{\rho}_{i,j}(v)| &\leq 2 \cdot \frac{\max_{\rho \in \boldsymbol{\rho}_{i,j}(v)} \Delta_\varphi(v,\rho)}{\min_{\rho \neq \rho' \in \boldsymbol{\rho}_{i,j}(v)} \Delta_\varphi(\rho,\rho')} \\ &\leq \frac{2(1+a)b^i \cdot \pi e^{-j}}{e^{-(j+1)}} \\ &= 2\pi e(1+a)b^i. \end{aligned}$$

◀

With this we are now ready to bound the number k of trees that a vertex is contained in, for tree-covers produced by the PROTON algorithm.

Proof of Theorem 15. First note that the values for c and ℓ hold for any graph due to Lemma 14. It remains to show that the stated bound on k is valid. To that end, we make use of Lemma 18, which states that v is contained in at most $\pi e(R(1+a)b^i + 2)$ trees in phase i, and sum over all phases. Since the radius of the shortest-path trees that are removed from the graph in two consecutive phases increases by a factor of b and the algorithm terminates when the first tree in a phase deletes the whole graph, there are at most $\lceil \log_b(\mathrm{diam}(G)) \rceil$ phases. Thus,

$$\begin{aligned} k &= \sum_{i=0}^{\log_b(\mathrm{diam}(G))+1} \pi e(R(1+a)b^i + 2) \\ &= \pi e \left(R(1+a) \left(\sum_{i=0}^{\log_b(\mathrm{diam}(G))+1} b^i \right) + 2\left(\log_b(\mathrm{diam}(G)) + 2\right) \right). \end{aligned}$$

Note that the remaining sum is a partial sum of a geometric series with $b > 1$, which can be computed as $\sum_{i=0}^{x} b^i = (b^{x+1} - 1)/(b - 1)$. We obtain

$$\begin{aligned} k &= \pi e \left(R(1+a) \frac{b^{\log_b(\operatorname{diam}(G))+2} - 1}{b-1} + 2\left(\log_b(\operatorname{diam}(G)) + 2\right) \right) \\ &= \pi e \left(\frac{1+a}{b-1}(b^2 \operatorname{diam}(G) - 1)R + 2\left(\log_b(\operatorname{diam}(G)) + 2\right) \right). \end{aligned}$$ ◀

Since hyperbolic random graphs are a special case of strongly hyperbolic unit disk graphs, where vertices are distributed in a disk of radius $R = \mathcal{O}(\log n)$ and since these graphs have a diameter of $\mathcal{O}(\log n)$ asymptotically almost surely [20], we obtain the following corollary.

▶ **Corollary 19.** *Let G be a hyperbolic random graph. Given the disk representation of G, $a > 0$, and $b > 1$, the* Proton *algorithm with the radially increasing root selection strategy computes a (c, ℓ, k)-tree-cover of G with $c = 1 + 2b/a$, $\ell = 2$, and, asymptotically almost surely*

$$k = \mathcal{O}\left(\frac{(1+a)b^2}{b-1} \cdot \log^2 n \right).$$

Together with Theorem 12, it follows that greedy routing on hyperbolic random graphs can be implemented such that the resulting scheme is starvation-free and has stretch $1 + 2b/a$ with additive bound 2. Moreover, by setting $a = b = 2$ we obtain a multiplicative stretch of 3, and can derive that the scheme, asymptotically almost surely, stores $\mathcal{O}(\log^4 n)$ bits at each vertex and takes $\mathcal{O}(\log^2 n)$ time per query, which improves upon the performance lower bound for general graphs.

4 Experiments

We designed a routing scheme that utilizes hierarchical structures and showed that it has small stretch, space requirements, and query times on strongly hyperbolic unit disk graphs. To evaluate how well our results translate to real-world networks, we performed experiments on 50 graphs from the Network Data Repository [25], with sizes ranging from 14 k to over 2.3 M vertices. Since we do not have unit disk representations for these, we used the degrees of the vertices as a proxy for their place in the hierarchical structure. That is, the root selection strategy processed the vertices by decreasing degree. For each graph, we computed a tree-cover using the Proton algorithm with parameters $a = b = 2$, and sampled 10 k vertex pairs for which the path obtained by our routing scheme was compared to a shortest path between them. Figure 6 shows boxplots aggregating our observations.

As expected, the maximum observed stretch is 3. However, this stretch occurred only rarely. In all networks most of the sampled routes had a stretch of at most 1.5 and in 16 of the 50 graphs the median stretch was 1. At the same time, the number of trees that a vertex was contained in on average remained small. In 42 of the 50 networks this number was less than 50, even in networks with over 2.3 M vertices.

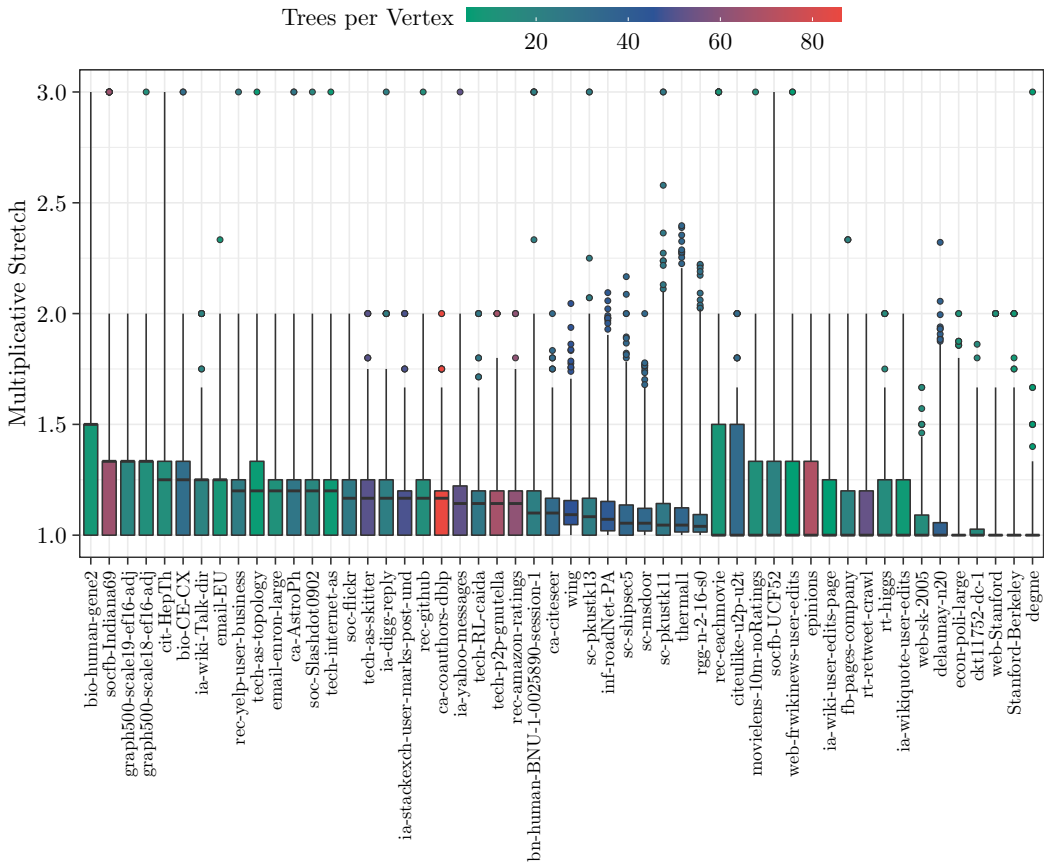


Figure 6 Multiplicative stretch when routing with a tree-cover obtained using PROTON with $a = b = 2$. Colors show the number of trees that an average vertex is contained in. Boxes denote the interquartile range extending to the 25th and 75th percentile with horizontal bars showing the median. Whiskers extend to 0.1% and 99.9%, while dots show values beyond that.

References

1 Baruch Awerbuch, Shay Kutten, and David Peleg. On buffer-economical store-and-forward deadlock prevention. *IEEE Transactions on Communications*, 42(11):2934–2937, 1994. doi:10.1109/26.328973.

2 Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM J. Discrete Math.*, 5:151–162, May 1992. doi:10.1137/0405013.

3 Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, and Daniel Stephan. Strongly hyperbolic unit disk graphs. *CoRR*, abs/2107.05518, 2021. arXiv:2107.05518.

4 Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1(1):62, 2010. doi:10.1038/ncomms1063.

5 Leizhen Cai. NP-completeness of minimum spanner problems. *Discrete Applied Mathematics*, 48(2):187–194, 1994. doi:10.1016/0166-218X(94)90073-6.

6 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990. doi:10.1016/0012-365X(90)90358-O.

7 Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28(1):210–236, 1998. doi:10.1137/S0097539794261295.

8 Tamar Eilam, Cyril Gavoille, and David Peleg. Compact routing schemes with low stretch factor. *Journal of Algorithms*, 46(2):97–114, 2003. doi:10.1016/S0196-6774(03)00002-6.

9 R. Flury, S. V. Pemmaraju, and R. Wattenhofer. Greedy routing with bounded stretch. In *IEEE INFOCOM 2009*, pages 1737–1745, 2009. doi:10.1109/INFCOM.2009.5062093.

10 Ofer Freedman, Paweł Gawrychowski, Patrick K. Nicholson, and Oren Weimann. Optimal distance labeling schemes for trees. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 185–194, 2017. doi:10.1145/3087801.3087804.

11 Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85–112, 2004. doi:10.1016/j.jalgor.2004.05.002.

12 Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 125–133, 1996. doi:10.1145/248052.248075.

13 Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: degree sequence and clustering. In *International Colloquium on Automata, Languages, and Programming*, pages 573–585. Springer, 2012.

14 R. Houthooft, S. Sahhaf, W. Tavernier, F. De Turck, D. Colle, and M. Pickavet. Fault-tolerant greedy forest routing for complex networks. In *2014 6th International Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 1–8, 2014. doi:10.1109/RNDM.2014.7014924.

15 M.L. Huson and A. Sen. Broadcast scheduling algorithms for radio networks. In *Proceedings of MILCOM '95*, volume 2, pages 647–651 vol.2, 1995. doi:10.1109/MILCOM.1995.483546.

16 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. *Algorithmica*, 80(3):830–848, 2018. doi:10.1007/s00453-017-0308-2.

17 Sándor Kisfaludi-Bak. Hyperbolic intersection graphs and (quasi)-polynomial time. In *Symposium on Discrete Algorithms (SODA)*, pages 1621–1638, 2020. doi:10.1137/1.9781611975994.100.

18 Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, 2010. doi:10.1103/PhysRevE.82.036106.

19 Anton Krohmer. *Structures & Algorithms in Hyperbolic Random Graphs*. Doctoral thesis, Universität Potsdam, 2016.

20 Tobias Müller and Merlijn Staps. The diameter of kpkvb random graphs. *Advances in Applied Probability*, 51(2):358–377, 2019. doi:10.1017/apr.2019.23.

21 F. Papadopoulos, D. Krioukov, M. Boguna, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010. doi:10.1109/INFCOM.2010.5462131.

22 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. doi:10.1002/jgt.3190130114.

23 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989. doi:10.1145/65950.65953.

24 Vijay Raghavan and Jeremy Spinrad. Robust algorithms for restricted domains. *Journal of Algorithms*, 48(1):160–172, 2003. doi:10.1016/S0196-6774(03)00048-8.

25 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL: http://networkrepository.com.

26 M. Tang, H. Chen, G. Zhang, and J. Yang. Tree cover based geographic routing with guaranteed delivery. In *2010 IEEE International Conference on Communications*, pages 1–5, 2010. doi:10.1109/ICC.2010.5502391.

27 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2001. doi:10.1145/378580.378581.

28 Huaming Zhang and Swetha Govindaiah. Greedy routing via embedding graphs onto semi-metric spaces. *Theoretical Computer Science*, 508:26–34, 2013. doi:10.1016/j.tcs.2012.01.049.

Tight Bounds for Connectivity Problems Parameterized by Cutwidth

Narek Bojikian
Humboldt-Universität zu Berlin, Germany

Vera Chekan
Humboldt-Universität zu Berlin, Germany

Falko Hegerfeld
Humboldt-Universität zu Berlin, Germany

Stefan Kratsch
Humboldt-Universität zu Berlin, Germany

Abstract

In this work we start the investigation of tight complexity bounds for connectivity problems parameterized by cutwidth assuming the Strong Exponential-Time Hypothesis (SETH). Van Geffen et al. [21] posed this question for Odd Cycle Transversal and Feedback Vertex Set. We answer it for these two and four further problems, namely Connected Vertex Cover, Connected Dominating Set, Steiner Tree, and Connected Odd Cycle Transversal. For the latter two problems it sufficed to prove lower bounds that match the running time inherited from parameterization by treewidth; for the others we provide faster algorithms than relative to treewidth and prove matching lower bounds. For upper bounds we first extend the idea of Groenland et al. [8] to solve what we call *coloring-like problems*. Such problems are defined by a symmetric matrix M over $\mathbb{F}_2$ indexed by a set of colors. The goal is to count the number (modulo some prime p) of colorings of a graph such that M has a 1-entry if indexed by the colors of the end-points of any edge. We show that this problem can be solved faster if M has small rank over $\mathbb{F}_p$. We apply this result to get our upper bounds for CVC and CDS. The upper bounds for OCT and FVS use a subdivision trick to get below the bounds that matrix rank would yield.

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

Keywords and phrases Parameterized complexity, connectivity problems, cutwidth

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.14

Related Version *Full Version*: https://arxiv.org/abs/2212.12385 [3]

Funding *Vera Chekan*: Supported by the DFG Research Training Group 2434 "Facets of Complexity".

1 Introduction

Parameterized complexity studies the complexity of (typically NP-hard) computational problems in a finer way where aside from the input size n, other values, called parameters and frequently denoted by k, are considered in the running time. This might be for example the solution size or some structural properties of input. The class of fixed-parameter tractable problems (FPT) contains problems that admit an algorithm with running time $\mathcal{O}(f(k)n^c)$ for a computable function f and a constant c. Since f is only required to be computable, it might grow very rapidly, e.g., $f(k) = 100^k$ or k^k, or even the power tower function are allowed by this definition. Therefore, from a practical point of view, a problem being FPT does not say much about the solvability of the problem in reasonable running time.


© Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch;
licensed under Creative Commons License CC-BY 4.0

This inspired the search for better functions f. Since the problems we deal with are already NP-hard, stronger hardness conjectures than $P \neq NP$ have been assumed to show that some function f is essentially optimal for a problem and a specific parameter. For example, assuming the Exponential Time Hypothesis (ETH) [10], it has been shown that many problems do not admit algorithms with single-exponential running time [7, 16], i.e., a running time of the form $\mathcal{O}^*(c^k)$ for some constant c. Since some problems in FPT are known to admit an algorithm with single-exponential running time, an even stronger conjecture known as the Strong Exponential Time Hypothesis (SETH) was stated and used to prove that some base c is optimal for such a problem under this conjecture. This conjecture claims, roughly speaking, that SAT cannot be solved much better than brute-forcing. Such lower bounds were easier to prove for structural parameters on graphs, where the value of the parameter reflects how well-structured or interconnected a graph is. Upper bounds for such parameters usually rely on dynamic programming employing some tricks and advanced techniques like fast subset convolution [1], rank-based methods [2, 6, 5], the isolation lemma [19] etc. For many classical problems parameterized by treewidth, it has been shown that the optimal running time under SETH is single-exponential and the base of the exponent is known, e.g., [15]. There is a special class of problems related to this question. These are the *connectivity problems*. Even though the class is not well-defined, all problems in this class impose connectivity constraints on the structure of a solution. In these problems we usually look for a set of vertices (or edges) that it is either connected itself and has some further properties (e.g., Connected Vertex Cover or Connected Dominating Set), or such that the input graph satisfies a certain disconnectivity requirement after its removal (e.g., Feedback Vertex Set). For a long time, the existence of single-exponential algorithms for connectivity problems parameterized by treewidth or other structural parameters remained open. But then a breakthrough work of Cygan et al. introduced a new view on such problems called *Cut&Count* [7]. This technique is randomized and it reduces connectivity problems to counting certain bipartitioned solutions modulo two.

Tight bounds for problems parameterized by treewidth and pathwidth have been widely studied (e.g., [7, 6, 5, 15]). An optimal dynamic programming algorithm traverses a tree or a path decomposition in a bottom-up manner and utilizes the fact that every bag of the decomposition is a small vertex separator. Therefore, it is also natural to study parameters that are based on edge separators. Imagine that the vertices of the graph are put on the line in some fixed order and the edges are drawn as x-monotone curves. The cutwidth of this arrangement is then the maximal number of edges crossing any vertical line. The cutwidth of the graph is then the smallest cutwidth of such an arrangement. Note that for any vertical line, the set of edges crossing it separates vertices lying on different sides of this line from each other. Therefore, cutwidth is an analogue of pathwidth based on edge separators. In fact, pathwidth can be defined in an analogous way to cutwidth, that is, we count the number of vertices on one side of the cut that have neighbors on the other side of the cut [13]. This also shows that pathwidth is upper-bounded by cutwidth.

Since cutwidth is an upper bound for pathwidth, an algorithm running in time $\mathcal{O}^*\big(f(\mathrm{pw})\big)$ also runs in time $\mathcal{O}^*\big(f(\mathrm{ctw})\big)$. In particular, single-exponential solvability transfers from pathwidth to cutwidth. However, it is possible that the optimal dependence on cutwidth is smaller than for pathwidth. Tight bounds for problems parameterized by cutwidth have already been determined for example for Coloring [12], #q-Coloring [8], Independent Set and Dominating Set [21], and for various factor problems [18]. However, for connectivity problems, this question remained open. Van Geffen et al. asked for the complexity of Odd Cycle Transversal, Feedback Vertex Set, and Hamiltonian Cycle [21]. We start this investigation in our work. Generally, tight bounds have also been studied for other decompositional parameters as well: treedepth (e.g., [9]), clique-width (e.g., [14]) etc.

Table 1 Tight bounds for parameterizations by treewidth and cutwidth.

Connected Vertex Cover (CVC)	3^{tw}	2^{ctw}
Connected Dominating Set (CDS)	4^{tw}	3^{ctw}
Odd Cycle Transversal (OCT)	3^{tw}	2^{ctw}
Feedback Vertex Set (FVS)	3^{tw}	2^{ctw}
Steiner Tree (ST)	3^{tw}	3^{ctw}
Connected Odd Cycle Transversal (COCT)	4^{tw}	4^{ctw}

Our contribution. Van Geffen et al. [21] asked for the exact complexity of Odd Cycle Transversal (OCT) and Feedback Vertex Set (FVS) parameterized by cutwidth under SETH. In this work, we answer this question for these two problems in addition to four other connectivity problems. For two of the problems, we show the optimal base of exponent is the same for the parameterizations by treewidth and cutwidth. For the remaining problems, the base is smaller for cutwidth. The right column of Table 1 contains the tight bounds for these problems parameterized by cutwidth and summarizes the results of our work. The middle column contains the analogous results for treewidth for comparison [7, 15].

Organization. We begin this work with a brief summary of the used notation. In Section 3 we define coloring-like problems and provide a general framework to solve these problems efficiently. In Section 4, we present the trick used to solve both Odd Cycle Transversal and Feedback Vertex Set. We follow by a brief summary of our lower-bound construction on the example of Steiner Tree. We conclude in Section 6 by providing possible directions of further research in this area. Technical details and the remaining results can be found in the full version of the paper [3].

2 Preliminaries

For $n \in \mathbb{N}_0$, let $[n] = \{1, 2, \ldots, n\}$ and $[n]_0 = [n] \cup \{0\}$. For a function $f\colon X \to Y$ and a set $S \subseteq X$, with $f_{|S}$ we denote the function $f_{|S} : S \to Y$ such that $f_{|S}(x) = f(x)$ for all $x \in S$. If $Y \not\subseteq \mathbb{Z}$ we define $f(S) = \{y \in Y \mid \exists x \in S\colon f(x) = y\}$. With $f^{-1}(y)$ we denote the set $\{x \in X \mid f(x) = y\}$. We abuse the notation for injective functions and consider the singleton $\{x\}$ as an element $x \in X$. For a function $f\colon X \to Y$ with $Y \subseteq \mathbb{Z}$ and a subset $S \subseteq X$, with $f(S)$ we denote the value $f(S) = \sum_{s \in S} f(s)$. Further, let $f\colon X \times \mathbb{Z}^2 \to Y$ be a function, sometimes we call it a *table*. We call X the *domain* of f and denote it with $\mathrm{dom}(f)$. If $0 \in Y$, the *support* of f is the set $\mathrm{supp}(f) = \left\{x \in X \;\middle|\; \exists K, W \in \mathbb{Z}\colon f(x, K, W) \neq 0\right\}$.

Apart from $[n]$, we will use several other interpretations of square brackets. First, Iverson's bracket notation: for a predicate p, the value $[p]$ is equal to 1 if p is true and 0 otherwise. Sometimes, for space reasons, we will also write 1_p instead of $[p]$. Second, for a function $f\colon X \to Y$, element x^*, and element $y \in Y$, with $f[x^* \mapsto y]$ we denote the function $f[x^* \mapsto y]\colon X \cup \{x^*\} \to Y$, where $f[x^* \mapsto y](x^*) = y$ and $f[x^* \mapsto y](x) = f(x)$ for $x \neq x^*$. Note that both $x^* \in X$ and $x^* \notin X$ are allowed by this definition. Finally, for two elements x, y, with $[x \mapsto y]$ we denote the unique function in $\{y\}^{\{x\}}$.

In this work, we will only consider simple loopless undirected graphs. A *linear arrangement* ℓ of a graph $G = (V, E)$ is a bijection $\ell\colon [n] \to V$. For $i \in [n]$, let $v_i = \ell(i)$ and with E_i we refer to the set of edges $E_i = \left\{\{v_j, v_k\} \in E \;\middle|\; j \leq i, k > i\right\}$ called the i*th cut* (also called the *cut at* v_i or *between* v_i and v_{i+1}). If an edge e belongs to E_i, we also say that it *crosses* the ith cut. We say that two edges $e \neq e'$ *overlap* on ℓ, if there exists $i \in [n]$ such

that $e, e' \in E_i$. The *cutwidth* $\mathrm{ctw}(\ell)$ of ℓ is defined as $\mathrm{ctw}(\ell) = \max_{i \in [n]} |E_i|$. The *cutwidth* $\mathrm{ctw}(G)$ of a graph G is then the smallest cutwidth over all linear arrangements of G. Let ℓ now denote a fixed linear arrangement of G. For $i \in [n]$, we use the following notation:

- $V_i = \{v_1, v_2, \ldots, v_i\}$, $G_i = G[V_i]$,
- $X_i = \left\{v_j \;\middle|\; j \leq i, \exists k > i \colon \{v_j, v_k\} \in E_i\right\} \cup \{v_i\}$,
- $Y_i = \left\{v_j \;\middle|\; j > i, \exists k \leq i \colon \{v_j, v_k\} \in E_i\right\}$,
- $H_i = (X_i \cup Y_i, E_i)$ the *cut-graph at* v_i (also called the i*th cut-graph*; note that it is bipartite with the *left side* X_i and the *right side* Y_i),
- and if $i \geq 2$, then $Z_i = X_{i-1} \cup \{v_i\}$.

Observe that every vertex $v \neq v_i \in X_i$ has an incident edge in the ith cut whose other endvertex belongs to Y_i. Therefore, the size of X_i is at most $\mathrm{ctw} + 1$. Further, for $i \in [n-1]$, we have $X_{i+1} \subseteq X_i \cup \{v_{i+1}\}$ (for any vertex $v \neq v_{i+1} \in X_{i+1}$, an incident edge crossing the $(i+1)$th cut, also crosses the ith cut). Hence, $X_{i+1} \cap X_i = X_{i+1} \setminus \{v_{i+1}\}$, and $X_{i+1} \setminus X_i = \{v_{i+1}\}$. To prove the tightness of our bounds, we will sometimes rely on results for problems parameterized by treewidth or pathwidth.

A path decomposition of a graph G is a sequence $\mathcal{B} = B_1, \ldots, B_r$ of the so-called *bags* such that:

- It holds that $B_1 \cup \cdots \cup B_r = V$.
- For each $\{u, v\} \in E$, there exists an index $i \in [r]$ such that $u, v \in B_i$.
- And for every $i < j \in [r]$, the property $v \in B_i \cap B_j$ implies $v \in B_k$ for every $i \leq k \leq j$.

The *pathwidth* of $\mathcal{B}$ is defined as $\mathrm{pw} = \max_{i \in [r]} |B_i| - 1$. The pathwidth of G is the smallest $\mathrm{pw}(\mathcal{B})$ over all path decompositions $\mathcal{B}$ of G. A path decomposition with certain useful properties is called *nice*. In the full version of the paper [3], we employ these properties to develop a dynamic programming algorithm and we also provide a formal definition there. We skip the definition of treewidth since we do not work with it explicitly. We will mainly use the following result:

▶ **Lemma 1** ([13]). *The cutwidth of a graph is an upper bound for its tree- and pathwidth.*

All lower bounds in this work assume the Strong Exponential Time Hypothesis (SETH). We use the following equivalent formulation of SETH:

▶ **Conjecture 2** ([10, 11]). *For any positive value δ there exists an integer d such that d-SAT cannot be solved in time $\mathcal{O}^*\left((2-\delta)^n\right)$ where n denotes the number of variables.*

As mentioned in the introduction, there is a class of the so-called connectivity problems. In their seminal paper Cygan et al. provided a breakthrough approach called *Cut&Count* to solve connectivity problems in single-exponential time based on the so-called consistent cuts [7]. Let us sketch the idea on an example of Connected Vertex Cover. A *vertex cover* of a graph is a set of vertices such that every edge of the graph is incident to some vertex in this set. So the Connected Vertex Cover is defined as follows.

Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a vertex cover $S \subseteq V$ of cardinality at most k such that $G[S]$ is connected?

Let $v \in V$ be a fixed vertex contained in some fixed vertex cover S. A *consistent cut* of S is a partition $L \cup R = S$ such that $v \in L$ and there is no edge between L and R in G. Let $\mathrm{cc}(S)$ denote the number of connected components of $G[S]$. By definition, every component is completely contained either in L or in R and the component containing v is in L. Therefore, the number of consistent cuts of S is $2^{\mathrm{cc}(S)-1}$. Crucially, this number is odd if and only if $G[S]$

is connected. So if we could assume that the solution is unique, then counting the number of consistent cuts modulo 2 would suffice to solve the problem. However, a graph might contain several solutions so that the corresponding cuts cancel out. To overcome this issue Cygan et al. assign weights to vertices and employ the Isolation Lemma of Mulmuley et al. [19] to ensure that with high probability the minimum-weight solution is unique. The following theorem summarizes this result in a suitable for us form applied to the parameterization by cutwidth:

▶ **Theorem 3** ([7]). *Let* $G = (V, E)$ *be a graph,* ℓ *a linear arrangement of* G *of cutwidth* ctw, $v \in V$ *a fixed vertex,* $\omega\colon V \to [2|V|]$ *a weight function, and* $K \in [n]_0, W \in [2|V|^2]_0$ *integers. With* $\mathcal{C}^K_W$ *(resp.* $\mathcal{D}^K_W$*) we denote the family of pairs* $(S, (L, R))$ *such that* S *is a vertex cover (resp. dominating set) of* G, $|S| = K$, $\omega(S) = W$, $v \in L$, *and* (L, R) *is a consistent cut of* S. *If there exists an algorithm* $\mathcal{A}$ *that given the above input computes the size of* $\mathcal{C}^K_W$ *(resp.* $\mathcal{D}^K_W$*) modulo 2 in time* $\mathcal{O}^*(\beta(\text{ctw}))$ *for some computable function* β, *then there exists a randomized algorithm that given a graph* G *and its linear arrangement of cutwidth* ctw *solves the CVC (resp. CDS) problem in time* $\mathcal{O}^*(\beta(\text{ctw}))$. *The algorithm cannot give false positives and may give false negatives with probability at most* $1/2$.

3 Coloring-like Problems

In this section, we generalize the approach of Groenland et al. [8] counting the number of proper q-list-colorings modulo some prime number p to what we call coloring-like problems. We will mostly use their definitions, and slightly adapt them for our purposes. Since many ideas and proofs are very similar to their approach, we only provide the sketch of an algorithm and emphasize key observations and new results here. The long version with the full description of the algorithm and the whole proof of its correctness can be found in the full version of the paper [3].

A *coloring-like problem* P is defined by a finite set $C = \{1, \ldots, |C|\}$ of *colors*, a set $Q \subseteq C$ of *special* colors, and a symmetric *consistency matrix* $M \in \{0,1\}^{|C|\times|C|}$. Additionally, the problem might contain a prime number p. An instance of P consists of a graph $G = (V, E)$ with a linear arrangement $\ell\colon [|V|] \to V$ of cutwidth ctw, a list function $a\colon V \to 2^C$ of allowed colors, numbers $N, K^*, W^* \in \mathbb{N}_0$, and a weight function $\omega\colon V \to [N]$. The goal is to compute the following value:

$$\left|\left\{c \in C^V \;\middle|\; |c^{-1}(Q)| = K^*, \omega(c^{-1}(Q)) = W^*, \right.\right.$$
$$\left.\left. \forall v \in V\colon c(v) \in a(v), \forall \{u,v\} \in E\colon M[c(u), c(v)] = 1 \right\}\right|.$$

This is the number of proper list-colorings (with respect to a) of G such that the end-vertices of every edge are colored with consistent colors (with respect to M), exactly K^* vertices are colored with special colors (i.e., the coloring has *order* K^*), and these vertices have total weight of W^* in ω (i.e., the coloring has *weight* W^*). If a prime p is given, the goal is to compute this value modulo p.

Observe that for $q = |C|$, $Q = \emptyset$, and the consistency matrix M defined by $M[i,j] = 1$ if $i \neq j$ and $M[i,j] = 0$ if $i = j$ for all $i, j \in [q]$, the coloring-like problem P is exactly the problem of counting the number of proper list q-colorings of a graph (modulo p). Groenland et al. showed that for a prime number p, this matrix has (full) rank of q over $\mathbb{F}_p$ if p does not divide $q-1$ and rank of $q-1$ otherwise. Based on this property, they developed an $\mathcal{O}^*((q-1)^{\text{ctw}})$ (if p divides $q-1$) resp. $\mathcal{O}^*(q^{\text{ctw}})$ (otherwise) algorithm to solve the

problem. We generalize their idea to our notion of a coloring-like problem (i.e., defined by an arbitrary consistency matrix) and apply it to Connected Vertex Cover and (with additional tricks) to Connected Dominating Set.

A *coloring* of a vertex set $X \subseteq V$ is an assignment $x \in C^X$. A coloring x is *valid* if for every vertex $v \in X$, it holds that $c(v) \in a(v)$ and for every edge $\{v, w\}$ of $G[X]$, it holds that $M[x(v), x(w)] = 1$. For sets $X \subseteq Y \subseteq V$ and colorings $x \in C^X$, $y \in C^Y$, we say that y *extends* x if $y_{|X} = x$. For sets $X, Y \subseteq V$ and colorings $x \in C^X$, $y \in C^Y$ we say that x is *compatible* with y and write $x \sim y$ if $x_{|X\cap Y} = y_{|X\cap Y}$ and for every edge $\{u, v\} \in E$ with $u \in X, v \in Y \setminus X$ it holds that $M[x(u), y(v)] = 1$. We emphasize that compatibility is not symmetrical in general. The definition of compatibility is indeed not completely intuitive but it is motivated by the technical details of the correctness proof of our algorithm (see [3]): we allow x and y to be compatible even if x has conflicts along the edges of $G[X]$.

For $i \in [n]$, $c \in C^{X_i}$, and $K, W \in \mathbb{Z}$, let the value $T_i[c, K, W]$ be defined as

$$T_i[c, K, W] = \Big|\Big\{\phi \in C^{V_i} \;\Big|\; \phi_{|X_i} = c, \big|\phi^{-1}(Q)\big| = K, \omega\big(\phi^{-1}(Q)\big) = W, \\ \forall v \in V_i \colon \phi(v) \in a(v), \forall \{u, v\} \in E\big(G[V_i]\big) \colon M\big[\phi(u), \phi(v)\big] = 1\Big\}\Big|.$$

This is the number of possibilities to extend a coloring c of X_i to a valid coloring of V_i of order K and weight W. Recall that $X_n = \{v_n\}$ and $V_n = V$. Therefore, the number of list colorings that we are looking for as a final result is given by $\sum_{s\in C} T_n\big[[v_n \mapsto s], K^*, W^*\big]$.

The following lemma generalizes a similar result of Groenland et al. (a proof is provided in the full version of the paper [3]).

▶ **Lemma 4.** *For $2 \le i \le n$, $c \in C^{X_i}$, and $K, W \in \mathbb{Z}$, the following holds*

$$T_i[c, K, W] = \big[c(v_i) \in a(v_i)\big] \sum_{\substack{z\in C^{X_{i-1}} \\ z\sim c}} T_{i-1}\big[z, K - [c(v_i) \in Q], W - \omega(v_i)\cdot[c(v_i) \in Q]\big].$$

Recall that we have $X_i \subseteq X_{i-1} \cup \{v_i\}$. So the intuition behind this lemma is that given a valid coloring of V_{i-1}, we can extend it to a valid coloring of V_i by assigning v_i some color from $a(v_i)$ if and only if no conflicts on the edges incident to v_i are created this way. Such edges cross the $(i-1)$st cut so their end-vertices other than v_i belong to X_{i-1} and it suffices to consider the restriction of the coloring to X_{i-1}.

This lemma provides the recurrence for the dynamic programming solving the problem P. With a small trick also developed by Groenland et al., it is possible to compute T_i by iterating through the support of T_{i-1} only once. This gives a rise to an $\mathcal{O}^*\big(|C|^{\mathrm{ctw}} N\big)$ algorithm. Next we present how it can be accelerated if M has small rank over a certain field. The main idea will be to transform every table T_i (for $i \in [n]$) in such a way that the arising table has a smaller support and it still contains a sufficient amount of information to solve the problem.

Let $\mathbb{F}$ denote the field $\mathbb{F} = \mathbb{F}_p$ if p is a part of the input and the field $\mathbb{F} = \mathbb{Q}$ otherwise. Let "$\equiv$" denote equality over $\mathbb{F}$. Let $X, Y \subseteq V$ be disjoint sets and let $\widehat{f}, f \colon C^X \times \mathbb{Z}^2 \to \mathbb{F}$ be tables. We say that $\widehat{f}$ (X, Y)-*represents* f if for every coloring $y \in C^Y$ and all $K, W \in \mathbb{Z}$, it holds that $\sum_{x\in C^X, x\sim y} \widehat{f}(x, K, W) \equiv \sum_{x\in C^X, x\sim y} f(x, K, W)$. Note that (X, Y)-representation is an equivalence relation, in particular, it is transitive. For $i \in [n]$ and tables $f, \widehat{f} \colon C^{X_i} \times \mathbb{Z}^2 \to \mathbb{F}$, we say that $\widehat{f}$ i-*represents* f if $\widehat{f}$ (X_i, Y_i)-represents f.

Let $\mathrm{rank}(M)$ denote the rank of M over $\mathbb{F}$. We may assume that the first $\mathrm{rank}(M)$ rows of M form a row basis (otherwise we may permute the colors in C and consider the corresponding consistency matrix). We call the colors $\big\{\mathrm{rank}(M) + 1, \ldots, |C|\big\}$ *reduced*. For

a set $X \subseteq V$ and a table $f\colon C^X \times \mathbb{Z}^2 \to \mathbb{F}$ we say that a vertex $v \in X$ is *reduced* (in f) if for every coloring $x \in C^X$ with $x(v) > \operatorname{rank}(M)$ (i.e., v has a reduced color in x), it holds that $f(x, K, W) \equiv 0$ for all $K, W \in \mathbb{Z}$. The following lemma generalizes a simpler version of Groenland et al. [8]. Since this result is new and non-trivial, we provide the construction of a reduced representative here while the proof of correctness can be found in the full version of the paper [3].

▶ **Lemma 5.** *Let $X, Y \subseteq V$ be disjoint, let $i \in [n]$, and let $f\colon C^X \times \mathbb{Z}^2 \to \mathbb{F}$ be a table such that $f^{-1}(\mathbb{F}\setminus\{0\}) \subseteq C^X \times [i]_0 \times [iN]_0$ holds. Let $R \subseteq X$ be a set of reduced vertices in f. Finally, let $v \in X \setminus R$ be a vertex that has exactly one neighbor in Y. Then there is an algorithm **Reduce** that, given this information as input, in time $\mathcal{O}^*\big(\operatorname{rank}(M)^{|R|}|C|^{|X|-|R|}N\big)$ outputs a function $\widehat{f}\colon C^X \times \mathbb{Z}^2 \to \mathbb{F}$ (X, Y)-representing f with reduced vertices $R \cup \{v\}$.*

Proof. For $b \in \big[|C|\big]$, let m_b denote the bth row of M and let $k = \operatorname{rank}(M)$. Since the set $\{m_1, \ldots, m_k\}$ forms a row basis of M, for every $b \in \big\{k+1, \ldots, |C|\big\}$ and $j \in [k]$, there exists $d_{b,j} \in \mathbb{F}$ such that $m_b \equiv \sum_{j=1}^{k} d_{b,j} m_j$. We define the values of $\widehat{f}$ as

$$\widehat{f}(x, K, W) \equiv \begin{cases} 0 & \text{if } x(v) \geq k+1, \\ f(x, K, W) + \sum\limits_{b=k+1}^{|C|} d_{b,x(v)} f\big(x[v \mapsto b], K, W\big) & \text{otherwise,} \end{cases}$$

for all $x \in C^X$, $K, W \in \mathbb{Z}$. It is not hard to see that $R \cup \{v\}$ is a set of reduced vertices of $\widehat{f}$ and that the non-zero entries of $\widehat{f}$ can be computed within the claimed running time (see [3] for more details).

It remains to prove that $\widehat{f}$ (X, Y)-represents f. So by the definition of $\widehat{f}$, we need to show that for all $y \in C^Y$ and $K, W \in \mathbb{Z}$

$$\sum_{x \in C^X, x \sim y} f(x, K, W) \equiv \sum_{z \in C^X, z \sim y, 1 \leq z(v) \leq k} \left(f(z, K, W) + \sum_{b=k+1}^{|C|} d_{b,z(v)} f\big(z[v \mapsto b], K, W\big) \right)$$

holds. In [3], we show that for every coloring $x \in C^X$ the summand $f(x, K, W)$ occurs one the left side as often (over $\mathbb{F}$) as on the right side thus proving the equality. For this purpose, we carry out a case distinction based on whether $x \sim y$ holds or not. ◀

For $i \in [n]$, a table $f\colon C^{X_i} \times \mathbb{Z}^2 \to \mathbb{F}$ is *fully reduced* if for every $v \in X_i$ that has exactly one neighbor in Y_i, the vertex v is reduced. The analogue of the following lemma has already been proven by Groenland et al., see [3] for the proof of our version.

▶ **Lemma 6.** *Let $2 \leq i \leq n$ and let $\widehat{T_{i-1}}$ be a fully reduced table that $(i-1)$-represents T_{i-1}. Given $\widehat{T_{i-1}}$ and a set R_{i-1} of reduced vertices for $\widehat{T_{i-1}}$, in time $\mathcal{O}^*\big(\operatorname{rank}(M)^{|R_{i-1}|} |C|^{|X_{i-1}|-|R_{i-1}|}N\big)$ it is possible to compute a table $\widehat{T_i}$ given by*

$$\widehat{T_i}[c, K, W] \equiv \big[c(v_i) \in a(v_i)\big] \sum_{\substack{z \in C^{X_{i-1}} \\ z \sim c}} \widehat{T_{i-1}}\Big[z, K - \big[c(v_i) \in Q\big], W - \omega(v_i) \cdot \big[c(v_i) \in Q\big]\Big]$$

along with a set R_i of reduced vertices of $\widehat{T_i}$ such that $|X_i \setminus R_i| \leq \big(\operatorname{ctw} - |R_i|\big)/2 + 1$ holds. The table $\widehat{T_i}$ then i-represents T_i.

The combination of the previous two lemmas results in a faster algorithm for P. In short, it can be summarized as follows. First, the table T_1 is computed via brute-forcing over all colorings of $X_1 = \{v_1\}$. If v_1 has the degree of one in G, then we apply the algorithm

Reduce and obtain a fully reduced 1-representative $\widehat{T_1}$ of T_1. Then, for $i = 2, \ldots, n$, we first apply Lemma 6 to obtain a i-representative $\widehat{T_i}$ of T_i and then we iteratively apply the **Reduce**-algorithm to $\widehat{T_i}$ to make all vertices in X_i with exactly one neighbor in Y_i reduced, i.e., to obtain a fully reduced i-representative of T_i. Finally, we output the value $\sum_{s \in C} \widehat{T_n}\big[[v_n \mapsto s], K^*, W^*\big]$.

The next theorem claims the running time and the correctness of this approach. The requirement $\sqrt{|C|} \leq \operatorname{rank}(M)$ is technical. Informally speaking, it ensures that the running time spent on the vertices of X_i that have more than one neighbor in Y_i (i.e., they are not reduced and can be colored with all $|C|$ colors) is not too large.

▶ **Theorem 7.** *Let P be a coloring-like problem such that $\sqrt{|C|} \leq \operatorname{rank}(M)$ holds. Then P can be solved in $\mathcal{O}^*\big(\operatorname{rank}(M)^{\operatorname{ctw}} N\big)$ time.*

3.1 Connected Vertex Cover

Now we can apply this result to solve the Connected Vertex Cover (CVC) problem. In [3] we provide a matching lower bound proving its tightness.

▶ **Theorem 8.** *There exists an algorithm that given a graph G and a linear arrangement ℓ of G of cutwidth ctw in time $\mathcal{O}^*(2^{\operatorname{ctw}})$ solves the Connected Vertex Cover problem. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

Proof. We solve the CVC problem by formulating it as a coloring-like problem using the Cut&Count technique. By Theorem 3, to prove the claim, it suffices to provide an algorithm $\mathcal{A}$ that given G, ℓ, a fixed vertex v of G, a weight function $\omega \colon V \to \big[2|V|\big]$, and numbers $K^* \in [n]_0, W^* \in \big[2|V|^2\big]_0$ computes the size of $\mathcal{C}^{K^*}_{W^*}$ modulo 2 in time $\mathcal{O}^*(2^{\operatorname{ctw}})$.

We fit this task into a coloring-like problem as follows. Let $C = \{X, L, R\}$ (X denotes not belonging to a vertex cover while L and R are the sides of a consistent cut), let $Q = \{L, R\}$ (to count the number of vertices in a vertex cover), and let the consistency matrix M have zero-entries $M[L, R] = M[R, L] = M[X, X] = 0$ whilst the remaining entries are equal to one. This consistency matrix reflects that for every edge, there must be an end-vertex in a vertex cover (i.e., in L or R) and there is no edge between L and R. The list function a is given by $a(v) = \{L\}$ and $a(w) = C$ for any $w \neq v$. The instance of a coloring-like problem defined by C, Q, M, $p = 2$, G, ℓ, a, $N = 2|V|$, K^*, W^*, and ω then corresponds to the problem of counting (modulo p) the number of consistent cuts of G of order K^* and weight W^* (see [3] for more details). It turns out that the consistency matrix M has rank of two over $\mathbb{F}_2$: the sum of rows indexed by X and L is equal to the row indexed by R (modulo 2). Thereby, by Theorem 7, the size of $\mathcal{C}^{K^*}_{W^*}$ modulo 2 can be determined in time $\mathcal{O}^*\big(2^{\operatorname{ctw}} 2|V|\big) = \mathcal{O}^*(2^{\operatorname{ctw}})$ and this concludes the proof. ◀

3.2 Connected Dominating Set

A *dominating set* of a graph is a set of vertices such that for every vertex of the graph not in this set, at least one of its neighbors belongs to this set. Connected Dominating Set is defined as follows.

Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a dominating set $S \subseteq V$ of G of cardinality at most k such that $G[S]$ is connected?

By Theorem 3, solving CDS again reduces to the problem of counting the number of certain consistent cuts (i.e., vertex-colorings). Unfortunately, the conditions on these partitions do not yet fit in the framework of a coloring-like problem. The conditions defining a general coloring-like problem in the previous subsection are *universal properties* determined by a consistency matrix M: **all** neighbors of a vertex of a color a are only allowed to be colored with colors from some set B. The satisfaction of these conditions can be verified by checking every edge of the graph independently. The trick from the proof of Lemma 5 that allowed to reduce the vertices with only one incident edge in the ith cut (for $i \in [n]$) utilized this property. However, CDS contains an *existential property*: for every vertex not in the dominating set, there **exists** a neighbor in it. Therefore, the edges cannot be checked independently.

We will overcome this issue as follows. In addition to the colors L, R (for the sides of a consistent cut), and D (for dominated vertices), we introduce two further colors A and F to handle partial solutions. The color A (stands for "allowed") denotes that a vertex is not in the partial solution but it is possibly dominated by it. The color F (stands for "forbidden") denotes that a vertex is not in a partial solution and it is not dominated by it. Then the number of partial solutions to which a vertex does not belong but in which it is dominated is equal to the number of solutions in which it is possibly dominated minus the number of solutions where it is not dominated (informally: $D = A - F$). This is an application of the well-known inclusion-exclusion paradigm. In this form, it has been used by Pilipczuk and Wrochna to solve the Dominating Set problem parameterized by treedepth [20]. In this way, the neighbors of a vertex colored with A are allowed to have any color and the neighbors of a vertex colored with F are not allowed to have color L or R. Thus, we obtain the desired universal property and we can solve this problem relying on a technique similar to the one we used for a general coloring-like problem.

So let

$$M = \begin{array}{c} \\ L \\ R \\ A \\ F \end{array} \begin{array}{c} \begin{array}{cccc} L & R & A & F \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

be the consistency matrix indexed by colors $\{L, R, A, F\}$ and capturing the interpretation of colors sketched above. The crucial observation for the construction of a fast algorithm solving the CDS problem is that the consistency matrix M does not have the full rank: the sum of the first three rows is equal to the fourth one over $\mathbb{F}_2$. Therefore, we may let the color F be reduced and apply a trick from Lemma 5 to reduce vertices with exactly one neighbor in the ith cut (for $i \in [n]$). It is challenging to combine the inclusion-exclusion with this trick but it is possible. On a high level, similarly to a general coloring-like problem, we traverse the linear arrangement from left to right and store the number of partial solutions using colors $\{L, R, A, F\}$ with certain footprints. But every time some vertex occurs on the left side of a cut for the last time (i.e., all edges incident to this vertex have already been considered), we carry out the transformation from the colors A and F to the color D for this vertex. We provide all details in the full version of the paper [3].

▶ **Theorem 9.** *There exists an algorithm that given a graph $G = (V, E)$ and a linear arrangement ℓ of G of cutwidth* ctw*, in time $\mathcal{O}^*(3^{\mathrm{ctw}})$ solves the Connected Dominating Set problem. The algorithm cannot give false positives and may give false negatives with probability at most $1/2$.*

In [3] we provide a matching lower bound proving the tightness.

4 Odd Cycle Transversal

In this section we provide an algorithm solving the Odd Cycle Transversal problem in time $\mathcal{O}^*(2^{\mathrm{ctw}})$. The problem is defined as follows:

> **Input**: A graph $G = (V, E)$ and an integer k.
> **Question**: Is there a subset $S \subseteq V$ of cardinality at most k such that $G - S$ is bipartite?

We present the main idea here and keep the details to the full version of the paper [3]. In [3], we also show that this problem cannot be solved in time $\mathcal{O}^*\big((2-\varepsilon)^{\mathrm{ctw}}\big)$ assuming SETH, hence proving the tightness of our upper bound.

▶ **Lemma 10.** *Let a graph $G = (V, E)$ and a linear arrangement ℓ of G of cutwidth at most* ctw *be given. Let $\hat{G} = (\hat{V}, \hat{E})$ be the graph resulting from G after subdividing each edge twice. Then $\hat{G}$ admits a linear arrangement of width at most* ctw*. Moreover, this linear arrangement can be computed from G and ℓ in polynomial time.*

Proof. We only sketch the proof and refer to the full version of the paper [3] for a complete version. Let $V_0 = \hat{V} \setminus V$ be the set of subdivision vertices. Each vertex w in V_0 lies on the unique induced path on 4 vertices whose end-vertices belong to V. We denote it by $P(w)$. For each vertex $v \in V$, let S_v be the set of vertices adjacent to v in $\hat{G}$. Let S_v^1 be the set of vertices $w \in S_v$ such that the end-vertex $u \neq v \in V$ of $P(w)$ lies before v on ℓ. Let $S_v^2 = S_v \setminus S_v^1$. We build $\hat{\ell}$ from ℓ by inserting the vertices in S_v^1 directly before v and the vertices in S_v^2 directly after v. Then for every vertex $w \in V_0$, the edges of $P(w)$ do not overlap on $\hat{\ell}$. Also for every cut in $\hat{\ell}$, its edges are in bijection with the edges of the corresponding cut in ℓ (namely, we map every edge of $\hat{E}$ to the edge of E it subdivides). This implies $\mathrm{ctw}(\hat{G}) \leq \mathrm{ctw}(G)$. We refer to [3] for details. ◀

The following lemma allows to turn a linear arrangement of G into a path decomposition of $\hat{G}$ with useful properties. The proof can be found in the full version of the paper [3] (see Figure 1 for a sketch).

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph and let ℓ be a linear arrangement of G of cutwidth* ctw*. Then the graph $\hat{G}$ admits a nice path decomposition $\hat{\mathcal{B}}$ of pathwidth at most* ctw *such that every bag of $\hat{\mathcal{B}}$ contains at most one vertex of V. Moreover, the path decomposition $\hat{\mathcal{B}}$ contains a polynomial number of bags and it can be computed in polynomial time from G and ℓ.*

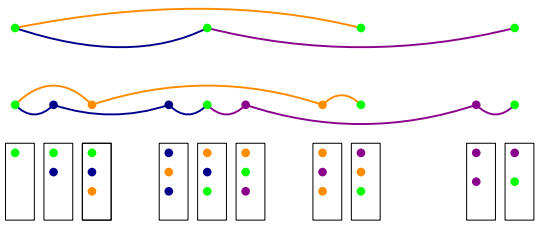


Figure 1 A linear arrangement ℓ of a graph G (top), the linear arrangement $\hat{\ell}$ of the graph $\hat{G}$ (middle), and the path decomposition $\hat{\mathcal{B}}$ of $\hat{G}$ of width at most $\mathrm{ctw}(\ell)$ (bottom). The subdivision-vertices and edges are depicted in the same color as the subdivided edge of G while the vertices of G are green. Each bag of $\hat{\mathcal{B}}$ contains at most one vertex of G (depicted in green).

We state the following lemma and refer to the full version of the paper [3] for a proof:

▶ **Lemma 12.** *Let* $G = (V, E)$ *be a graph and let* $\hat{G}$ *be the graph resulting from* G *by subdividing each edge twice. Then an odd cycle transversal of* G *is an odd cycle transversal of* $\hat{G}$*. Also an odd cycle transversal of* $\hat{G}$ *that only uses vertices of* V *is an odd cycle transversal of* G *as well. Finally, the graph* $\hat{G}$ *admits a minimum odd cycle transversal that only contains vertices from* V*. In particular, both graphs have the same odd cycle transversal number.*

▶ **Theorem 13.** *Given a graph* $G = (V, E)$ *and a linear arrangement* ℓ *of* G *of cutwidth at most* ctw*, the size of the minimum odd cycle transversal of* G *can be computed in* $\mathcal{O}^*(2^{\mathrm{ctw}})$*.*

Proof. Let $\hat{G} = (\hat{V}, \hat{E})$ be the graph resulting from G by subdividing each edge twice, let $\hat{n} = |\hat{V}|$, and let $\hat{\mathcal{B}} = B_1, \dots B_r$ be nice a path decomposition of $\hat{G}$ satisfying Lemma 11. Both $\hat{G}$ and $\hat{\mathcal{B}}$ can be built in polynomial time. By Lemma 12, we can restrict ourselves to odd cycle transversals of $\hat{G}$ that only use vertices from V. So we describe a dynamic programming algorithm that traverses the bags of $\hat{\mathcal{B}}$ from B_1 to B_r to find the size of a minimum odd cycle transversal of $\hat{G}$ with this property, i.e., we seek the smallest set $S \subseteq V$ such that $\hat{G} - S$ is bipartite. We use the set $\mathcal{C} = \{\mathcal{B}, \mathcal{W}, \mathcal{D}\}$ of states. They stand for *black*, *white* (the sides of a bipartition), and *deleted* (i.e., in an odd cycle transversal), respectively. Every table T_i ($i \in [r]$) is indexed by assignments $\delta \colon B_i \to \mathcal{C}$ such that $\delta^{-1}(\mathcal{D}) \subseteq V$ (i.e., only the vertices from V may have the state $\mathcal{D}$) and integers $K \in [\hat{n}]_0$. The entry $T_i(\delta, K)$ is the number of tuples (S, c) such that the following properties hold:

- $S \subseteq \hat{G}[B_1 \cup \cdots \cup B_i] \cap V$,
- $|S| \leq K$,
- $S \cap B_i = \delta^{-1}(\mathcal{D})$,
- c is a proper 2-coloring of $\hat{G}[B_1 \cup \cdots \cup B_i] - S$,
- and c agrees with δ on the values of $B_i \setminus S$, i.e., $c_{|_{B_i \setminus S}} = \delta_{|_{B_i \setminus S}}$.

So the graph G admits an odd cycle transversal of size at most K if and only if $T_r(-, K) > 0$ holds ($-$ denotes the unique assignment to $B_r = \emptyset$). The computation of the tables $T_1, \dots, T_r$ uses standard dynamic programming so we omit the details here (see [3]). Recall that we have $|B_i| \leq \mathrm{ctw} + 1$ and $|B_i \cap V| \leq 1$ for any $i \in [\hat{n}]$ by the choice of $\mathcal{B}$. Since the vertices in $\hat{V} \setminus V$ are only assigned states $\mathcal{B}$ and $\mathcal{W}$ in our dynamic programming routine, the number of table entries of T_i is bounded by $3 \cdot 2^{\mathrm{ctw}} \cdot (\hat{n} + 1)$. Since the size of $\hat{G}$ is linear in the size of G, the running time of the provided algorithm can be bounded by $\mathcal{O}^*(2^{\mathrm{ctw}})$. We refer to [3] for details. ◀

In [3], we show that an analogous idea of the edge-subdivision can be applied to accelerate the Cut&Count dynamic programming and solve the Feedback Vertex Set problem in $\mathcal{O}^*(2^{\mathrm{ctw}})$. In [3] we also provide a matching lower bound.

5 Steiner Tree

In this section we exemplify our lower-bound construction on Steiner Tree. We refer to the full version of the paper [3] for the rest of the lower bounds. The problem is defined as follows.

> **Input**: A graph $G = (V, E)$, a set $T \subseteq V$ of *terminals*, and an integer k.
> **Question**: Is there a set $S \subseteq V$ of cardinality at most k such that $T \subseteq S$ and $G[S]$ is connected?

Here we will sketch a reduction, that shows that this problem cannot be solved in time $\mathcal{O}^*\big((3-\varepsilon)^{\mathrm{ctw}}\big)$ assuming SETH. We refer to the full version for formal proofs [3]. The Steiner Tree problem can be solved in time $\mathcal{O}^*(3^{\mathrm{tw}})$ when parameterized by treewidth [7] so by Lemma 1,

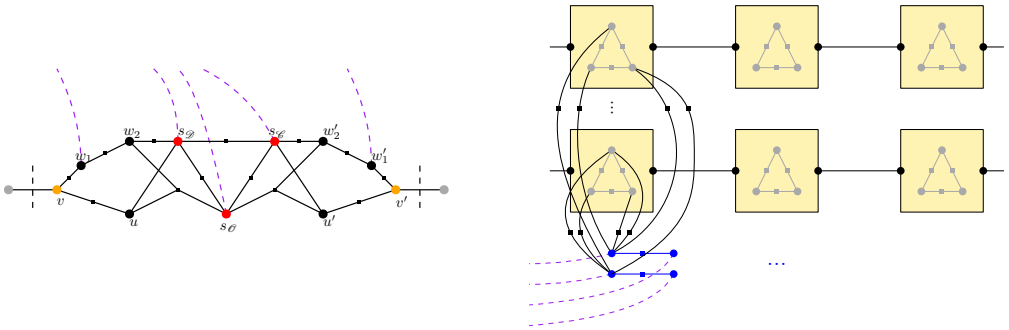
it can also be solved in $\mathcal{O}^*(3^{\mathrm{ctw}})$. Thus, under SETH, this bound is tight. In the following lemma an integer t_0 occurs. It is a constant depending on d and ε from the formulation of SETH we use (see Conjecture 2). The precise choice of t_0 is provided in the full version [3].

▶ **Lemma 14.** *Let $d \in \mathbb{N}$ and let I be an instance of the d-SAT problem with n variables. Let $t_0 \in \mathbb{N}$ be any constant, $s = \lceil n/t_0 \rceil$, and $t = \lceil t_0 \log_3 2 \rceil$. In polynomial time, an instance $\big(G = (V, E), T, k\big)$ of the Steiner Tree problem can be constructed such that G admits a Steiner tree of size at most k if and only if I is a yes-instance. Moreover, G has cutwidth at most $s \cdot t + \mathcal{O}(1)$ and a linear arrangement ℓ of G of such cutwidth can be constructed in polynomial time as well.*

Proof. As mentioned earlier, a formal proof can be found in the full version of the paper [3]. We follow the basic setup introduced by Lokshtanov et al. [15]. First, we partition the variables of I into s groups each of size at most t_0. With each group, we associate a set of t path-like structures (called *paths* for simplicity) resulting in $n' = s \cdot t$ such paths in total. Each path consists of $(2n' + 1)m$ copies of a so-called *path gadget*. In each path gadget, there are two special vertices called the *left-end vertex* and the *right-end vertex*. We add an edge between the right-end vertex of each path gadget and the left-end vertex of the following one on the same path (resulting in a path-like structure). Each path gadget also contains three special vertices $s_{\mathcal{C}}, s_{\mathcal{D}}, s_{\mathcal{O}}$ called *clique vertices*. We add an edge between each pair of clique vertices of the path gadget and subdivide it by a terminal. Thus, any Steiner tree must contain at least two of the clique vertices in each path gadget. The choice of the budget k (i.e., the size of the desired Steiner tree) ensures that exactly two clique vertices from each path gadget belong to any Steiner tree of size exactly k. In order to achieve this, we can define a packing of disjoint sets of vertices (called *components*) of the graph G and provide a lower bound on the number of vertices any Steiner tree must have in each such component. By setting the budget k to be equal to the sum of these lower bounds, we ensure that the Steiner tree matches the lower bound exactly in each component. For instance, the three clique vertices of each path gadget form a component with a lower bound of two. The choice of k also restricts the set of possible Steiner trees and simplifies the analysis. There are several further vertices in a path gadget, we give more details on their function later. We refer to Figure 2 (a) for a visualization of a path gadget.

Given a Steiner tree S of size k of G, we can then assign one of three *states* to every path gadget depending on the clique vertex $s_{\mathcal{C}}, s_{\mathcal{D}}$, or $s_{\mathcal{O}}$ that does not belong to S. We call these states $\mathcal{C}$, $\mathcal{D}$, and $\mathcal{O}$, respectively. We call the set of the jth path gadgets on each path the *jth column*. We partition each column into s sets of t consecutive path gadgets. We call each such set a *bundle*. Note that a bundle contains path gadgets that belong to the same group of paths. A bundle can have one of 3^t state combinations. The key idea of the construction is to define a mapping ϕ that assigns a different state combination to each of at most 2^{t_0} Boolean assignments of the corresponding variable group.

For this purpose, to each bundle B we attach a so-called decoding gadget. A decoding gadget Y is a matching of size 3^t. We subdivide each of its edges by a terminal. Hence, one endpoint of each matching edge must be included in any Steiner tree. Again, the choice of the budget k ensures that any Steiner tree of size k contains exactly one endpoint of each matching edge. Each edge e_σ of the matching corresponds to a different assignment σ of states $\mathcal{C}, \mathcal{D}, \mathcal{O}$ to the path gadgets of the bundle it is attached to. We fix one end-vertex of e_σ to be the *left* endpoint v_σ and the other one to be the *right* endpoint u_σ. We attach Y to its bundle B in such a way that the following holds. Given a Steiner tree S of size k, there exists



(a) A path gadget with orange left- and right-end vertices and red clique vertices.

(b) A group of paths, corresponding to a variable group. Yellow boxes denote single path gadgets. A bundle is a group of vertically aligned path gadgets. A blue decoding gadget is attached to each bundle.

Figure 2 A single path gadget (a) and a group of paths (b) corresponding to a variable group. Terminals are depicted as small squares. Adjacencies to the root-path depicted in purple.

at most one matching edge whose right endpoint belongs to S. Then for the remaining edges of Y, the left endpoint belongs to S. This will allow us to "decode" a truth-value assignment corresponding to S.

We also introduce a so-called *root-path*: it is a simple path, each of its vertices adjacent to a private terminal. Therefore, all vertices of the root-path belong to any Steiner tree. The vertices of this path are then used to ensure the connectivity of the vertices in a Steiner tree and restrict the set of relevant Steiner trees to simplify the analysis. Among others, the clique vertices of each path gadget and the endpoints of each decoding gadget have private neighbors on the root-path.

We partition the set of all columns into $n' + 1$ groups of m consecutive columns. For each group, we assign a private clause C of I to each column in this group. We attach a new terminal to each column as follows. For each $i \in [s]$, let Π_i be the set of all partial Boolean assignments to the ith variable group that satisfy C, and let Σ_i be the set of state assignments over the ith bundle of this column corresponding to the elements of Π_i, i.e., $\Sigma_i = \{\phi(\pi) \mid \pi \in \Pi_i\}$. We add an edge between this terminal and the right-end of each matching edge e_σ in the corresponding decoding gadget attached to the ith bundle with $\sigma \in \Sigma_i$. This ensures that for any Steiner tree S of size k, at least one bundle is assigned a state σ that corresponds to a Boolean assignment π satisfying the clause C.

Hence, given a Boolean assignment π satisfying I, we can build a Steiner tree S of size exactly k in G. Let π_i be the restriction of π to the ith variable group for each $i \in [s]$ and let $\sigma_i = \phi(\pi_i)$. We choose the state of each path gadget so that each bundle in the ith group of paths is assigned the state σ_i. For each decoding gadget attached to a bundle in this group of paths, we add the right end u_{σ_i} of the matching edge corresponding to σ_i to S. This ensures that each terminal added to some column has a neighbor in the Steiner tree, and consecutive path gadgets on each path have the same state in S. This maintains the connectivity of the Steiner tree.

On the other hand, given a Steiner tree of size k, if we could find a group of m columns corresponding to m different clauses such that for each $i \in [s]$, all the bundles of the ith group of paths have the same state assignment on these m columns, then we can define partial assignments of each of the s variable groups such that the union of all partial assignments yields a Boolean assignment satisfying I. For this purpose, we build the path gadgets in such a way that for the ordering of states defined by $\mathcal{D} \preceq \mathcal{O} \preceq \mathcal{C}$, the following holds. If a

path gadget has a state $x \in \{\mathcal{C}, \mathcal{D}, \mathcal{O}\}$ in S, then the following path gadget of the same path has some state y with $x \preceq y$. So the state of path gadgets along each path might change at most twice. Since there are n' paths, at most $2n'$ state changes can occur. And since there are $2n' + 1$ groups of columns, there always exists a group where no such change happens. This ensures the existence of a Boolean assignment satisfying I. We refer to [3] for a formal proof.

Finally, we bound the cutwidth of the graph G. We achieve this by defining a linear arrangement ℓ of width at most $n' + \mathcal{O}(1)$. The arrangement ℓ is built columnwise, i.e., it contains the columns of path gadgets one after another. After each bundle, it contains the decoding gadget attached to it and after each column, the terminal attached to it. Since the vertices of different path gadgets do not interleave on the linear arrangement and since both path gadgets and decoding gadgets have constant size, the edges of G having both ends in the same gadget contribute most a constant to the cutwidth of ℓ. Similarly, the edges incident to decoding gadgets contribute only a constant value. Hence, ignoring the root-path for now, only the edges between consecutive path gadgets are left. Since we created the arrangement columnwise, any cut of G contains at most one edge between two consecutive path gadgets on a fixed path. There are n' such paths and this yields an upper bound of $n' + \mathcal{O}(1)$ on the cutwidth of ℓ without the vertices of the root-path. The ordering of vertices on the root-path is then chosen in such a way that its edges do not overlap on ℓ and the cutwidth is only increased by a constant when these vertices are taken into account. ◀

The following theorem shows that this reduction suffices to prove that the Steiner Tree problem does not admit an $\mathcal{O}^*((3-\varepsilon)^{\mathrm{ctw}})$ algorithm for any positive ε assuming SETH.

▶ **Theorem 15.** *Assuming SETH, there is no algorithm that solves the Steiner Tree problem in time $\mathcal{O}^*\big((3-\varepsilon)^{\mathrm{ctw}}\big)$ for any positive real ε.*

Proof. If such an algorithm exists, for any instance of the d-SAT problem, we can build the graph defined in Lemma 14 for the values t_0 and t depending on ε and d only, and run this algorithm on the created instance. This yields an algorithm for the d-SAT problem with running time $\mathcal{O}^*((2-\delta)^n)$ for a positive value of δ. We omit the details and refer to the full version [3] for a complete proof. ◀

The lower bounds for the remaining five problems considered in this work can be found in the full version of the paper [3]. Apart from lower bounds already mentioned in previous sections, there we also provide the lower bound for Connected Odd Cycle Transversal whose tightness is also implied by an upper bound derived from the parameterization by treewidth.

6 Conclusion

We have initiated the study of the exact complexity of hard connectivity problems parameterized by cutwidth (under SETH) and we provided tight bounds for six problems, namely Connected Vertex Cover, Connected Dominating Set, Odd Cycle Transversal, Feedback Vertex Set, Steiner Tree, and Connected Odd Cycle Transversal.

One specific question that remains open is the exact complexity of Hamiltonian Cycle parameterized by cutwidth (also open for treewidth). For pathwidth, it is known that $(2+\sqrt{2})$ is the optimal base assuming SETH [6]. For more general questions, recall that cutwidth is an edge-separator analogue of pathwidth. It would be interesting to study tight bounds for connectivity problems when parameterized by edge-separator analogues of treewidth such as tree-cut width [22], edge-treewidth [17], and edge-cut width [4].

References

1. 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.
2. 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.
3. Narek Bojikian, Vera Chekan, Falko Hegerfeld, and Stefan Kratsch. Tight bounds for connectivity problems parameterized by cutwidth, 2022. doi:10.48550/arXiv.2212.12385.
4. Cornelius Brand, Esra Ceylan, Robert Ganian, Christian Hatschka, and Viktoriia Korchemna. Edge-cut width: An algorithmically driven analogue of treewidth based on edge cuts. In Michael A. Bekos and Michael Kaufmann, editors, *Graph-Theoretic Concepts in Computer Science - 48th International Workshop, WG 2022, Tübingen, Germany, June 22-24, 2022, Revised Selected Papers*, volume 13453 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2022. doi:10.1007/978-3-031-15914-5_8.
5. Radu Curticapean, Nathan Lindzey, and Jesper Nederlof. A tight lower bound for counting Hamiltonian cycles via matrix rank. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1080–1099. SIAM, 2018. doi:10.1137/1.9781611975031.70.
6. 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.
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. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 150–159. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.23.
8. 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.
9. 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.
10. 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.
11. 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.
12. Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 47:1–47:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.47.
13. Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Inf. Process. Lett.*, 42(6):345–350, 1992. doi:10.1016/0020-0190(92)90234-M.
14. Michael Lampis. Finer tight bounds for coloring on clique-width. *SIAM J. Discret. Math.*, 34(3):1538–1558, 2020. doi:10.1137/19M1280326.

15 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.

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

17 Loïc Magne, Christophe Paul, Abhijat Sharma, and Dimitrios M. Thilikos. Edge-treewidth: Algorithmic and combinatorial properties. *CoRR*, abs/2112.07524, 2021. arXiv:2112.07524.

18 Dániel Marx, Govind S. Sankar, and Philipp Schepper. Degrees and gaps: Tight complexity results of general factor problems parameterized by treewidth and cutwidth. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 95:1–95:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.95.

19 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.

20 Michal 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.

21 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.

22 Paul Wollan. The structure of graphs not admitting a fixed immersion. *J. Comb. Theory, Ser. B*, 110:47–66, 2015. doi:10.1016/j.jctb.2014.07.003.

Twin-Width V: Linear Minors, Modular Counting, and Matrix Multiplication

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

Ugo Giocanti
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Patrice Ossona de Mendez
Centre d'Analyse et de Mathématique Sociales CNRS UMR 8557, Paris, France
Computer Science Institute, Charles University (IUUK), Prague, Czech Republic

Stéphan Thomassé
Univ Lyon, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR5668, France

Abstract

We continue developing the theory around the twin-width of totally ordered binary structures (or equivalently, matrices over a finite alphabet), initiated in the previous paper of the series. We first introduce the notion of parity and linear minors of a matrix, which consists of iteratively replacing consecutive rows or consecutive columns with a linear combination of them. We show that a matrix class (i.e., a set of matrices closed under taking submatrices) has bounded twin-width if and only if its linear-minor closure does not contain all matrices. We observe that the fixed-parameter tractable (FPT) algorithm for first-order model checking on structures given with an $\mathcal{O}(1)$-sequence (certificate of bounded twin-width) and the fact that first-order transductions of bounded twin-width classes have bounded twin-width, both established in *Twin-width I*, extend to first-order logic with modular counting quantifiers. We make explicit a win-win argument obtained as a by-product of *Twin-width IV*, and somewhat similar to bidimensionality, that we call rank-bidimensionality. This generalizes the seminal work of Guillemot and Marx [SODA '14], which builds on the Marcus-Tardos theorem [JCTA '04]. It works on general matrices (not only on classes of bounded twin-width) and, for example, yields FPT algorithms deciding if a small matrix is a parity or a linear minor of another matrix given in input, or exactly computing the grid or mixed number of a given matrix (i.e., the maximum integer k such that the row set and the column set of the matrix can be partitioned into k intervals, with each of the k^2 defined cells containing a non-zero entry, or two distinct rows and two distinct columns, respectively).

Armed with the above-mentioned extension to modular counting, we show that the twin-width of the product of two conformal matrices A, B (i.e., whose dimensions are such that AB is defined) over a finite field is bounded by a function of the twin-width of A, of B, and of the size of the field. Furthermore, if A and B are $n \times n$ matrices of twin-width d over $\mathbb{F}_q$, we show that AB can be computed in time $\mathcal{O}_{d,q}(n^2 \log n)$.

We finally present an *ad hoc* algorithm to efficiently multiply two matrices of bounded twin-width, with a single-exponential dependence in the twin-width bound. More precisely, pipelined to observations and results of Pilipczuk et al. [STACS '22], we obtain the following. If the inputs are given in a compact tree-like form (witnessing twin-width at most d), called twin-decomposition of width d, then two $n \times n$ matrices A, B over $\mathbb{F}_2$ can be multiplied in time $4^{d+o(d)}n$, in the sense that a twin-decomposition of their product AB, with width $2^{d+o(d)}$, is output within that time, and each entry of AB can be queried in time $\mathcal{O}_d(\log \log n)$. Furthermore, for every $\varepsilon > 0$, the query time can be brought to constant time $\mathcal{O}(1/\varepsilon)$ if the running time is increased to near-linear $4^{d+o(d)}n^{1+\varepsilon}$. Notably, the running time is sublinear (essentially square root) in the number of (non-zero) entries.

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

Keywords and phrases Twin-width, matrices, parity and linear minors, model theory, linear algebra, matrix multiplication, algorithms, computational complexity

© Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé;
licensed under Creative Commons License CC-BY 4.0

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 15; pp. 15:1–15:16

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

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.15

Related Version *Full Version*: https://arxiv.org/abs/2209.12023

Funding This work was supported by the ANR projects (French National Research Agency) TWIN-WIDTH (ANR-21-CE48-0014-01) and Digraphs (ANR-19-CE48-0013-01), and is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 810115 – Dynasnet).

1 Introduction

Since its introduction, the treewidth of a graph has proved to be a particularly important concept in graph theory, both in finite model theory [19], in algorithmic design (see for instance the textbook of Cygan et al. [12, Chapter 7]) and in structural analysis (see the Graph Minors series of Robertson and Seymour [27]). This invariant is strongly related to the concept of graph minor. Recall that a minor of a graph is a graph obtained by a succession of edge contractions and vertex or edge deletions. The treewidth of a graph is monotone with respect to this operation, in the sense that the treewidth of a minor of a graph G cannot be larger than the treewidth of G. By a classical theorem by Robertson and Seymour [28], a class of graphs has bounded treewidth if and only if its *minor closure* (that is, the set of all the minors of graphs in the class) does not contain all grids. In particular, a graph with huge treewidth admits a large square grid as a minor. This result, as well as its subsequent qualitative improvements (see [9], for instance), is the basis of the so-called *bidimensionality* algorithmic technique, a win-win argument leveraging low treewidth or the existence of a large grid minor [16].

This paper is the fifth of a series dedicated to a novel invariant of binary structures, the *twin-width* (see Section 2 for formal definitions). This invariant appeared to be particularly relevant for the study of *ordered* binary structures, and especially matrices over a finite alphabet [4]. Some of our results will only need the matrix entries to belong to a finite alphabet, while some will require the entries to belong to a finite field. A *submatrix* of a matrix is obtained by deleting some rows and columns. Most of our results concern (infinite) sets of matrices. In our framework, it will be natural to consider sets of matrices closed under the operation of taking a submatrix. Sets of matrices with this property are called *matrix classes*, analogously to permutation classes, which are classes of permutations closed under taking subpermutations. The alphabet or field being fixed (and having at least two elements), a matrix class is said to be *proper* if it does not include all matrices with entries in the prescribed alphabet or field.

The notion of a *rank Latin division* has been introduced in [4] (see definition in Section 3). It consists of a regular partition of the rows and columns delimiting blocks that either have constant entries or have full rank, in a globally controlled way, where full-rank blocks draw a universal permutation. Just as the grids act as witnesses of a large treewidth, the rank Latin divisions witness a large twin-width: a matrix has either small twin-width or has a submatrix with a large rank Latin division. This is effective: in FPT time, either a contraction sequence of the matrix (witnessing that the twin-width is low) is output or a large rank Latin division is found in a submatrix (witnessing that the twin-width is high).

In this paper, we introduce an operation that plays an analogous role with respect to the twin-width of ordered binary structures that taking a minor plays with treewidth. Applied to 0,1-matrices, this operation consists of a succession of row or column deletions, and replacements of two consecutive rows or columns by their entry-wise sum (modulo 2).

More generally, when applied to matrices over a finite field $\mathbb{F}_p$, this operation consists of a succession of replacements of two consecutive rows or columns by a linear combination of these (over $\mathbb{F}_p$), and any matrix over $\mathbb{F}_p$ obtained this way is a *linear minor* of the original matrix. We say that a matrix class *excludes* a matrix M as a linear minor if M is not a linear minor of a matrix in the class. As expected, every matrix is a linear minor of any matrix having a sufficiently large rank Latin division, thus classes with unbounded twin-width do not exclude any matrix as a linear minor (Lemma 18). It appears that this necessary condition is also sufficient.

▶ **Theorem 1.** *A matrix class over a finite field has bounded twin-width if and only if it excludes some matrix as a linear minor.*

While this characterization of bounded twin-width is related to those expressed in terms of matrix divisions (see Section 2.3 for definitions, and [5, 4] for the corresponding results), the operational nature of linear minors make them closer to what is currently missing in the unordered setting: an operation that is to twin-width what graph minors are to treewidth. In addition, even if the current paper only deals with finite fields, linear minors appear to naturally extend to the case of infinite fields, when the other invariants based on matrix divisions fail to do so. As such, Theorem 1 paves the way to the adequate extension of twin-width to (ordered) binary structures on infinite domains.

Our proof of Theorem 1 involves some (finite) model theoretic arguments. From a model theoretical point of view, a matrix over a finite alphabet of size p is seen as a structure with two linearly ordered sets of elements, the row and column index sets, and p binary predicates expressing the presence of a particular symbol at a specific entry. The logical formulas we will consider will allow distinguishing row and column indices, comparing indices of a same sort, and testing whether the entry of the matrix defined by two indices contains a given symbol.

It appears that the twin-width of ordered structured behaves very nicely with respect to first-order logic and (as we shall see) its modulo-counting extension. This situation is reminiscent of the relation of treewidth (and cliquewidth) with monadic second-order logic [10] and its modulo-counting extension [11].

Indeed, it follows from the results proved in the first paper of the series [5], that first-order model checking (that is: the problem of deciding whether a first-order (FO) sentence φ is satisfied on a structure) is fixed-parameter tractable on matrices over a finite alphabet, when parametrized by φ, the size of the alphabet, and the twin-width of the matrix, provided that some so-called *d-sequence* witnessing the upper bound on the twin-width is given together with the matrix. Here we observe that this result extends to the more expressive first-order logic with modulo-counting (FO+MOD), which is the logic obtained by adding to the standard first-order constructions new quantifiers $\exists^{i[p]}$, where "$\exists^{i[p]} x\ \varphi(x)$" expresses that the number of witnesses x for the formula φ is congruent to i modulo p.

Logical formulas also allow defining new structures from an original structure. This is the essence of the notion of transduction. A *transduction* of binary structures T first colors the elements of a given binary structure $\mathbf{A}$ in all possible ways, thus constructing a set of colored structures. Then, each of these colored structures gives rise to a new binary structure by means of fixed logical formulas, thus constructing a set $\mathsf{T}(\mathbf{A})$ of derived structures, the transduction of $\mathbf{A}$ by T. A *simple interpretation* is the same without the initial coloring process, hence a given structure produces a single other structure. A set $\mathcal{D}$ of structures is a transduction of a set $\mathcal{C}$ of structures if there exists a transduction T with $\mathcal{D} \subseteq \mathsf{T}(\mathcal{C})$. A set $\mathcal{C}$ of structures is *monadically dependent* if the set of all finite graphs is not a transduction of $\mathcal{C}$. (While this actually follows from [1], we will take here this characteristic property as

a definition of monadic dependence.) Note that it has been recently proved [8] that, for hereditary classes of structures (like matrix classes) monadic dependence coincides with the classical notion of dependence (or NIP), which is one of the most fundamental dividing lines in model theory; a proof of such a collapse in the particular case of hereditary classes of ordered graphs was previously shown in [4].

In [5], it was proved that for every FO-transduction T of binary structures, the maximum twin-width of a structure in $\mathsf{T}(\mathbf{A})$ is bounded by a function (depending on T) of the twin-width of $\mathbf{A}$. We also extend this result to FO+MOD-transductions. As an example, there is a transduction L_p such that for every matrix M over $\mathbb{F}_p$, the set $\mathsf{L}_p(M)$ is exactly the set of all linear minors of M. Thus, the closure by linear minors of a matrix class with bounded twin-width also has bounded twin-width, from which Theorem 1 follows.

Together with the results established in [4], this leads to the following equivalence, where the equivalence with the properties in bold is proved in the current paper.

▶ **Theorem 2.** *Given a matrix class $\mathcal{M}$ over a finite field, the following are equivalent.*

- **(i)** *$\mathcal{M}$ has bounded twin-width;*
- **(ii)** ***$\mathcal{M}$ excludes a linear minor;***
- **(iii)** *$\mathcal{M}$ is monadically dependent;*
- **(iv)** *every matrix class that is an* FO*-transduction of $\mathcal{M}$ is proper;*
- **(v)** ***every matrix class that is an* FO+MOD*-transduction of $\mathcal{M}$ is proper;***
- **(vi)** *$\mathcal{M}$ is small (i.e. the number of $n \times n$ matrices in $\mathcal{M}$ is at most $2^{O(n)}$);*
- **(vii)** ***every* FO+MOD*-transduction of $\mathcal{M}$ is small;***
- **(viii)** *$\mathcal{M}$ is subfactorial (i.e. the number of $n \times n$ matrices in $\mathcal{M}$ is less than $n!$, for sufficiently large n).*

Assuming that FPT $\neq$ AW[$*$]*, those conditions are further equivalent to:*

- **(ix)** FO*-model checking is* FPT *on $\mathcal{M}$;*
- **(x)** FO+MOD***-model checking is* FPT *on $\mathcal{M}$.***

We now consider some consequences of these results (for more, see long version).

We call *rank-bidimensional* a parameterized problem defined on matrices whenever the presence of a large rank Latin division in a submatrix incurs an (easy) FPT algorithm. Thus, we get the following.

▶ **Theorem 3.** *Every* FO+MOD*-definable rank-bidimensional problem is in* FPT*.*

From Theorem 3, we obtain FPT algorithms for deciding if a (small) matrix is a linear minor of another matrix, for exactly computing the grid number, mixed number, and grid rank of a matrix (see Section 2 for definitions).

Next we show that, over a finite field, the square M^2 of a matrix M with bounded twin-width has bounded twin-width, by expressing the squaring operation as an FO+MOD-transduction. From the characterization in terms of large rank Latin division of submatrices, it follows that if two matrices A and B have small twin-width, then so does the matrix $\left(\begin{smallmatrix} 0 & A \\ B & 0 \end{smallmatrix}\right)$. As $\left(\begin{smallmatrix} 0 & A \\ B & 0 \end{smallmatrix}\right)^2 = \left(\begin{smallmatrix} AB & 0 \\ 0 & BA \end{smallmatrix}\right)$, we deduce:

▶ **Theorem 4.** *There is a computable function $f : \mathbb{N}^2 \to \mathbb{N}$ such that the following holds. Let A and B be two conformal matrices over a finite field $\mathbb{F}_q$, both of twin-width at most d. Then the twin-width of the product AB is at most $f(d, q)$.*

Note that, by similar arguments, the sum of two conformal matrices over $\mathbb{F}_q$, both of twin-width at most d, has twin-width at most $f'(d, q)$, for some computable function f'.

We now consider the problem from an algorithmic point of view. From a computational perspective, the data structures used to encode matrices over a finite field are crucial. Encoding matrices as bipartite binary structures allows using the machinery developed for (ordered) graphs. In this setting, natural witnesses for twin-width boundedness are *d-sequences* (or, *contraction sequences*), which we mentioned earlier when discussing first-order model checking complexity on classes with bounded twin-width (see Section 2.2 for a formal definition). A naive implementation of the algorithm presented in [4, Theorem 2] runs in time $\exp(\exp(\mathcal{O}(d^2 \log d)))n^3$, and outputs a $2^{\mathcal{O}(d^4)}$-sequence if the twin-width is indeed at most d. We show how to bring the dependence in n down to $\mathcal{O}_d(n^2 \log n)$ (Theorem 8).

Gajarský et al. [18], building on Pilipczuk et al. [26], showed that given an n-vertex graph (or binary structure) G with a d-sequence and a first-order formula $\varphi(x_1, \ldots, x_k)$, one can compute in time $\mathcal{O}_{d,\varphi}(n^{1+\varepsilon})$ (resp. $\mathcal{O}_{d,\varphi}(n)$) a data structure that answers for any query $v_1, \ldots, v_k \in V(G)$ whether $G \models \varphi(v_1, \ldots, v_k)$ in time $\mathcal{O}_{d,\varphi}(1/\varepsilon)$ (resp. $\mathcal{O}_{d,\varphi}(\log \log n)$). This result can be extended to FO+MOD. Then, the squaring operation can be performed by means of a simple interpretation, which gives a near-linear representation of M^2 (in the domain size, that is, sublinear in the number of matrix entries) where entries can be queried in constant time.

We thus obtain an algorithm that takes two matrices of bounded twin-width (*without* witnesses) and outputs their product in quasilinear time in the number of entries.

▶ **Theorem 5.** *Given two $n \times n$ matrices A and B over $\mathbb{F}_q$, both of twin-width at most d, there is an algorithm to compute their product AB in time $\mathcal{O}_{d,q}(n^2 \log n)$.*

However, this algorithm is not practical due to the acute dependence on the twin-width bound. We thus place ourselves in a setting where inputs are already in compact form (witnessing low twin-width). The use of an adapted internal representation is a classical technique of digital computing (Fourier transform, redundant representation of numbers, etc.). Likewise, it appears that convenient representations of matrices of bounded twin-width for matrix computations are *twin-decompositions* [3, 6]. Informally, a twin-decomposition is a tree whose leaves are bijectively mapped to the domain of the structure (here, to the row and column indices), and internal nodes are ordered and naturally correspond to contractions. The binary relations (here, the entries) are encoded by additional edges joining pairs of nodes of the tree, and respecting some specific rules. Every binary structure with bounded twin-width has a twin-decomposition with linearly many extra edges, hence the twin-decomposition forms a degenerate graph. The *width* of the twin-decomposition is related to this degeneracy (see Section 2 for precise definitions). Notice that a twin-decomposition of constant width takes quasilinear space to describe a set of binary relations with possibly quadratically many pairs. We show that a twin-decomposition can be computed from a contraction sequence in quadratic time and observe that, conversely, a contraction sequence can be computed from a twin-decomposition in linear time.

Our last contribution is an *ad hoc* efficient matrix multiplication algorithm for matrices over $\mathbb{F}_p$ with bounded twin-width, which we state here in the case of matrices over $\mathbb{F}_2$.

▶ **Theorem 6** (Theorem 7+[26])**.** *Let A and B be two $n \times n$ matrices over $\mathbb{F}_2$ given in the form of twin-decompositions of width at most d. For every $\varepsilon > 0$, there is a $4^{d+o(d)}n^{1+\varepsilon}$-time algorithm that outputs a twin-decomposition of the product AB of width $2^{d+o(d)}$ and a data structure of size $2^{d+o(d)}n^{1+\varepsilon}$ such that querying an entry of AB takes time $\mathcal{O}(1/\varepsilon)$.*

The following (Theorem 7) is our main technical contribution: an algorithm computing a twin-decomposition of the square M^2 of a matrix M from a twin-decomposition of M, which extends to a matrix multiplication algorithm for matrices each represented by a twin-decomposition.

▶ **Theorem 7.** *Let A and B be two $n \times n$ matrices over $\mathbb{F}_2$ given in the form of twin-decompositions of width at most d. There is a $4^{d+o(d)}n$-time algorithm that outputs a twin-decomposition of the product AB of width $2^{d+o(d)}$.*

Contrary to Theorem 5 that hides a non-elementary dependence in the twin-width bound, the dependence in d given by Theorem 7 is single-exponential. In addition, since Theorem 7 assumes that a twin-decomposition is given in input, the running time is sublinear in the number of matrix entries, n^2 (as opposed to quasilinear for Theorem 5).

The entries of AB can then be queried in time essentially the height of the twin-decomposition, which can be made logarithmic. However, by computing the data structure introduced by Pilipczuk et al. [26] in $\mathcal{O}(d'n^{1+\varepsilon})$ time and space where d' upperbounds the width of a twin-decomposition of the matrix, the entry queries can be performed in $\mathcal{O}(1/\varepsilon)$ time. Theorems 6 and 7 carry over on any finite field $\mathbb{F}_q$ with running time $q^{2d+o(d)}n$ and $2^{\mathcal{O}_q(d)}n^{1+\varepsilon}$, respectively.

An intriguing question concerns the existence of such results over infinite fields (starting with $\mathbb{Q}$). We do not have a direct definition of twin-width of matrices over $\mathbb{Q}$ based on contraction sequences. However linear-minor freeness naturally carries to infinite fields, and thus, it is natural to consider that a class of matrices over $\mathbb{Q}$ has bounded twin-width if its closure under linear minors is not the set of all matrices. This can be equivalently stated via the notion of *grid rank* of a matrix M, i.e., the largest k for which there is a $k \times k$ subdivision of M in which every block has rank at least k. Note that if a matrix has grid rank k, then any linear minor has grid rank at most k. Indeed, one can even show that a class of matrices has bounded grid rank if and only if it does not contain some matrix as a linear minor. We believe that computing the product of two matrices over $\mathbb{Q}$ with bounded grid rank should be done in almost quadratic time, however, we lack a structural decomposition as in the finite field case.

There is a vast literature on computing matrix multiplication, or other natural primitives of linear algebra, on classes of structured matrices. We give a few references on rank-structured matrices (see for instance [15, 21, 7, 30, 29]) and matrices of bounded treewidth.

A square matrix has quasiseparable order s if all its submatrices that are completely above the main diagonal, or completely below it, have rank at most s. Note that on adjacency matrices this is equivalent to the *linear rank-width* parameter (a dense analogue of pathwidth). Pernet [24] shows that multiplying two $n \times n$ matrices with quasiseparable order s can be done in time $\mathcal{O}(s^{\omega-2}n^2)$, where ω is the exponent of matrix multiplication, or $\mathcal{O}(s^3 n)$ if the matrices are given in a suitable compact form [25]. The closely-related semiseparable matrices also have efficient multiplication algorithms [31]. So-called $\mathscr{H}$-matrices (for hierarchical matrices) and $\mathscr{H}^2$-matrices admit almost linear-time algorithms for vector-matrix multiplication [7].

One can naturally extend the treewidth graph parameter to 0,1-matrices M by considering the treewidth of the bipartite graph whose biadjacency matrix is M. Fomin et al. [17] show how to compute the determinant, the rank, and to solve a linear system defined by an $n \times n$ matrix of treewidth k, in time $k^{\mathcal{O}(1)}n$. It was recently shown by Dong et al. [13] how to solve linear programs in expected almost linear-time on matrices of bounded treewidth.

2 Preliminaries

We denote by $[i, j]$ the set of integers $\{i, i+1, \ldots, j-1, j\}$, and $[i]$ is a short-hand for $[1, i]$. We use the standard graph-theoretic notations: $V(G)$, $E(G)$, $N_G[S]$, $N_G(S)$ respectively denote the vertex set, edge set, closed neighborhood of S, open neighborhood of S. Given a matrix M, we may interchangeably denote by $M_{x,y}$ or $M[x, y]$ the entry of M at row x and column y.

2.1 Binary structures and matrices

A relational *signature* Σ is a finite set of relation symbols R, each with a specified arity $r \in \mathbb{N}$. A Σ*-structure* $\mathbf{A}$ is defined by a set A (the *domain* of $\mathbf{A}$) together with a relation $R^{\mathbf{A}} \subseteq A^r$ for each relation symbol $R \in \Sigma$ with arity r. The syntax and semantics of first-order formulas over Σ, or Σ*-formulas* for brevity, are defined as usual. A *binary structure* is a Σ-structure such that every relation symbol of Σ has arity at most 2. An *ordered binary structure* is a structure $\mathbf{A}$ over a signature Σ consisting of unary and binary relation symbols which includes the symbol $<$, defining in $\mathbf{A}$ a total order on the domain of $\mathbf{A}$.

A matrix M over a finite alphabet A with rows R and columns C is viewed as an ordered binary structure with domain $R \uplus C$, equipped with the following relations:

- a unary relation R interpreted as the set of rows,
- an antisymmetric binary relation $<$ which defines a total order on $R \uplus C$, extending the total orders on the rows and columns of M in such a way that the rows precede the columns,
- one binary relation E_a, for each $a \in A$, where $E_a(r, c)$ holds if and only if r is a row, c is not (hence is a column), and a is the entry of M at row r and column c.

2.2 Contraction sequences and twin-width

A *trigraph* G has vertex set $V(G)$, (black) edge set $E(G)$, and red edge set $R(G)$, with $E(G)$ and $R(G)$ being disjoint. The *set of neighbors* $N_G(v)$ of a vertex v in a trigraph G consists of all the vertices adjacent to v by a black or red edge. A d-trigraph is a trigraph G such that the *red graph* $(V(G), R(G))$ has degree at most d. In that case, we also say that the trigraph has *red degree* at most d. A (vertex) *contraction* or *identification* in a trigraph G consists of merging two (non-necessarily adjacent) vertices u and v into a single vertex z, and updating the edges of G in the following way. Every vertex of the symmetric difference $N_G(u) \triangle N_G(v)$ is linked to z by a red edge. Every vertex x of the intersection $N_G(u) \cap N_G(v)$ is linked to z by a black edge if both $ux \in E(G)$ and $vx \in E(G)$, and by a red edge otherwise. The rest of the edges (not incident to u or v) remain unchanged. We insist that the vertices u and v (together with the edges incident to these vertices) are removed from the trigraph.

A d*-sequence* (or *contraction sequence*) is a sequence of d-trigraphs $G_n, G_{n-1}, \ldots, G_1$, where $G_n = G$, $G_1 = K_1$ is the graph on a single vertex, and G_{i-1} is obtained from G_i by performing a single contraction of two (non-necessarily adjacent) vertices. We observe that G_i has precisely i vertices, for every $i \in [n]$. The twin-width of G, denoted by $\mathrm{tww}(G)$, is the minimum integer d such that G admits a d-sequence. See Figure 1 for an illustration.

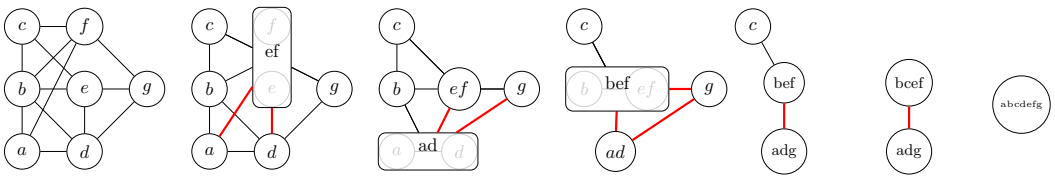


Figure 1 A 2-sequence witnessing that the initial graph has twin-width at most 2.

Twin-width can be generalized from graphs to binary structures in some (functionally) equivalent ways [5, 2]. Here we choose the following definition.

On general binary structures, red edges exist between two vertices $x, y \in V(G_i)$ whenever there are up to four vertices $u \neq v, u' \neq v' \in V(G)$ such that u and u' (which might be the same vertex) were contracted (together with possibly other vertices) to form x, similarly

v and v' were contracted to form y, and the atomic types of (u, v) and of (u', v'), or of (v, u) and of (v', u'), are distinct. If instead, all such pairs (u, v) have the same atomic type, this shared atomic type labels the edge xy in G_i. Contraction sequences, d-sequences, and twin-width are then similarly defined.

In particular, we now have a definition of twin-width for the matrices of Section 2.1.

2.3 Matrix divisions

We will often denote by R (resp. C) the sets of row (resp. column) indices of a matrix M. For $X \subseteq R$ and $Y \subseteq C$, we denote by $M[X, Y]$ the submatrix of M consisting of the entries at rows in X and columns in Y.

A (k, ℓ)-*division* of a matrix M is a pair of partitions $(\mathcal{R} = \{R_1, \dots, R_k\}, \mathcal{C} = \{C_1, \dots, C_\ell\})$ of R and C, respectively, such that every R_i corresponds to consecutive rows, and every C_j, to consecutive columns. A k-*division* is a (k, k)-division. We may call a set of consecutive row or column indices an *interval*. The *grid number*, *mixed number*, *grid rank*, respectively, of a matrix is the largest integer k such that M has a k-division $(\{R_1, \dots, R_k\}, \{C_1, \dots, C_k\})$ for which, for every $i, j \in [k]$, respectively,

- $M[R_i, C_j]$ has a non-zero entry,
- $M[R_i, C_j]$ has at least two distinct rows and at least two distinct columns,
- $M[R_i, C_j]$ has at least k distinct rows or at least k distinct columns.

The divisions are called k-*grid minor*, k-*mixed minor*, and *rank-k division*, respectively. An *interval minor* of a matrix M is a matrix N with k rows and ℓ columns such that M has a (k, ℓ)-division $(\{R_1, \dots, R_k\}, \{C_1, \dots, C_\ell\})$ such that for every $i \in [k]$ and every $j \in [\ell]$, $N_{i,j}$ is an entry of $M[R_i, C_j]$. We call k-Grid Minor, k-Mixed Minor, Rank-k Division, Interval Minor Containment, respectively, the computational problems of deciding if an input matrix has grid number at least k, mixed number at least k, grid rank at least k, and if a matrix is an interval minor of another matrix.

2.4 Computing contraction sequences

Efficiently (in polynomial time, FPT time, or even slice-wise polynomial time) approximating the twin-width of a binary structure remains a challenging open question. However, such an algorithm is known for totally ordered binary structures, or matrices over a finite alphabet [4]. Following [4], we bring the complexity from a cubic down to an almost quadratic dependence in the number of rows and columns.

▶ **Theorem 8.** *Given an $n \times n$ symmetric matrix M over a finite alphabet A, and an integer d, there is an algorithm running in time $\mathcal{O}_{d,|A|}(n^2 \log n)$ that*

- *either correctly reports that the twin-width of M is larger than d,*
- *or outputs an $\exp(\exp(\exp(\mathcal{O}_{|A|}(d^4))))$-sequence for M.*

2.5 Twin-decompositions

A twin-decomposition of a graph G also uses the framework of a rooted carving decomposition, i.e., a rooted binary tree whose leaves are in one-to-one correspondence with the vertices of G. In the case of twin-decompositions though, the internal nodes of the rooted binary tree are totally ordered and the *width* is quite different from how carving-width is defined.

A rooted binary tree with $n - 1$ internal nodes bijectively labeled by ℓ on $[n - 1]$ is said *ranked* if whenever u and v are two distinct internal nodes such that v is a descendant of u, then $\ell(u) < \ell(v)$ holds. By convention, we then decide that all the leaves are labeled $+\infty$

(or equivalently n). For every $i \in [n]$, the *i-th border* of a ranked tree $\mathcal{T}$ with $n-1$ internal nodes is the set of maximal rooted subtrees whose roots have label at least i. We denote by $B_i(\mathcal{T})$ the i-th border of $\mathcal{T}$. It can be easily shown (using that $\mathcal{T}$ is a ranked tree) that the i-th border of $\mathcal{T}$ consists of exactly i subtrees. We denote by $r(\mathcal{T})$ the root of $\mathcal{T}$.

A *twin-decomposition* of an n-vertex graph G is a pair $(\mathcal{T}, \mathcal{B})$ where

(a) $\mathcal{T}$ is a rooted binary **t**ree, ranked by ℓ on $[n-1]$, whose leaves are in one-to-one correspondence with $V(G)$, and

(b) $\mathcal{B}$ (for **b**icliques) is a set of edges over $V(\mathcal{T})$ (disjoint from edge set of $\mathcal{T}$), such that:

1. $\mathcal{B}$ partitions the edge set of G, where an edge between $u, v \in V(\mathcal{T})$ is interpreted as the biclique of G linking every leaf in the subtree of $\mathcal{T}$ rooted at u, to every leaf in the subtree of $\mathcal{T}$ rooted at v, and
2. No edge of $\mathcal{B}$ *crosses* an i-th border, i.e., links a node in a subtree $\mathcal{T}'$ of $B_i(\mathcal{T})$ but *not* the root of $\mathcal{T}'$ to a vertex outside every subtree of $B_i(\mathcal{T})$.

The width of the twin-decomposition $(\mathcal{T}, \mathcal{B})$ is again defined as the maximum red degree among every vertex of every trigraph G_i (as previously defined). See Figure 2 for an illustration of a twin-decomposition corresponding to a particular contraction sequence.

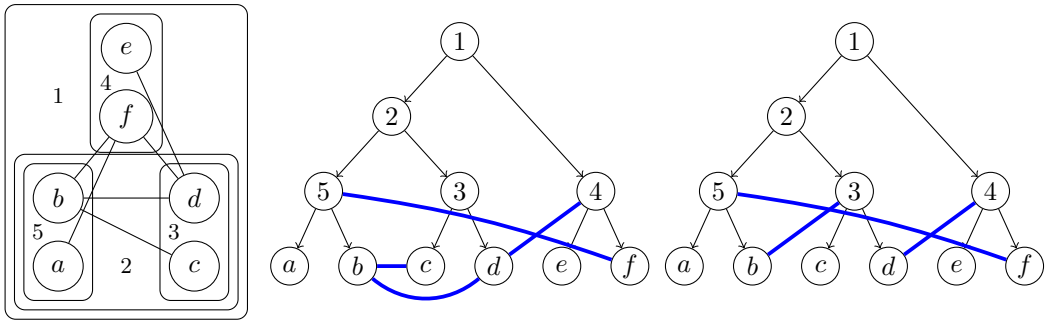


Figure 2 Left: a graph G with a contraction sequence (or partition sequence), where trigraph G_i is obtained after performing the contraction labeled i. Center: the twin-decomposition corresponding to this contraction sequence, with the edges of $\mathcal{B}$ in blue. Right: a ranked tree $\mathcal{T}$ and a partition $\mathcal{B}$ of the edges of G that does *not* make for a twin-decomposition, since the edge $b3$ crosses $B_5(\mathcal{T})$ (and $B_4(\mathcal{T})$).

For our intended purposes in this extended abstract, we only described twin-decompositions for graphs but they readily generalize to binary structures (see long version).

2.6 Computing a twin-decomposition from a contraction sequence

There is an easy linear (in the input size, i.e., in the number of edges) algorithm that, given a d-sequence, computes a corresponding twin-decomposition (of same width d). The *list of triples* of a contraction sequence $G_n, \ldots, G_i, \ldots, G_1$ is $(u_n, v_n, z_n), \ldots, (u_i, v_i, z_i), \ldots, (u_2, v_2, z_2)$ such that the contraction of u_i and v_i in G_i into a new vertex z_i results in G_{i-1}.

▶ **Theorem 9.** *A twin-decomposition of width d of an n-vertex graph G given with the list of triples of a d-sequence can be computed in time $\mathcal{O}(n^2)$.*

A consequence of the second paper of the series is that, given twin-decompositions of bounded width, one can find twin-decompositions of (larger) bounded width, where in addition the tree $\mathcal{T}$ has logarithmic depth.

▶ **Theorem 10** ([2, see Lemma 23 and Proposition 22]). *For every integer d, there is a larger integer D such that every n-vertex graph of twin-width d admits a twin-decomposition of width at most D and depth $\mathcal{O}_d(\log n)$.*

Given a twin-decomposition $(\mathcal{T}, \mathcal{B})$ of G with width d and depth h, the presence of an edge between two vertices of G can be decided in time $\mathcal{O}(dh)$. This yields a linear-space representation of G with edge queries in logarithmic time. Pilipczuk et al further show that:

▶ **Theorem 11** ([26]). *Given a twin-decomposition of width d of an n-vertex graph G, and any $\varepsilon > 0$, there is a data structure of size $\mathcal{O}(dn^{1+\varepsilon})$, computable in time $\mathcal{O}(dn^{1+\varepsilon})$ that supports edge queries of G in time $\mathcal{O}(1/\varepsilon)$.*

All the results mentioned in this section extend from graphs to binary structures.

2.7 Parameterized complexity of model checking

First-order (FO) matrix model checking asks, given a matrix M (or a totally ordered binary structure $\mathcal{S}$) and a first-order sentence φ, if $M \models \varphi$ holds, that is, if φ is true in M. FO model checking is fixed-parameter tractable (FPT) on a matrix class $\mathcal{M}$, with respect to the sentence size and the input matrix, if there exists a constant c and a computable function f, such that $M \models \varphi$ can be decided in time $f(|\varphi|)\,(m+n)^c$, for every $n \times m$ matrix $M \in \mathcal{M}$ and FO sentence φ.

FO model checking of general (unordered) graphs is AW[$*$]-complete [14], and thus very unlikely to be FPT. Indeed FPT $\neq$ AW[$*$] is a much weaker assumption than the already widely-believed Exponential Time Hypothesis [22], and if false, would in particular imply the existence of a subexponential algorithm solving 3-SAT. FO model checking of general binary structures of bounded twin-width given with an $\mathcal{O}(1)$-sequence can even be solved in linear FPT time $f(|\varphi|)\,|U|$, where U is the domain of the structure [5].

Gajarský et al. [18] reproved that result using a different, and more standard formalism. Building on Theorem 11, they also presented an algorithm that inputs a binary structure given with an $\mathcal{O}(1)$-sequence and a formula with some free variables, and after some preprocessing in near-linear time, can answer queries of the form *does the given tuple satisfy the formula in the structure* in constant time.

▶ **Theorem 12** ([18]). *For every $\varepsilon > 0$, given a binary Σ-structure $\mathbf{A}$ on a domain of size n, a d-sequence of $\mathbf{A}$, and a first-order Σ-formula $\varphi(x_1, \ldots, x_k)$, there is a data structure computable in time $\mathcal{O}_{d,\varphi}(n^{1+\varepsilon})$ that given any query of the form $v_1, \ldots, v_k \in A$ reports in time $\mathcal{O}_{d,\varphi}(1/\varepsilon)$ whether $\mathbf{A} \models \varphi(v_1, \ldots, v_k)$ holds.*

In classes of ordered binary structures or matrices, one need not require that the contraction sequence is given in input. Indeed there is an FPT approximation algorithm, that takes a matrix M of twin-width d, and outputs a $g(d)$-sequence of M in time $h(d)\,|M|^{\mathcal{O}(1)}$ [4]. Hence, FO matrix model checking can be solved in FPT time $f(|\varphi|)\,|M|^{\mathcal{O}(1)}$ [4] in classes of bounded twin-width. We observe that this algorithm extends to FO+MOD model checking.

▶ **Theorem 13** (follows with some small adjustments from [5, Section 7]). *Given a d-sequence of a binary structure $\mathcal{S}$, and a prenex* FO+MOD*-sentence φ of quantifier rank ℓ, one can decide $\mathcal{S} \models \varphi$ in time $f(\ell, d)n$ for some computable non-elementary function f.*

The same applies to the fact that bounded twin-width is preserved by transductions.

▶ **Theorem 14** (similarly follows from [5, Section 8]). *Let $\mathcal{C}$ be a set of binary structures with bounded twin-width, and* T *be an* FO+MOD*-transduction. Then* T$(\mathcal{C})$ *has bounded twin-width.*

3 Equivalence of bounded twin-width and linear-minor freeness

Alternatively to the definition given in introduction, linear and parity minors can be characterized by matrix divisions. A *parity minor* of a matrix M is any $n \times m$ matrix N obtained by summing up every cell of an (n, m)-division of a submatrix of M. We denote that by $N \leqslant_{\text{pm}} M$. See Figure 3 for an illustration.

$$M = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \left[\begin{array}{cc|cc|cc|cc} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \qquad N = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Figure 3 A parity minor, equivalently linear minor, N of a matrix M over $\mathbb{F}_2$. In the middle, the deleted rows and columns of M are in light gray, while the $(4, 3)$-division of the remaining submatrix giving rise to N is represented with solid black lines.

Let us call $\mathbb{F}$-*weighting of* M any mapping $w : \text{rows}(M) \cup \text{cols}(M) \to \mathbb{F}$. Equivalently a linear minor of a matrix M over a finite field $\mathbb{F}$ is any $n \times m$ matrix N obtained from an $\mathbb{F}$-weighting w and (n, m)-division $\mathcal{D} = (\{R_1, \dots, R_n\}, \{C_1, \dots, C_m\})$ of M by replacing every cell $M[R_i, C_j]$ of $\mathcal{D}$ by the single entry $\sum_{r \in R_i, c \in C_j} w(r)w(c)M_{r,c}$, that is, setting $N_{i,j} = \sum_{r \in R_i, c \in C_j} w(r)w(c)M_{r,c}$. We denote that by $N \leqslant_{\text{lm}} M$.

▶ **Observation 15.** *Over $\mathbb{F}_2$, parity minors and linear minors coincide.*

The *linear-minor closure* of $\mathcal{M}$, denoted by $\text{Clos}_{\text{lm}}(\mathcal{M})$, is the matrix class of all matrices N which are linear minors of some M in $\mathcal{M}$. We say that $\mathcal{M}$ is *parity-minor free* (resp. *linear-minor free*) if its parity-minor closure (resp. linear-minor closure) is not the set of all $\mathbb{F}$-matrices. We first show that bounded twin-width matrix classes over finite fields are linear-minor free, and in particular parity-minor free.

▶ **Lemma 16.** *Let $\mathbb{F}$ be a finite field. For every matrix class $\mathcal{M}$ over $\mathbb{F}$ of bounded twin-width, $\mathcal{M}$ is linear-minor free.*

Proof sketch. We can express the set of linear minors of a matrix by a FO+MOD-transduction. Indeed the modulo-counting quantifiers allow one to write a first-order formula summing up the entries in a given contiguous submatrix. Thus by Theorem 14, $\text{Clos}_{\text{lm}}(\mathcal{M})$ has bounded twin-width, and cannot be the set of all $\mathbb{F}$-matrices. ◀

For the converse, we will need the notion of *rank Latin divisions* previously introduced [4]. For any integers $k \geqslant 2$ and $d \geqslant 1$, a *rank-k Latin d-division* of a $kd^2 \times kd^2$ matrix M is a regular d-division $\mathcal{D}$ of M that can be refined into a regular d^2-division $((R_1, \dots, R_{d^2}), (C_1, \dots, C_{d^2}))$ such that

- $\forall i \in [d^2]$, $M[R_i, C_j]$ is constant for every $j \in [d^2]$ but one j_i for which it has rank k,
- $\forall j \in [d^2]$, $M[R_i, C_j]$ is constant for every $i \in [d^2]$ but one i_j for which it has rank k,
- and every cell of $\mathcal{D}$ contains exactly one $M[R_i, C_j]$ with rank k.

It was previously shown that matrix classes with unbounded twin-width contain matrices with rank-k Latin d-divisions for arbitrarily large values of k and d. For our purpose we will only need $k = 2$ and d diverging.

▶ **Lemma 17** ([4]). *Let $\mathcal{M}$ be a matrix class of unbounded twin-width over a finite field. Then for every d, there is a matrix $M \in \mathcal{M}$ with a rank-2 Latin d-division.*

Equipped with that technical lemma, we can show the following (see long version).

▶ **Lemma 18.** *Let $\mathbb{F}$ be a finite field and $\mathcal{M}$ be a matrix class of $\mathbb{F}$-matrices. If $\mathcal{M}$ is linear-minor free then it has bounded twin-width.*

4 Fixed-parameter algorithms for matrix division problems

We show how to use Theorem 13 and the approximation algorithm of matrix twin-width [4, Theorem 2], to decide matrix problems involving a division (of a submatrix) with some FO+MOD-definable properties in fixed-parameter time. This is based on a win-win argument generalizing the algorithmic scheme of Guillemot and Marx [20] to solve Permutation Pattern, and somewhat resembling the bidimensionality technique [16]. It allows for instance to detect a k-grid minor or a k-mixed minor in an $n \times n$ matrix, or to decide if a $k \times k$ matrix is a parity or linear minor of an $n \times n$ matrix in time $f(k)n^{\mathcal{O}(1)}$.

▶ **Theorem 19** ([4]). *Given as input an $n \times m$ matrix M over a fixed finite field $\mathbb{F}$, and an integer k, there is an $f(k)(n+m)^{\mathcal{O}(1)}$-time algorithm which returns*
- *either a rank-k Latin division of a submatrix of M,*
- *or a contraction sequence certifying that $\mathrm{tww}(M) \leqslant g(k)$.*

where f and g are computable functions.

A parameterized problem Π, taking as input a matrix over a fixed finite field and a non-negative integer, is FO+MOD-*definable* if, there is a computable function f, and for every non-negative integer k, there is a FO+MOD$[\tau]$ sentence $\varphi_{\Pi,k}$ of size $f(k)$ such that (M,k) is a YES-instance of Π if and only if $M \models \varphi_{\Pi,k}$. A parameterized problem Π, taking as input a matrix M and a non-negative integer k, is said *rank-bidimensional* if, for some computable functions f and g, the existence of a rank-$f(k)$ Latin division in M (i.e., a submatrix of M has a rank-$f(k)$ Latin division) permits to decide Π in time $g(k)|M|^{\mathcal{O}(1)}$. We can now state the main observation of this section.

▶ **Theorem 20.** *Every* FO+MOD-*definable rank-bidimensional problem is in* FPT.

Proof. Let Π be a rank-bidimensional problem, with computable functions f and g. Let (M,k) be an input of Π. We run the algorithm of Theorem 19 with parameter $f(k)$. In time $f'(k)|M|^{\mathcal{O}(1)}$, this either yields a rank-$f(k)$ division of a submatrix of M, and we can decide (M,k) in further time $g(k)|M|^{\mathcal{O}(1)}$ (since Π is rank-bidimensional), or a $g'(k)$-sequence of M, and we can conclude by Theorem 13 (since Π is FO+MOD-definable). ◀

As a corollary of Theorem 20, we obtain for instance that deciding if N is a parity minor of M is fixed-parameter tractable in the size of N. Indeed we show (in the long version) how to express the parity minor containment with FO+MOD sentences.

▶ **Theorem 21.** *Let N be a $k \times k$ matrix, and M be an $n \times m$ matrix, both over $\mathbb{F}_2$. One can decide $N \leqslant_{pm} M$ in time $f(k)(n+m)^{\mathcal{O}(1)}$ for some computable function f.*

Similarly we can invoke Theorem 20 for the following problems. In the following theorem, Linear Minor Containment is the problem of deciding $N \leqslant_{\mathrm{lm}} M$, given two matrices N, M over a fixed finite field, parameterized by $|N|$.

▶ **Theorem 22.** *Linear Minor Containment, Interval Minor Containment, k-Grid Minor, k-Mixed Minor, Rank-k Division, Permutation Pattern are in* FPT.

5 Products of bounded twin-width matrices over a finite field

First we show that if a matrix class $\mathcal{M}$ over a finite field has bounded twin-width, then so does its set of squares $\mathcal{M}^2 = \{AB : \text{A and B are two conformal matrices of } \mathcal{M}\}$.

▶ **Theorem 23.** *There is a function $f : \mathbb{N}^3 \to \mathbb{N}$ such that for every conformal matrices A and B over $\mathbb{F}_q$, The product AB (over $\mathbb{F}_q$) has twin-width at most $f(\mathrm{tww}(A), \mathrm{tww}(B), q)$.*

Proof. Since $\begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix} = \begin{pmatrix} AB & 0 \\ 0 & BA \end{pmatrix}$ and

$$\mathrm{tww}\left(\begin{pmatrix} 0 & X \\ Y & 0 \end{pmatrix}\right) = \mathrm{tww}\left(\begin{pmatrix} X & 0 \\ 0 & Y \end{pmatrix}\right) = \max(\mathrm{tww}(X), \mathrm{tww}(Y), 2),$$

we shall just prove that there is a function g such that M^2 has twin-width at most $g(\mathrm{tww}(M))$, for every square matrix M over $\mathbb{F}_q$. There is a simple FO+MOD-interpretation S such that $\mathsf{S}(M) = M^2$. Indeed one can keep the relations R and $\prec$ as in M, and express $E_i^{M^2}(x, y)$ as

$$\bigvee_{\substack{a:[q-1]^2 \to [0,q-1] \\ \sum_{j,k \in [q-1]^2} a(j,k) \cdot (\tilde{j}\tilde{k}) = \tilde{i}}} \bigwedge_{j,k \in [q-1]} \exists^{a(j,k)[q]} z \; E_j^M(x, z) \;\wedge\; E_k^M(z, y),$$

where $\tilde{i}$ is the element of $\mathbb{F}_q$ corresponding to relation E_i. The expression $\tilde{j}\tilde{k}$ is a product in $\mathbb{F}_q$, while $a(j,k) \cdot (\tilde{j}\tilde{k})$ is the sum of $a(j,k)$ occurrences of $\tilde{j}\tilde{k}$. As every element of $(\mathbb{F}_q, +)$ has an order dividing q, it is enough to count the number of pairs $(\tilde{j}, \tilde{k}) = (M_{x,z}, M_{z,y})$ modulo q, which the formula does. We finally invoke Theorem 14 to conclude that $\mathrm{tww}(M^2)$ is bounded by a function of $\mathrm{tww}(M)$ and q. ◀

▶ **Theorem 24.** *Let q be a prime power, and d be a natural. Let A, B be two $n \times n$ matrices over $\mathbb{F}_q$, both of twin-width at most d. One can compute the product AB in time $\mathcal{O}_{d,q}(n^2 \log n)$.*

Proof. By Theorem 8, we compute an $\mathcal{O}_{d,q}(1)$-sequence for

$$M = \begin{pmatrix} 0 & A \\ B & 0 \end{pmatrix}, \text{ in time } \mathcal{O}_{d,q}(n^2 \log n).$$

We conclude either by turning this contraction sequence into a twin-decomposition in time $\mathcal{O}_{d,q}(n^2)$, by Theorem 9, and invoking the upcoming practical matrix squaring of Theorem 25, or by combining the FO+MOD-interpretation of Theorem 23 (and Theorem 14) with the efficient algorithm of Gajarský et al. [18] (see Theorem 12) to compute the interpretations of bounded twin-width structures. We can finally read off the top-left block AB in M^2 in time $\mathcal{O}_{d,q}(n^2)$.

If we chose the former approach, we now have a twin-decomposition $(\mathcal{T}, \mathcal{B})$ of AB. We can initialize an $n \times n$ matrix to all 0 entries, and for each edge of $\mathcal{B}$ labeled ℓ, fill the corresponding entries with ℓ. This takes quadratic time since we access each matrix entry at most once. If we instead went with the latter approach, we shall simply make $(q-1)n^2$ constant-time queries to build AB, $q-1$ for each entry of AB. ◀

The most technical contribution of the paper is the following, stated and shown in the language of edge-colored graphs in the long version. For every prime power $q = p^\alpha$, we let $m(q)$ denote the cost of basic arithmetic computations in $\mathbb{F}_q$; here only addition, subtraction and multiplication are needed.

▶ **Theorem 25.** *For every prime power $q \geqslant 2$, there is an $\mathcal{O}(m(q)d^2q^{2d}n)$-time algorithm that, given a twin-decomposition $(\mathcal{T}, \mathcal{B})$ of width d of a symmetric $n \times n$ matrix M over $\mathbb{F}_q$, outputs a twin-decomposition of width $\mathcal{O}(d^2q^d)$ of M^2.*

Note that in practice, if $q = p^\alpha$ with p prime, one can choose $m(q) = \mathcal{O}(\log_p(q)^2 \log(p)^2)$ (see for example [23, Table 2.8]). The comparatively good dependence on twin-width is achieved by working directly on the twin-decomposition and problem-specific insights. The algorithm follows three steps. In the first step, a contraction sequence is computed for M^2 as a refinement of the one for M. The contraction sequence naturally yields the tree $\mathcal{T}'$ of a twin-decomposition $(\mathcal{T}', \mathcal{B}')$, where one shall now find $\mathcal{B}'$. The next step aims to obtain a set $\mathcal{B}_1$ encoding M^2 (but not yet a twin-decomposition). This set is then cleaned in the third and last step into a twin-decomposition $(\mathcal{T}', \mathcal{B}')$ of M^2 of width at most exponential in the width of the twin-decomposition $(\mathcal{T}, \mathcal{B})$.

References

1 John T. Baldwin and Saharon Shelah. Second-order quantifiers and the complexity of theories. *Notre Dame Journal of Formal Logic*, 26(3):229–303, 1985.

2 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width II: small classes. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1977–1996, 2021. doi:10.1137/1.9781611976465.118.

3 Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 35:1–35:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.35.

4 Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, Pierre Simon, Stéphan Thomassé, and Szymon Torunczyk. Twin-width IV: ordered graphs and matrices. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 924–937. ACM, 2022. doi:10.1145/3519935.3520037.

5 Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

6 Édouard Bonnet, Jaroslav Nesetril, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *CoRR*, abs/2102.06880, 2021. arXiv:2102.06880.

7 Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Introduction to hierarchical matrices with applications. *Engineering analysis with boundary elements*, 27(5):405–422, 2003.

8 S. Braunfeld and M.C. Laskowski. Existential characterizations of monadic NIP, 2022. doi:10.48550/arXiv.2209.05120.

9 Julia Chuzhoy and Zihan Tan. Towards tight(er) bounds for the excluded grid theorem. *J. Comb. Theory, Ser. B*, 146:219–265, 2021. doi:10.1016/j.jctb.2020.09.010.

10 B. Courcelle. Graph rewriting: an algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, chapter 5, pages 142–193. Elsevier, Amsterdam, 1990.

11 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

12 Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015.

13 Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: a multiscale representation of robust central path. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1784–1797. ACM, 2021. doi:10.1145/3406325.3451056.

14 Rodney G. Downey, Michael R. Fellows, and Udayan Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. In Douglas S. Bridges, Cristian S. Calude, Jeremy Gibbons, Steve Reeves, and Ian H. Witten, editors, *First Conference of the Centre for Discrete Mathematics and Theoretical Computer Science, DMTCS 1996, Auckland, New Zealand, December, 9-13, 1996*, pages 194–213. Springer-Verlag, Singapore, 1996.

15 Yuli Eidelman and Israel Gohberg. On a new class of structured matrices. *Integral Equations and Operator Theory*, 34(3):293–324, 1999.

16 Fedor V. Fomin, Erik D. Demaine, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Bidimensionality. In *Encyclopedia of Algorithms*, pages 203–207. Springer, 2016. doi:10.1007/978-1-4939-2864-4_47.

17 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

18 Jakub Gajarský, Michal Pilipczuk, Wojciech Przybyszewski, and Szymon Torunczyk. Twin-width and types. 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 123:1–123:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.123.

19 Erich Grädel, Phokion G Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y Vardi, Yde Venema, Scott Weinstein, et al. *Finite Model Theory and its applications*. Springer, 2007.

20 Sylvain Guillemot and Dániel Marx. Finding small patterns in permutations in linear time. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 82–101, 2014. doi:10.1137/1.9781611973402.7.

21 Wolfgang Hackbusch. A sparse matrix arithmetic based on H-matrices. part I: introduction to h-matrices. *Computing*, 62(2):89–108, 1999. doi:10.1007/s006070050015.

22 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.

23 Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. URL: http://www.cacr.math.uwaterloo.ca/hac/.

24 Clément Pernet. Computing with quasiseparable matrices. In Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao, editors, *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016*, pages 389–396. ACM, 2016. doi:10.1145/2930889.2930915.

25 Clément Pernet and Arne Storjohann. Time and space efficient generators for quasiseparable matrices. *J. Symb. Comput.*, 85:224–246, 2018. doi:10.1016/j.jsc.2017.07.010.

26 Michal Pilipczuk, Marek Sokolowski, and Anna Zych-Pawlewicz. Compact representation for matrices of bounded twin-width. 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 52:1–52:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.52.

27 Neil Robertson and Paul D. Seymour. Graph minors. I. Excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

28 Neil Robertson and Paul D. Seymour. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92–114, 1986. doi:10.1016/0095-8956(86)90030-4.

29 Christopher De Sa, Albert Gu, Rohan Puttagunta, Christopher Ré, and Atri Rudra. A two-pronged progress in structured dense matrix vector multiplication. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1060–1079. SIAM, 2018. doi:10.1137/1.9781611975031.69.

30 Raf Vandebril, Marc Van Barel, Gene Golub, and Nicola Mastronardi. A bibliography on semiseparable matrices. *Calcolo*, 42(3):249–270, 2005.

31 Jianlin Xia, Shivkumar Chandrasekaran, Ming Gu, and Xiaoye S. Li. Fast algorithms for hierarchically semiseparable matrices. *Numer. Linear Algebra Appl.*, 17(6):953–976, 2010. doi:10.1002/nla.691.

Non-Adaptive Proper Learning Polynomials

Nader H. Bshouty
Department of Computer Science, Technion, Haifa, Israel

Abstract

We give the first polynomial-time *non-adaptive* proper learning algorithm of Boolean sparse multivariate polynomial under the uniform distribution. Our algorithm, for s-sparse polynomial over n variables, makes $q = (s/\epsilon)^{\gamma(s,\epsilon)} \log n$ queries where $2.66 \leq \gamma(s,\epsilon) \leq 6.922$ and runs in $\tilde{O}(n) \cdot poly(s, 1/\epsilon)$ time. We also show that for any $\epsilon = 1/s^{O(1)}$ any non-adaptive learning algorithm must make at least $(s/\epsilon)^{\Omega(1)} \log n$ queries. Therefore, the query complexity of our algorithm is also polynomial in the optimal query complexity and optimal in n.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Polynomial, Learning, Testing

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.16

Related Version *Full Version*: https://eccc.weizmann.ac.il/report/2022/098/

1 Introduction

In this paper, we study the non-adaptive learnability of the class of sparse (multivariate) polynomials over GF(2). A polynomial over GF(2) is the sum in GF(2) of monomials, where a monomial is a product of variables. It is well known that every Boolean function has a unique representation as a (multilinear) polynomial over GF(2). A Boolean function is called s-sparse polynomial if its unique polynomial expression contains at most s monomials.

In the learning model [1, 19], the learning algorithm has access to a black-box query oracle to a function f that is s-sparse polynomial. The goal is to run in $poly(n, s, 1/\epsilon)$ time, make $poly(n, s, 1/\epsilon)$ black-box queries and, with probability at least $2/3$, learn a Boolean function h that is ϵ-close to f under the uniform distribution, i.e., $\mathbf{Pr}_x[f(x) \neq h(x)] \leq \epsilon$. The learning algorithm is called *proper learning* if it outputs a s-sparse polynomial. The learning algorithm is called *exact learning algorithm* if $\epsilon = 0$.

In the adaptive learning algorithms, the queries can depend on the answers to the previous queries, wherein in the non-adaptive learning algorithms, the queries are independent of the answers to the previous queries.

Adaptive proper and non-proper learning algorithms of s-sparse polynomials that run in polynomial-time and make a polynomial number of queries have been studied by many authors [2, 4, 5, 6, 7, 10, 12, 11, 14, 15, 17, 18].

Non-adaptive proper and non-proper learning algorithms of s-sparse polynomials have been studied in [13, 16, 17]. In [16], Hellerstein and Servedio gave a non-proper learning algorithm that learns only from random examples under any distribution (PAC-learning without black-box queries, [19]) that runs in time $n^{O(n \log s)^{1/2}}$. Roth and Benedek, [17], show that for any $s \geq 2$ polynomial-time proper PAC-learning without black-box queries of s-sparse polynomials implies RP=NP. They also gave a non-adaptive proper exact learning ($\epsilon = 0$) algorithm that makes $(n/\log s)^{\log s}$ black-box queries. They also show that to exactly learn s-sparse polynomial, you need at least $(n/\log s)^{\log s}$ black-box queries. See also [13, 14].


© Nader H. Bshouty;
licensed under Creative Commons License CC-BY 4.0

To the best of our knowledge, no polynomial-time *non-adaptive* proper (or non-proper) learning algorithm is known for s-sparse polynomials. In this paper, we give the first polynomial-time non-adaptive proper learning algorithm for sparse multivariate polynomial. We prove

▶ **Theorem 1.** *There is a non-adaptive proper learning algorithm for s-sparse polynomial that runs in polynomial-time and makes $(s/\epsilon)^{O(1)} \log n$ queries.*

In [17], Roth and Benedek show that any deterministic non-adaptive learning algorithm with $\epsilon = 1/s^{O(1)}$ must make at least $(s/\epsilon)^{\Omega(1)} \log n$ queries. We prove the same bound for randomized algorithms. This shows that our query complexity in Theorem 1 is also polynomial in the optimal query complexity and optimal in n.

Our paper is organized as follows. In Section 2, we give the technique used for our algorithm in Theorem 1. In Section 3, we provide some definitions and preliminary results. The learning algorithm is given in Section 4. In Section 5, we give another algorithm that has sublinear complexity in $1/\epsilon$ when ϵ is "very" small. Then, in Section 6, we give lower and upper bounds for the query complexity of algorithms with unlimited computational power. Appendices A and B are dedicated to two proofs of two lemmas that are needed for the main algorithm.

2 Techniques

In this section, we give a brief overview of the techniques used for the main result, Theorem 1. For the other results, see Sections 5 and 6.

Our learning algorithm is composed of the following five reductions.

The first reduction reduces the non-adaptive learning of s-sparse polynomials to non-adaptive exact learning s-sparse polynomials with monomials of size at most $d = O(\log(s/\epsilon))$, i.e., degree-d s-sparse polynomials. Given a s-sparse polynomial f, we assign each variable x_i to 0 with probability[1] $p = 1 - 2^{-\Theta(\sqrt{\log s/\log(1/\epsilon)})}$. In this *partial assignment*, with high probability, monomial of size greater than d vanish. Then we learn the resulted functions. We take enough random zero assignments (partial assignments) so that, with high probability, for each small monomial M of f, there is an partial assignment q that M does not vanish under q. Collecting all the monomials of degree at most d in all the resulted functions gives a hypothesis that ϵ-approximate the target function f.

The second reduction reduces the non-adaptive exact learning of degree-d s-sparse polynomials to non-adaptive exact learning only the monomials of degree d of the degree-d s-sparse polynomial. This is done by running all the algorithms that learn the monomials of degree i, $i \in [d]$ on the target f (which is a degree-d s-sparse polynomial). After learning the monomials of degree d in f, we let g be their sum and then learn the monomials of degree $d-1$ in $f+g$ (which are the monomials of degree $d-1$ in f). Then continue the same way with $f+g$.

The third reduction reduces the non-adaptive exact learning of the monomials of degree d in the degree-d s-sparse polynomials to exact learning degree-d s-sparse *determinant-polynomials*. A degree-d s-sparse determinant-polynomial is a polynomial over the new nd variables $\{y_{i,j}\}_{i\in[n],j\in[d]}$ of the form

$$\sum_{I\in S} \det(\mathcal{Y}^{(I)})$$

[1] We can also choose a constant p, but this value of p is the one that minimizes the query complexity of the tester.

where S is a set of d-subsets of $[n]$, $|S| = s$ and for $I = \{i_1, \ldots, i_d\} \in S$, $\mathcal{Y}^{(I)}$ is the $d \times d$ matrix where $\mathcal{Y}^{(I)}_{k,j} = y_{i_k,j}$. Notice that the degree-d s-sparse determinant-polynomials is a homogeneous degree-d $(d!s)$-sparse polynomial over dn variables. For this reduction, we use the operator ϕ_d defined in [9] that changes the degree-d s-sparse polynomial f to degree-d s-sparse determinant-polynomial. This operator is linear, changes each monomial $\Pi_{i\in I}x_i$ of degree d in f to $\det(\mathcal{Y}^{(I)})$, removes monomials of degree less than d, and each black-box query to $\phi_d f$ can be simulated by $2^d = poly(s/\epsilon)$ queries to f. Obviously, if we can learn S in the degree-d s-sparse determinant-polynomial $\phi_d f$, we can learn the monomials of degree d of the degree-d s-sparse polynomials f.

Notice that the monomials of degree-d s-sparse determinant-polynomials are of the form $y_{i_1,1}y_{i_2,2}\cdots y_{i_d,d}$. This reduction aims to change the polynomial to a homogeneous polynomial that their monomials are of the form $y_{i_1,1}\cdots y_{i_d,d}$, so we can apply the following (fourth) reduction, which can be applied only to such polynomials.

The fourth reduction uses the simulation in Lemma 2 (see also [9]), which shows that any black-box query in $\mathrm{GF}(2^t)^n$ to a homogeneous degree-d polynomial f with monomials of the form $y_{i_1,1}\cdots y_{i_d,d}$ can be simulated in $poly(2^d,t)\tilde{O}(n) = poly(s/\epsilon)\tilde{O}(n)$ time by $O(2^{1.66d}t) = poly(s/\epsilon)t$ black-box queries in $\mathrm{GF}(2)^n$. We will choose $t = (d+1)\log n$, which is necessary for applying the following (fifth) reduction and the final algorithm.

Notice that this reduces exact learning degree-d s-sparse determinant-polynomials with black-box queries in $\mathrm{GF}(2)^{dn}$ to exact learning degree-d s-sparse determinant-polynomials with black-box queries in $\mathrm{GF}(2^t)^{dn}$. There are many non-adaptive learning algorithms of sparse polynomials over large fields. However, unfortunately, as we said before, degree-d s-sparse determinant-polynomials are degree-d $(d!s)$-sparse polynomials and $d!s = s^{\Omega(\log\log s)}$, which makes the time and query complexity of the algorithm super-polynomial. So, we need another reduction to reduce the number of monomials.

The fifth reduction reduces non-adaptive exact learning degree-d s-sparse determinant-polynomials with queries in $\mathrm{GF}(2^t)^{dn}$ to non-adaptive exact learning degree-d s-sparse (multilinear) polynomial over $\mathrm{GF}(2^t)$ with queries in $\mathrm{GF}(2^t)^n$. We simply choose, uniformly at random, dn elements $\alpha_{i,j}$ in $\mathrm{GF}(2^t)$ and substitute $y_{i,j} = \alpha_{i,j}x_i$. This changes each $\det(\mathcal{Y}^{(I)})$ to the monomial $(\det\Gamma^{(I)})\prod_{i\in I}x_i$ where for $I = \{i_1, i_2, \ldots, i_d\}$, $\Gamma^I_{k,j} = \alpha_{i_k,j}$. We then argue that whp, $\det\Gamma^{(I)} \neq 0$ and, therefore, we can find all the terms of the determinant-polynomials of the target.

Combining all the above, we reduce non-adaptive learning s-sparse polynomial to non-adaptive exact learning degree-d s-sparse polynomial with black-box that answer queries in $\mathrm{GF}(2^t)^n$. Now we use the algorithm of Ben-Or and Tiwari [3] with some modification to learn degree-d s-sparse multilinear polynomial over $\mathrm{GF}(2^t)$ with $2s$ queries. We show that to use Ben-Or and Tiwari algorithm, it is enough to have $t = (d+1)\log n$. This implies Theorem 1.

The following depicts the above reductions. Here, $\mathbb{P}_s$ is the class of s-sparse polynomials, $\mathbb{P}_{d,s}$ is the class of degree-d s-sparse polynomials, $\mathbb{DP}_{d,s}$ is the class of degree-d s-sparse determinant-polynomials, $\mathbb{P}_{d,s}[2^t]$ is the class of degree-d s-sparse multilinear polynomials over $\mathrm{GF}(2^t)$ and "qu. in" is an abbreviation of "queries in".

$$\begin{array}{llll} & & \mathbb{DP}_{d,s} \text{ qu. in } \mathrm{GF}(2^t)^{dn} \to & \mathbb{P}_{d,s}[2^t] \text{ qu. in } \mathrm{GF}(2^t)^n \\ & & \uparrow & \\ \mathbb{P}_s \to & \mathbb{P}_{d,s} \to \mathbb{P}_{d,s}\ d\text{-mono.} \to & \mathbb{DP}_{d,s} \text{ qu. in } \mathrm{GF}(2)^{dn} & \end{array}$$

3 Definitions and Preliminary Results

We will denote by $\mathbb{P}_s$ the class of s-sparse polynomials over the Boolean variables $(x_1, \ldots, x_n)$ and $\mathbb{P}_{d,s} \subset \mathbb{P}_s$, the class of degree-$d$ s-sparse polynomials. For a power of two q, the classes $\mathbb{P}_s[q]$ is the class of s-sparse multilinear polynomials over the field GF(q) and $\mathbb{P}_{d,s}[q] \subset \mathbb{P}_s[q]$, the class of degree-$d$ s-sparse multilinear polynomials over GF(q).

Formally, let $\mathcal{S}_{n,\leq d} = \cup_{i \leq d} \mathcal{S}_{n,i}$, where $\mathcal{S}_{n,i} = \binom{[n]}{i}$ is the set of all i-subsets of $[n] = \{1, 2, \ldots, n\}$. The class $\mathbb{P}_{d,s}$ (resp. $\mathbb{P}_{d,s}[q]$) is the class of all the polynomials of the form

$$\sum_{I \in S} a_I \prod_{i \in I} x_i$$

where $S \subseteq \mathcal{S}_{n,\leq d}$, $|S| \leq s$ and $a_I = 1$ for all I (resp. $a_I \in \text{GF}(q) \backslash \{0\}$ for all I). The class $\mathbb{P}_s$ (resp. $\mathbb{P}_s[q]$) is $\mathbb{P}_{n,s}$ (resp. $\mathbb{P}_{n,s}[q]$).

Let $\{y_{j,i}\}_{i \in [n], j \in [d]}$ be nd variables. A degree-d s-sparse *determinant-polynomial* is a polynomial of the form

$$\sum_{I \in S} \det(\mathcal{Y}^{(I)}) \tag{1}$$

where $S \subseteq \mathcal{S}_{n,d}$, $|S| \leq s$ and for $I = \{i_1, \ldots, i_d\} \subseteq [n]$, $\mathcal{Y}^{(I)}$ is the $d \times d$ matrix where $\mathcal{Y}^{(I)}_{j,k} = y_{j,i_k}$. We denote by $\mathbb{DP}_{d,s}$ the class of degree-d s-sparse determinant-polynomials and $\mathbb{DP}_d = \cup_{s \geq 0} \mathbb{DP}_{d,s}$.

Consider the operator $\phi_d : \mathbb{P}_{d,s} \to \mathbb{DP}_{d,s}$ defined as[2]

$$\phi_d f = \sum_{J \subseteq [d]} f\left(\sum_{j \in J} y_j\right) \tag{2}$$

where $y_j = (y_{j,1}, \ldots, y_{j,n})$, $j \in [d]$. By Lemma 61 in [8], we have

$$\phi_d \sum_{I \in S} \prod_{i \in I} x_i = \sum_{I \in S, |I| = d} \det(\mathcal{Y}^{(I)}).$$

That is,

1. ϕ_d removes all the monomials of degree less than d from f.
2. It changes each monomial $\prod_{i \in I} x_i$ with $|I| = d$ to $\det(\mathcal{Y}^{(I)})$.
3. Given $\phi_d f$ represented as in (1), one can find the degree-d monomials of f in linear time.
4. Every black-box query to $\phi_d f$ can be simulated by 2^d black-box queries to f.

To see 3, notice that any row of $\mathcal{Y}^{(I)}$ uniquely gives I and therefore uniquely gives the monomial $\prod_{i \in I} x_i$.

We denote by $\mathbb{HP}_d$ the set of homogeneous polynomials F (over GF(2)) of degree d over the variables $Y = \{y_{i,j}\}_{i \in [n], j \in [d]}$ where each monomial of F is of the form $y_{1,i_1} y_{2,i_2} \cdots y_{d,i_d}$ where $\{i_1, i_2, \ldots, i_d\} \in \binom{[n]}{d}$. Obviously, $\mathbb{DP}_{d,s} \subset \mathbb{HP}_d$.

The following Lemma shows why we need the operator ϕ_d. Its proof is in Appendix A.

[2] In [8] the sum contains $(-1)^{d-|J|}$. This disappears here since in GF(2), $-1 = 1$.

▶ **Lemma 2.** *For any integer power of two $t > 1$ there is an algorithm that runs in time $n \cdot poly(t, 2^d)$ and finds $N = \tilde{O}(2^{1.66d})t$ polynomial-time computable maps $M_i : \mathrm{GF}(2^t)^{dn} \to \mathrm{GF}(2)^{dn}$, $i \in [N]$, and elements $\{\alpha_i\}_{i\in[N]}$ in $\mathrm{GF}(2^t)$ such that for every $\beta \in \mathrm{GF}(2^t)^{dn}$ and every $F \in \mathbb{HP}_d$*

$$F(\beta) = \sum_{i=1}^{N} \alpha_i F(M_i(\beta)).$$

In particular, if $F \in \mathbb{DP}_d$ then a black-box query in $\mathrm{GF}(2^t)^{dn}$ to F can be simulated in time $n \cdot poly(N)$ by N black-box queries in $\mathrm{GF}(2)^{dn}$ to F.

In this paper, the representation that is used for elements in the Galois field $\mathrm{GF}(2^t)$ is $\mathrm{GF}(2)[w]/\ (g(w))$ for some irreducible polynomial $g \in \mathrm{GF}(2)[w]$ of degree t.

4 The Learning Algorithm

In this section, we give the learning algorithm for $\mathbb{P}_s$. Recall the reductions from Section 2.

$$\begin{array}{llll} & & \mathbb{DP}_{d,s} \text{ qu. in } \mathrm{GF}(2^t)^{dn} \to & \mathbb{P}_{d,s}[2^t] \text{ qu. in } \mathrm{GF}(2^t)^n \\ & & \uparrow & \\ \mathbb{P}_s \to & \mathbb{P}_{d,s} \to \ \mathbb{P}_{d,s}\ d\text{-mono.} \ \to & \mathbb{DP}_{d,s} \text{ qu. in } \mathrm{GF}(2)^{dn} & \end{array}$$

4.1 The Reduction Algorithms

In this subsection, we give the reductions. We will prove a lemma for each reduction.

Let $f(x_1, \ldots, x_n)$ be any Boolean function. A *p-zero projection* of f is a random function, $f(z) = f(z_1, \ldots, z_n)$ where each z_i is equal to x_i with probability p and is equal to 0 with probability $1 - p$. The first reduction is Lemma 6 from [11]. The Lemma in [11] is stated for adaptive algorithms. The same proof holds for non-adaptive algorithms.

▶ **Lemma 3** ([11] ($\mathbb{P}_s \to \mathbb{P}_{d,s}$))**.** *Let $0 < p < 1$, $w = (s/\epsilon)^{\log(1/p)} \ln(16s)$ and*

$$D = \log \frac{s}{\epsilon} + \frac{\log s + \log\log s + 6}{\log(1/p)}.$$

Suppose there is a non-adaptive proper learning algorithm that exactly learns $\mathbb{P}_{d,s}$ with $Q(d, \delta)$ queries in time $T(d, \delta)$ and probability of success at least $1 - \delta$. Then there is a non-adaptive proper learning algorithm that learns $\mathbb{P}_s$ with $O(w \cdot Q(D, 1/(16w)) \cdot \log(1/\delta))$ queries, in time $w \cdot T(D, 1/(16w)) \log(1/\delta)$, probability of success at least $1 - \delta$ and accuracy $1 - \epsilon$.

We now give the second reduction.

▶ **Lemma 4** ($\mathbb{P}_{d,s} \to \mathbb{P}_{d,s}$ d-monomials)**.** *Suppose there is a non-adaptive algorithm that for $f \in \mathbb{P}_{d,s}$ exactly learns the monomials of degree d of f with $Q(d, \delta)$ queries in time $T(d, \delta)$ and probability of success at least $1 - \delta$. Then there is a non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}$ with $\sum_{j=0}^{d} Q(j, \delta/d)$ queries, time $O(d^2 s \max_j Q(j, \delta/d)) + \sum_{j=0}^{d} T(j, \delta/d)$ and probability of success at least $1 - \delta$.*

Proof. Let $f \in \mathbb{P}_{d,s}$. Let f_i be the sum of all the monomials of f of degree i, and s_i the number of monomials of f_i. Let $A(d, \delta)$ be a non-adaptive algorithm that exactly learns the monomials of degree d of $f \in \mathbb{P}_{d,s}$ with $Q(d, \delta)$ queries in time $T(d, \delta)$ and probability of success at least $1 - \delta$. We define an algorithm B as follows. First, algorithm B makes all

the queries that all $A(i, \delta/d)$, $i \leq d$, make. Let C_i be the set of queries that $A(i, \delta/d)$ makes, $i \leq d$. Then B continues to run $A(d, \delta/d)$ with the answers of the queries in C_d and learns the monomials of degree d of f. Let f_d be the sum of those monomials. Then B continues to run $A(d-1, \delta/d)$ with $\{(a, f(a)+f_d(a))|a \in C_{d-1}\}$. That is, for each query a of $A(d-1, \delta/d)$ we query f to find $f(a)$ and then return the answer $f(a) + f_d(a)$ to $A(d-1, \delta/d)$. Since $f + f_d$ is the sum of all the monomials of degree at most $d-1$ of f, $A(d-1, \delta/d)$ learns the monomials of degree $d-1$ of $f + f_d$, which are the monomials of degree $d-1$ of f. At the $d-i+1$-th stage, algorithm B continues to run $A(i, \delta/d)$ on

$$\left\{ \left(a, f(a) + \sum_{j=i+1}^{d} f_j(a) \right) \middle| a \in C_{d-1} \right\}$$

where f_j, $j \in \{i+1, i+2, \cdots, d\}$ is the sum of all the monomials of f of degree j. Since $g := f + \sum_{j=i+1}^{d} f_j$ is of degree i, $A(i, \delta/d)$ learns the monomials of degree i of g, which are the monomials of degree i of f. Notice that all the queries are all made for f, so the algorithm in non-adaptive.

It is clear that B exactly learns f, makes $\sum_{j=0}^{d} Q(j, \delta/d)$ queries, runs in time

$$O\left(\sum_{j=0}^{d} \left(\sum_{i=0}^{j} s_i \right) d \cdot Q(j, \delta/d) \right) + T(j, \delta/d) = O(d^2 s \max_j Q(j, \delta/d)) + \sum_{j=0}^{d} T(j, \delta/d),$$

and has probability of success at least $1 - \delta$. ◀

The following is the third reduction.

▶ **Lemma 5** ($\mathbb{P}_{d,s}$ d-monomials $\to$ $\mathbb{DP}_{d,s}$)**.** *Suppose there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ (with the matrix representation as in (1)) with $Q(d, \delta)$ queries in time $T(d, \delta)$ and probability of success at least $1 - \delta$. Then there is a non-adaptive exact learning algorithm that for $f \in \mathbb{P}_{d,s}$ learns the monomials of degree d of f with $2^d Q(d, \delta)$ queries in time $T(d, \delta) + O(2^d Q(d, \delta)n)$ and probability of success at least $1 - \delta$.*

Proof. Let $A(d, \delta)$ be a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $Q(d, \delta)$ queries in time $T(d, \delta)$ and probability of success at least $1 - \delta$. We define an algorithm $B(d, \delta)$ as follows. Define $F = \phi_d f \in \mathbb{DP}_{d,s}$ and run $A(d, \delta)$ to learn F. Recall that ϕ_d removes all the monomials of degree less than d from f and changes each monomial $\prod_{i \in I} x_i$ with $|I| = d$ on f to $\det(\mathcal{Y}^{(I)})$. By item 4 in Section 3, every black-box query to F can be simulated by 2^d black-box queries to f. Given F in its matrix representation as in (1), we can find the degree-d monomials of f. See item 3 in Section 3. ◀

The following is the fourth reduction.

▶ **Lemma 6** ($\mathbb{DP}_{d,s}$ queries in $\mathrm{GF}(2)^{dn}$ $\to$ $\mathbb{DP}_{d,s}$ queries in $\mathrm{GF}(2^t)^{dn}$)**.** *Let t be a power of two. Suppose there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $Q(d, \delta)$ black-box queries in $\mathrm{GF}(2^t)^{dn}$, time $T(d, \delta)$ and probability of success at least $1 - \delta$. Then there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $2^{1.66d} t Q(d, \delta)$ black-box queries in $\mathrm{GF}(2)^{dn}$, time $T(d, \delta) + poly(t, 2^d) Q(d, \delta)n$ and probability of success at least $1 - \delta$.*

Proof. The result follows from Lemma 2. ◀

The following is the fifth reduction.

▶ **Lemma 7** ($\mathbb{DP}_{d,s}$ queries in $\mathrm{GF}(2^t)^{dn} \longrightarrow \mathbb{P}_{d,s}[2^t]$ queries in $\mathrm{GF}(2^t)^n$). *Let $\ell = \lceil \log(2s/\delta)/(t-1) \rceil$. Suppose there is a non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $Q(d,\delta)$ queries in $\mathrm{GF}(2^t)^n$ in time $T(d,\delta)$ and probability of success at least $1-\delta$. Then there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with $\ell \cdot Q(d,\delta/(2\ell))$ queries in $\mathrm{GF}(2^t)^{dn}$ in time $O(\ell dnt) + \ell T(d,\delta/(2\ell))$ and probability of success at least $1-\delta$.*

Proof. Let $A(d,\delta)$ be a non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. Let $F(y_1, y_2, \ldots, y_d) \in \mathbb{DP}_{d,s}$ where $y_j = (y_{j,1}, \ldots, y_{j,n})$, $j \in [d]$. We define an algorithm $B(d,\delta)$ as follows. Algorithm B uniformly at random chooses $dn\ell$ elements $\alpha_{j,i}^{(k)} \in \mathrm{GF}(2^t)$, $j \in [d]$, $i \in [n]$, and $k \in [\ell]$. Let $z_j^{(k)} = (\alpha_{j,1}^{(k)} x_1, \ldots, \alpha_{j,n}^{(k)} x_n)$, $j \in [d]$, $k \in [\ell]$ and $g^{(k)}(x_1, \ldots, x_n) = F(z_1^{(k)}, z_2^{(k)}, \ldots, z_d^{(k)})$. If $F = \sum_{I \in S} \det(\mathcal{Y}^I)$, then

$$g^{(k)}(x_1, \ldots, x_n) = \sum_{I \in S} \left(\left(\det \Gamma^{(k,I)} \right) \prod_{i \in I} x_i \right)$$

where for $I = \{i_1, \ldots, t_d\}$,

$$\Gamma^{(k,I)} = \begin{bmatrix} \alpha_{1,i_1}^{(k)} & \cdots & \alpha_{1,i_d}^{(k)} \\ \vdots & \ddots & \vdots \\ \alpha_{d,i_1}^{(k)} & \cdots & \alpha_{d,i_d}^{(k)} \end{bmatrix}.$$

Therefore, $\prod_{i \in I} x_i$ is a monomial of $g^{(k)}$ if and only if

1. $\det(\mathcal{Y}^I)$ is a term of F, and
2. $\det \Gamma^{(k,I)} \neq 0$.

Now notice that each $g^{(k)}$ is an s-sparse polynomial. Therefore, if we learn the monomials of $g^{(k)}$, $k \in [\ell]$, then we learn the terms $\det(\mathcal{Y}^I)$ of F for which $\det \Gamma^{(k,I)} \neq 0$. So, algorithm B runs $A(d,\delta/(2\ell))$ to learn all $g^{(k)}$, and from the monomials of all $g^{(k)}$ finds the terms of F.

The probability that B fails to learn all the monomials of F is equal to the probability that for some term $\det(\mathcal{Y}^I)$ of F, there is no $k \in [\ell]$ such that $\det \Gamma^{(I,k)} \neq 0$. Since $\alpha_{j,i}^{(k)} \in \mathrm{GF}(2^t)$ are uniformly random, this probability is at most

$$\begin{aligned} s\left(1 - \left(1 - \frac{1}{2^{dt}}\right)\left(1 - \frac{1}{2^{(d-1)t}}\right) \cdots \left(1 - \frac{1}{2^t}\right)\right)^\ell &\leq s\left(\frac{1}{2^{dt}} + \frac{1}{2^{(d-1)t}} + \cdots + \frac{1}{2^t}\right)^\ell \\ &\leq \frac{s}{2^{(t-1)\ell}} \leq \frac{\delta}{2}. \end{aligned}$$

The probability that $A(d,\delta/(2\ell))$ fails to learn all $g^{(k)}$ is at most $\ell(\delta/(2\ell)) = \delta/2$. This completes the proof. ◀

Now we show

▶ **Lemma 8** ($\mathbb{P}_{d,s}[2^t]$ queries in $\mathrm{GF}(2^t)^n$). *Let $t = d'm$ be an integer where $d < d' = O(d)$ and $\log n < m = O(\log n)$. There is a deterministic non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $2s$ queries in $\mathrm{GF}(2^t)^n$ in time $\mathrm{poly}(d,s) \cdot \tilde{O}(n)$.*

Proof. The result follows from the BCH decoding and the algorithm of Ben-Or and Tiwari [3], with a small modification. See the details in the Appendix B. ◀

4.2 Query and Time Complexity

In this section, we prove,

▶ **Theorem 9.** *Let $\beta > 0$ be any real number. There is a non-adaptive proper learning algorithm for s-sparse polynomial with accuracy parameters $\epsilon = 1/s^\beta$ and confidence parameter δ that makes*

$$q = O\left(\left(\frac{s}{\epsilon}\right)^{2.66+\frac{1}{\beta+1}+\frac{3.262}{\sqrt{\beta+1}}} \log n \log(1/\delta)\right) = O\left(\left(\frac{s}{\epsilon}\right)^{6.922} \log n \log(1/\delta)\right)$$

queries and runs in time $\tilde{O}(n) \cdot poly(s, 1/\epsilon) \log(1/\delta)$.

Proof. Denote by $\lceil x \rceil_2$ the smallest power-of-two integer that is greater than or equal to x. Notice that $x \leq \lceil x \rceil_2 < 2x$.

We choose $t = \lceil D+1 \rceil_2 \lceil \log n \rceil_2 = O(D \log n)$, $\tau := \log(1/p)$,

$$D = \log \frac{s}{\epsilon} + \frac{\log s + \log\log s + 6}{\tau}, \text{ and } w = \left(\frac{s}{\epsilon}\right)^\tau \ln(16s),$$

where p will be determined later. We only need to know here that for this choice of p, we will have $D = \Theta(\log(s/\epsilon))$. By Lemma 8, there is a deterministic non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{D,s}[2^t]$ with $Q_1(D,\delta) = 2s$ queries over $\mathrm{GF}(2^t)$ in time $T_1(D,\delta) = poly(D,s)\tilde{O}(n)$. By Lemma 7, there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{D,s}$ with $Q_2(D,\delta) = \tilde{O}(s)\log(1/\delta)$ queries over $\mathrm{GF}(2^t)$ in time $T_2(D,\delta) = poly(D,s)\tilde{O}(n)\log(1/\delta)$ and probability of success at least $1-\delta$. By Lemma 6, there is a non-adaptive proper exact learning algorithm that learns $\mathbb{DP}_{d,s}$ with

$$Q_3(D,\delta) = 2^{1.66D} t Q_2(D,\delta) = \tilde{O}\left(\frac{s^{2.66+\frac{1.66}{\tau}+o_s(1)}}{\epsilon^{1.66}} \log n \log(1/\delta)\right)$$

queries in time

$$T_3(D,\delta) = T_2(D,\delta) + poly(t, 2^D) Q_2(d,\delta) n = \tilde{O}(n) poly(s, 1/\epsilon) \log(1/\delta)$$

and probability of success at least $1-\delta$. By Lemma 5 and 4, there is a non-adaptive exact learning algorithm that learns $\mathbb{P}_{D,s}$ with

$$Q_4(D,\delta) = D2^D Q_3(D, \delta/D) = \tilde{O}\left(\frac{s^{3.66+\frac{2.66}{\tau}+o_s(1)}}{\epsilon^{2.66}} \log n \log(1/\delta)\right)$$

queries in time $T_4(D,\delta) = \tilde{O}(n) \cdot poly(s, 1/\epsilon)\log(1/\delta)$ and probability of success at least $1-\delta$. Now, by Lemma 3, there is a non-adaptive proper learning algorithm that learns $\mathbb{P}_s$ with

$$Q(D,\delta) = O(w \cdot Q_4(D, 1/(16w))\log(1/\delta)) = \tilde{O}\left(\frac{s^{3.66+\tau+\frac{2.66}{\tau}+o_s(1)}}{\epsilon^{2.66+\tau}} \log n \log(1/\delta)\right)$$

queries in time $T(D,\delta) = w \cdot T_4(D, 1/(16w))\log(1/\delta) = \tilde{O}(n) \cdot poly(s, 1/\epsilon)\log(1/\delta)$, probability of success at least $1-\delta$ and accuracy $1-\epsilon$.

For $\epsilon = 1/s^\beta$ we choose[3] $\tau = \sqrt{2.66/(1+\beta)}$ and get

$$Q(D,\delta) = O\left(\left(\frac{s}{\epsilon}\right)^{2.66+\frac{1}{\beta+1}+\frac{3.262}{\sqrt{\beta+1}}} \log n \log(1/\delta)\right). \qquad \blacktriangleleft$$

[3] This choice of τ minimizes the query complexity of the algorithm.

5 The Learning Algorithm for Small ϵ

In this section, we give a more query-efficient learning algorithm for s-sparse polynomials when $\epsilon < 1/s^{0.752 \log s}$. We prove

▶ **Theorem 10.** *Let $\beta > 0$ be any real number. There is a non-adaptive proper learning algorithm for s-sparse polynomials with accuracy parameters $\epsilon = 1/s^{\beta}$ and confidence parameter δ that makes*

$$q = \left(s \log \frac{1}{\epsilon}\right)^{2 \log s} s^{O(\log \log s)} \log n \log(1/\delta) = \left(\frac{s}{\epsilon}\right)^{\frac{2 \log s + 2 \log \beta + O(\log \log s)}{\beta + 1}} \log n \log(1/\delta)$$

queries and runs in time $\tilde{O}(qsn)$.

5.1 Technique

We will use the following result from [17]. See also [13].

▶ **Lemma 11.** *Let $U_{n,s}$ be the set of all the assignments in $\{0,1\}^n$ of Hamming weight at least $n - \lfloor \log s \rfloor - 1$. There is a non-adaptive exact learning algorithm for $\mathbb{P}_s$ that makes the black-box queries in $U_{n,s}$ and runs in time $sn(en/\log s)^{\log s + 1}$.*

In particular,

1. *Let $f, g \in \mathbb{P}_s$ be two distinct s-sparse polynomials. There is $a \in U_{n,s}$ such that $f(a) \neq g(a)$.*
2. *If $f \in \mathbb{P}_s$ depends on the variable x_i then there are two assignments $a, b \in U_{n,s}$ that differ only in the ith coordinate and $f(a) \neq f(b)$.*

The above lemma follows from the fact that if f is of size $s > 1$, then there is $i \in [n]$ such that $f = f_1 x_i + f_2$ and $f_2 \not\equiv 0$. Then either[4] $f_2 = f_{x_i \leftarrow 0}$ is of size $\lfloor s/2 \rfloor$, or $f_1 = f_{x_i \leftarrow 0} + f_{x_i \leftarrow 1}$ is of size $\lfloor s/2 \rfloor$. See [17].

Item 2 follows from applying item 1 to $f(x)$ and $g(x) = f(1, x_2, \ldots, x_n)$.

In the first stage of our algorithm, we set each variable to 0 with probability $1 - 1/O(\log(s/\epsilon))$. This assignment removes monomials of size $D > \Omega((\log s)(\log(s/\epsilon))$. By Lemma 3, to be able to collect all the monomials of size at most $\log(s/\epsilon)$ of f, it is enough to take $O(\log s)$ such assignments. This reduced the non-adaptive learning of s-sparse polynomials to non-adaptive learning degree-d s-sparse polynomials, where $d = O((\log s)(\log s/\epsilon))$.

Now, let g be a degree-d s-sparse polynomial where $d = O((\log s)(\log s/\epsilon))$. The function $g(x)$ depends on at most $v = sd = O(s \log s \log(s/\epsilon))$ variables. We, uniformly at random, assign the n variables of g into $w = O(v^2)$ new variables $Y = \{y_1, y_2, \ldots, y_w\}$. Let $h(y)$ be the resulted function. With high probability, different relevant variables in g are assigned to different variables in Y. Now, assuming this event occurs, we run two algorithms. The first one learns $h(y)$ by the algorithm in Lemma 11. This takes $|U_{w,s}| \leq (ew/\log s)^{\log s} = (s \log(1/\epsilon))^{O(\log s)}$ queries. The second algorithm uses item 2 in Lemma 11 to find the relevant variable in $g(x)$ corresponding to each y_i (if any). To achieve that, for every two assignments in $U_{w,s}$ that differ in one coordinate, say i, we non-adaptively search for the relevant variable among all the variables that are assigned to y_i. Each search takes $O(\log n)$. This algorithm takes $w|U_{w,s}| \log n = (s \log(1/\epsilon))^{O(\log s)} \log n$ queries. This proves the Theorem.

[4] $f_{x_i \leftarrow 0}$ is f when we substitute 0 in x_i.

5.2 The Algorithm

In this section, we give the algorithm and prove its correctness.

By Lemma 3 with $\log(1/p) = 1/\log(s/\epsilon)$, we have

▶ **Lemma 12** ($\mathbb{P}_s \to \mathbb{P}_{d,s}$)**.** *Let $w = 2\ln(16s)$. Suppose there is a non-adaptive proper learning algorithm that exactly learns $\mathbb{P}_{d,s}$ with $Q(d,\delta)$ queries in time $T(d,\delta)$ and probability of success at least $1-\delta$. Then there is a non-adaptive proper learning algorithm that learns $\mathbb{P}_s$ with $O(w \cdot Q(D, 1/(16w))\log(1/\delta))$ queries where*

$$D = \left(\log \frac{s}{\epsilon}\right)(\log s + \log\log s + 7),$$

in time $w \cdot T(D, 1/(16w))\log(1/\delta)$, probability of success at least $1-\delta$ and accuracy $1-\epsilon$.

The following is trivial:

▶ **Lemma 13.** *There is a non-adaptive exact proper learning algorithm for $C = \{0, 1, x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ that makes $\log n + O(1)$ queries and runs in time $O(n \log n)$.*

We now prove

▶ **Lemma 14.** *There is a proper exact learning algorithm for $\mathbb{P}_{d,s}$ with probability of success at least $1-\delta$ that makes $q = O(d(2e(ds)^2/\log s)^{\log s}\log n \log(1/\delta))$ queries and runs in time $O(qn)$.*

Proof. The algorithm draws uniformly at random a map $\phi : [n] \to [m]$ where $m = (2ds)^2$, and defines $F(x_1, \ldots, x_m) = f(x_{\phi(1)}, \ldots, x_{\phi(n)})$ and then exactly learns F using the algorithm in Lemma 11. Then, for every $i \in [m]$ and every $a, b \in U_{m,s}$ that differ in the ith coordinate, it learns, using the algorithm in Lemma 13, $f(\pi^{(i,a)})$ where

$$\pi_j^{(i,a)} = \begin{cases} a_{\phi(j)} & \phi(j) \neq i \\ x_j & \phi(j) = i \end{cases}.$$

Then the algorithm returns $F(y_1, \ldots, y_m)$ where $y_i = x_j$ if there is $a \in U_{m,s}$ such that $f(\pi^{(i,a)}) \in \{x_j, \bar{x}_j\}$ and $y_i = 0$ otherwise.

We now prove the correctness of the algorithm. Let $x_{r_1}, \ldots, x_{r_\ell}$, $\ell \leq ds$, be the relevant variables of f. The probability that $\phi(r_1), \ldots, \phi(r_\ell)$ are distinct is at least $(1-1/m)(1-2/m)\cdots(1-(\ell-1)/m) \geq 7/8$. We now assume that this event occurs. In particular, $x_{\phi(r_1)}, \ldots, x_{\phi(r_\ell)}$ are the relevant variables of F.

Let $\phi(r_i) = t_i$. Let $a, b \in U_{m,s}$ be two assignments that differ in the t_i-th coordinate and $F(a) \neq F(b)$. Then $f(\pi^{(t_i,a)})$ is a non-constant function, and since $\phi(r_i) = t_i$, we have $f(\pi^{(t_i,a)}) \in \{x_{r_i}, \bar{x}_{r_i}\}$. Therefore, $y_{\phi(r_i)} = y_{t_i} = x_{r_i}$ and $F(y_1, \ldots, y_m) = f(y_{\phi(1)}, \ldots, y_{\phi(n)}) = f$.

The query complexity of the algorithm is $|U_{m,s}| = (em/\log s)^{\log s}$ for learning F and $d|U_{m,s}| = d(em/\log s)^{\log s}\log n$ for learning all $f(\pi^{(i,a)})$. This algorithm has a success probability of at least $7/8$. Repeating the algorithm $O(\log(1/\delta))$ times gives the result. ◀

We are now ready to prove Theorem 10.

Proof. We first run the algorithm in Lemma 12. Then for each projection runs the algorithm in Lemma 14 with $d = D$. This gives the query complexity in the Theorem. ◀

6 Upper and Lower Bounds

In this section, we prove lower and upper bounds for learning sparse polynomials with unlimited computational power.

As we have seen in the previous sections, for constant confidence δ, the query complexity of the learning algorithms for $\epsilon = 1/s^\beta$ is of the form $(s/\epsilon)^{\gamma'} \log n$. In this section, we will study learning algorithms with query complexity

$$\left(\frac{s}{\epsilon}\right)^{\gamma(\beta)} \log n.$$

We denote by $\Gamma(\beta) = \min_A \gamma(\beta)$ the minimal possible value of $\gamma(\beta)$ among all the learning algorithms A of s-sparse polynomials with unlimited computational power. Formally,

$$\Gamma(\beta) = \min_A \lim_{n\to\infty} \frac{\log(q(A)/\log n)}{\log(s/\epsilon)}.$$

In particular, we may assume that s and $1/\epsilon$ are arbitrary small compared to n.

By Theorem 9 and 10, we already know the following bounds

$$\Gamma(\beta) \le \begin{cases} 2.66 + \frac{3.262}{\sqrt{\beta+1}} + \frac{1}{\beta+1} & \beta < 0.752 \log s \\ \frac{2\log s + 2\log\beta + O(\log\log s)}{\beta+1} & \beta \ge 0.752 \log s \end{cases}$$

In this section, we prove the upper bound $\Gamma(\beta) \le 1 + \min(1,\beta)/(\beta+1)$. The above bounds gives the upper bound

$$\Gamma(\beta) \le \begin{cases} 1 + \frac{\beta}{\beta+1} & \beta < 1 \\ 1 + \frac{1}{\beta+1} & 1 \le \beta < 4.923 \\ \frac{\log\beta}{\beta+1} + \Theta\left(\frac{1}{\beta}\right) & \beta \ge 4.923 \end{cases}. \tag{3}$$

The exact bound when $\beta \ge 4.923$ is

$$\Gamma(\beta) \le \min_{0.801<\eta<0.847} \frac{\eta+1}{\beta+1} + \left(1+\eta^{-1}\right) H_2\left(\frac{1}{(\beta+1)(1+\eta^{-1})}\right),$$

where $H_2(x) = -x\log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function. In particular, $\Gamma(\beta) \le 1.5$ for all β and $\Gamma(\beta) \to 0$ as $\beta \to 0$.

We then prove the lower bound,

$$\Gamma(\beta) \ge \begin{cases} \frac{1}{\beta+1} & 0 < \beta < 0.441 \\ 0.694 & 0.441 \le \beta < 2.548 \\ \frac{\log\beta}{\beta+1} + \Theta\left(\frac{1}{\beta}\right) & \beta \ge 2.548 \end{cases}. \tag{4}$$

The exact bound when $\beta \ge 2.548$ is

$$\Gamma(\beta) \ge \frac{\beta \cdot H_2(1/\beta)}{\beta+1}.$$

Notice that our first algorithm in Section 4, for $\epsilon = 1/s^{O(1)}$, makes $(s/\epsilon)^{\gamma} \log n$ queries where $2.66 \leq \gamma \leq 6.922$ and the lower bound for this case is $(s/\epsilon)^{1/(\beta+1)} \log n = (s/\epsilon)^{\Theta(1)} \log n$. Therefore, our first algorithm is polynomial in the optimal query complexity and optimal in n.

The second algorithm in Section 5, for $\epsilon = 1/s^{\omega(\log s)}$, makes $(s/\epsilon)^{O((\log s + \log \beta)/\beta)} \log n = (s/\epsilon)^{o_\beta(1)} \log n$ queries and the lower bound is $(s/\epsilon)^{\beta H_2(1/\beta)/(\beta+1)} \log n = (s/\epsilon)^{\Theta(\log \beta/\beta)} \log n$. Therefore, the query complexity of the second algorithm is polynomial in the optimal query complexity when $\beta = s^{\Omega(1)}$.

All the upper bounds are in the full paper. We next give the lower bounds.

6.1 Lower Bounds

In this section, we give three lower bounds that prove the lower bound in (4). We remind the reader that the parameters s and $1/\epsilon$ are arbitrary small compared to n.

We first give the following information-theoretic lower bound.

▶ **Theorem 15.** *Any non-adaptive algorithm for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ must make at least*

$$\Omega\left(\left(\log \frac{1}{\epsilon}\right) s \log n\right)$$

queries. In particular, when $\epsilon = 1/s^{\beta}$, the bound is

$$\tilde{\Omega}\left(\left(\frac{s}{\epsilon}\right)^{\frac{1}{\beta+1}}\right) \log n \text{ and } \Gamma(\beta) \geq \frac{1}{\beta+1}.$$

Proof. Consider the class $C = \mathbb{P}_{\log(1/(2\epsilon)),s}$. Consider a (randomized) non-adaptive learning algorithm A_R for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ and accuracy ϵ. Then A_R is also a (randomized) non-adaptive learning algorithm for C. Since any two distinct functions in C have distance 2ϵ, A_R exactly learns[5] C with a confidence probability of at least $2/3$. By Yao's minimax principle, there is a deterministic non-adaptive exact learning algorithm A_D with the same query complexity as A_R that learns at least $(2/3)|C|$ functions in $|C|$. By the information theoretic lower bound, the query complexity of A_D is at least $\log |C| - 2$. Since

$$\log |C| = \log \binom{n}{s}_{\log(1/(2\epsilon))} = \Omega\left(\left(\log \frac{1}{\epsilon}\right) s \log n\right),$$

the result follows. ◀

We now show the following.

▶ **Theorem 16.** *Let $\epsilon = 1/s^{\beta}$, $\beta > 2$. Any non-adaptive algorithm for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ must make at least*

$$\Omega\left(\left(\frac{s}{\epsilon}\right)^{\frac{\beta \cdot H_2(1/\beta)}{\beta+1}} \log n\right)$$

queries.

In particular, for $\beta > 2$,

$$\Gamma(\beta) \geq \frac{\beta \cdot H_2(1/\beta)}{\beta+1} \geq \frac{\log \beta}{\beta+1}.$$

[5] Just take the function in C closest to the output of A_R.

Proof. Let $t = \log(1/\epsilon) - \log s - 2 \geq \log s - 2$ and $r = \log s$. Let W be the set of all pairs (I, J) where I and J are disjoint sets, $I \cup J = [t + r]$, $|I| \geq t$, and $|J| = t + r - |I| \leq r$. For every $(I, J) \in W$, define $f_{I,J} = \prod_{i\in I} x_i \prod_{j\in J}(1 + x_j)$. For $k \in [n]\backslash[t + r]$ define $f_{I,J,k} = x_k \cdot f_{I,J}$. Consider the set C of all such functions. First notice that $f_{I,J,k} \in C \subset \mathbb{P}_{t+r+1,s} \subseteq \mathbb{P}_s$ and $\mathbf{Pr}[f_{I,J,k} = 1] \geq 2^{-(t+r+1)} = 2^{-\log(1/\epsilon)+1} = 2\epsilon$. Furthermore, since for $(I_1, J_1, k_1) \neq (I_2, J_2, k_2)$ the degree of $f_{I_1,J_1,k_1} + f_{I_2,J_2,k_2}$ is $\log(1/\epsilon) - 1$, we also have $\mathbf{Pr}[f_{I_1,J_1,k_1} \neq f_{I_2,J_2,k_2}] \geq 2\epsilon$. Therefore, any learning algorithm for $\mathbb{P}_s$ (with accuracy ϵ and confidence $2/3$) is a learning algorithm for C and thus is an exact learning algorithm for C.

Consider now a (randomized) non-adaptive exact learning algorithm A_R for C with a success probability of at least $2/3$ and accuracy ϵ. By Yao's minimax principle, there is a deterministic non-adaptive exact learning algorithm A_D that for uniformly at random $f \in C$, with a probability at least $2/3$, A_D returns f. We will show that A_D must make more than $q = (1/20)w \log N$ queries where $N = n - (t + r)$ and $w = |W| = \sum_{i=0}^{r} \binom{t+r}{i}$. Now since, $r \leq (t + r)/2 + 1$,

$$w = \sum_{i=0}^{\log s} \binom{\log \frac{1}{\epsilon} - 2}{i} \geq \tilde{\Omega}\left(2^{H_2\left(\frac{\log s}{\log(1/\epsilon)}\right)\log(1/\epsilon)}\right) = \tilde{\Omega}\left(\left(\frac{1}{\epsilon}\right)^{H_2(1/\beta)}\right) = \tilde{\Omega}\left(\left(\frac{s}{\epsilon}\right)^{\frac{\beta\cdot H_2(1/\beta)}{\beta+1}}\right)$$

we get the result.

To this end, suppose for the contrary, A_D makes q queries. Let $S = \{a^{(1)}, \ldots, a^{(q)}\}$ be the set of queries that A_D makes. For every $(I, J) \in W$, let $S_{I,J} = \{a \in S | f_{I,J}(a) = 1\}$. Since for any two distinct $(I_1, J_1), (I_2, J_2) \in W$, we have $f_{I_1,J_1} \cdot f_{I_2,J_2} = 0$, the sets $\{S_{I,J}\}_{(I,J)\in W}$ are disjoint. Let $f = f_{I',J',k'}$ be uniformly at random function in C. We will show that, with probability at least $4/5$, A_D fails to learn f, which gives a contradiction.

Since

$$E_{(I,J)\in W}[|S_{I,J}|] = \frac{\sum_{(I,J)\in W} |S_{I,J}|}{|W|} = \frac{q}{w} = (1/20)\log N,$$

by Markov's bound with probability at least $9/10$, we have $|S_{I',J'}| \leq (1/2)\log N$. Since for $(I, J) \neq (I', J')$, $f_{(I',J',k')}(S_{I,J}) = \{0\}$, the only queries that are relevant for learning k' in f are the ones in $S_{I',J'}$. Therefore, it is enough to show that, if $|S_{I',J'}| \leq (1/2)\log N$, then with probability at least $9/10$, the queries in $S_{I',J'}$ fail to learn k'. Since the algorithm is deterministic and the number of possible answers to the queries in $S_{I',J'}$ is $2^{(1/2)\log N} = \sqrt{N}$, the number of possible distinct outputs of the algorithm is at most $\sqrt{N}$. Since k' is drawn uniformly at random and can take N possible values, the probability that the algorithm returns k' is less than or equal to $\sqrt{N}/N = 1/\sqrt{N} \leq 1/10$. Therefore, the algorithm fails to output k' with probability at least $9/10$. This completes the proof. ◀

We now show

▶ **Theorem 17.** *Let $\epsilon = 1/s^\beta$, $0.44 \leq \beta \leq 2.61$. Any non-adaptive algorithm for $\mathbb{P}_s$ with a confidence probability of at least $2/3$ must make at least*

$$\Omega\left(\left(\frac{s}{\epsilon}\right)^{0.694} \log n\right)$$

queries.

Proof. Let η be such that $\frac{1+\beta\eta}{1+\beta} = \frac{5+\sqrt{5}}{10} \approx 0.7236$. Then $0.0955 \leq \eta \leq 0.618$. Also

$$\frac{1 + \beta\eta}{\beta(1 - \eta)} = \frac{1}{(1 + \beta)/(1 + \beta\eta) - 1} = \frac{3 + \sqrt{5}}{2} \approx 2.618. \tag{5}$$

Let

$$t = \log s + (2\eta - 1)\log(1/\epsilon) + 3 = (1 + \beta(2\eta - 1))\log s + 3 \tag{6}$$

and $r = (1-\eta)\log(1/\epsilon) - 3 = \beta(1-\eta)\log s - 3$. We first show that $t \geq 3$. By 5, $1+\beta\eta > \beta(1-\eta)$ and therefore $(1 + \beta(2\eta - 1)) > 0$. Thus, by 6, $t \geq 3$.

Let $\sigma = 8\epsilon^{1-\eta}s = 8s^{1-\beta(1-\eta)}$. Since, by 5, $\beta(1-\eta) = (1+\beta\eta)/2.618 \leq (1 + 2.61 \cdot 0.618)/2.618 < 1$, we have $\sigma \geq 8$.

Let W be the set of all r-sets $J \subseteq I := [t + r]$. For every $J \in W$, define $f_J = \prod_{i\in I\setminus J} x_i \prod_{j\in J}(1 + x_j)$. Let U be the set of all σ-sets $\mathcal{J} = \{(J_1, t_1), \ldots, (J_\sigma, t_\sigma)\}$ where $J_1, \ldots, J_\sigma \subseteq I$ are distinct r-sets and $t_k \in [n]\setminus[t + r]$. Let

$$f_{\mathcal{J}}(x) = \sum_{k=1}^{\sigma} x_{t_k} f_{J_k} = \sum_{k=1}^{\sigma} \left(x_{t_k} \prod_{i\in I\setminus J_k} x_i \prod_{i\in J_k} (1 + x_i) \right).$$

We first show that $f_{\mathcal{J}}$ is well-defined. That is, it is possible to choose σ distinct r-sets $J_1, \ldots, J_\sigma \subset I$. This is possible because

$$\binom{t+r}{r} = \binom{(1+\beta\eta)\log s}{\beta(1-\eta)\log s - 3} = s^{(1+\beta\eta)H\left(\frac{\beta(1-\eta)}{1+\beta\eta}\right) - o(1)} = s^{0.9594(1+\beta\eta) - o(1)} \tag{7}$$

and $\sigma = 8s^{1-\beta(1-\eta)} = 8s^{1-0.382(1+\beta\eta)} < s^{0.626(1+\beta\eta)} < s^{0.9594(1+\beta\eta)-o(1)} = \binom{t+r}{r}$.

Consider the class $C = \{f_{\mathcal{J}} | \mathcal{J} \in U\}$. The number of monomials of $f_{\mathcal{J}}(x)$ is $\sigma 2^r = s$ and therefore $C \subseteq \mathbb{P}_s$. Since the terms of $f_{\mathcal{J}}$ are disjoint, we have $\mathbf{Pr}[f_{\mathcal{J}} = 1] = \sigma 2^{-(t+r)-1} = 4\epsilon^{1-\eta}s \cdot (\epsilon^\eta/s) = 4\epsilon$. Notice that the distance between every two functions in C can be as small as $2 \cdot 2^{-(t+r)} = 2\epsilon^\eta/s = 2\epsilon^{\eta+\beta}$, which may take values less than ϵ. Therefore, the argument used in Theorem 16 will not hold here.

Consider now a (randomized) non-adaptive learning algorithm A_R for C with a success probability of at least $2/3$ and accuracy ϵ. By Yao's minimax principle, there is a deterministic non-adaptive learning algorithm A_D that for a uniformly at random target function $f_{\mathcal{J}} \in C$, with probability at least $2/3$, A_D returns a hypothesis $h^{(\mathcal{J})}$ that is ϵ-close to $f_{\mathcal{J}}$. Consider the deterministic algorithm A'_D that runs A_D and outputs $\mathcal{J}'$ where $f_{\mathcal{J}'}$ is the closest function in C to $h^{(\mathcal{J})}$. Since $h^{(\mathcal{J})}$ is ϵ-close to both $f_{\mathcal{J}}$ and $f_{\mathcal{J}'}$, $f_{\mathcal{J}}$ and $f_{\mathcal{J}'}$ are 2ϵ-close. We will show in the sequel that if $f_{\mathcal{J}}$ and $f_{\mathcal{J}'}$ are 2ϵ-close, then $|\mathcal{J} \cap \mathcal{J}'| \geq \sigma/2$. This shows that A'_D returns a σ-set $\mathcal{J}'$ that, with probability at least $2/3$, $|\mathcal{J} \cap \mathcal{J}'| \geq \sigma/2$ where $f_{\mathcal{J}}$ is the target function. We now show that if A'_D makes less than $q = (1/100)w \log N$ queries where $w = \binom{t+r}{r}$ and $N = n - (s + t)$ then, with probability at least $2/3$, algorithm A'_D fails to output such $\mathcal{J}'$. Now, since, by (7),

$$\binom{t+r}{r} = s^{0.9594(1+\beta\eta)-o(1)} = \left(\frac{s}{\epsilon}\right)^{0.9594\frac{1+\beta\eta}{\beta+1}-o(1)} = \left(\frac{s}{\epsilon}\right)^{0.6942-o(1)},$$

the result follows.

To this end, suppose A'_D makes less than q queries. Let $S = \{a^{(1)}, \ldots, a^{(q)}\}$ be the queries that A'_D makes. For every $J \in W$, let $S_J = \{a \in S | f_J(a) = 1\}$. Since for any two distinct $J_1, J_2 \in W$, we have $f_{J_1} f_{J_2} = 0$, the sets $\{S_J\}_{J\in W}$ are disjoint. Since

$$E_{J\in W}[|S_J|] = \frac{\sum_{J\in W} |S_J|}{|W|} = \frac{q}{w} = \frac{\log N}{100},$$

by Markov's bound, at least $49/50$ fraction of the r-subsets J of $[t+r]$ satisfy $|S_J| \leq (1/2)\log n$. Now, for a uniformly at random target function $f_{\mathcal{J}} = f_{\{(J_1,t_1),\ldots,(J_\sigma,t_\sigma)\}} \in C$, let X_i be an indicator random variable that is equal to 1 if $|S_{J_i}| > (1/2)\log N$. Then $\mathbf{E}[X_i] \leq 1/50$. Let

$X = X_1 + \cdots + X_\sigma$. Since $\mathbf{E}[X] \leq \sigma/50$, by Markov's bound, with probability at least $4/5$, at least $9\sigma/10$ of the J_is in $\mathcal{J}$ satisfy $|S_{J_i}| < (1/2)\log N$. Assume, wlog, $|S_{J_i}| < (1/2)\log N$ for $i \leq 9\sigma/10$. As in Theorem 16, the algorithm A'_D can find each t_i, $i \leq 9\sigma/10$ with probability at most $1/\sqrt{N} < 1/20$. Since the t_is are drawn iid and uniformly at random, by Chernoff bound, with probability at least $4/5$, the algorithm A'_D fails to find $\sigma/2$ of the t_is for $i \leq 9\sigma/10$. Therefore, with probability at least $2/3$, A'_D fails to return a σ-set $\mathcal{J}'$ such that $|\mathcal{J} \cap \mathcal{J}'| \geq \sigma/2$.

It remains to show that if $f_{\mathcal{J}}$ is 2ϵ-close to $f_{\mathcal{J}'}$, then $|\mathcal{J} \cap \mathcal{J}'| \geq \sigma/2$. Suppose, for the contrary, $|\mathcal{J} \cap \mathcal{J}'| < \sigma/2$. Suppose, wlog, $\mathcal{J} = \{(J_1, t_1), \ldots, (J_\sigma, t_\sigma)\}$ and

$$\mathcal{J}' = \{(J_1, t_1), \ldots, (J_\ell, t_\ell), (J_{\ell+1}, t'_{\ell+1}), \ldots, (J_\mu, t'_\mu), (J'_{\mu+1}, t'_{\mu+1}), \ldots, (J'_\sigma, t'_\sigma)\}$$

where J_i, J'_k, $i \in [\sigma]$, $k \geq \mu+1$ are distinct r-subsets of $[r+t]$ and $t'_i \neq t_i$ for $\mu \geq i > \ell$. Then

$$f_{\mathcal{J}} + f_{\mathcal{J}'} = \sum_{k=\ell+1}^{\mu} (x_{t_k} + x_{t'_k}) f_{J_k} + \sum_{k=\mu+1}^{\sigma} x_{t_k} f_{J_k} + \sum_{k=\mu+1}^{\sigma} x_{t'_k} f_{J'_k}.$$

Since $f_{J_i}, f_{J'_k}$, $i \in [\sigma]$, $k \geq \mu+1$ are pairwise disjoint and $2\sigma - \mu - \ell > \sigma/2$, we have

$$\mathbf{Pr}[f_{\mathcal{J}} \neq f_{\mathcal{J}'}] = \mathbf{Pr}[f_{\mathcal{J}} + f_{\mathcal{J}'} = 1] = (2\sigma - \mu - \ell) 2^{-(t+r)-1} > \frac{\sigma}{2} 4\epsilon^{1-\eta} s \cdot (\epsilon^\eta / s) = 2\epsilon.$$

A contradiction. ◀

References

1. Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.
2. Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000. doi:10.1145/337244.337257.
3. Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 301–309. ACM, 1988. doi:10.1145/62212.62241.
4. Francesco Bergadano, Nader H. Bshouty, and Stefano Varricchio. Learning multivariate polynomials from substitution and equivalence queries. *Electron. Colloquium Comput. Complex.*, TR96-008, 1996. URL: https://eccc.weizmann.ac.il/eccc-reports/1996/TR96-008/index.html.
5. Laurence Bisht, Nader H. Bshouty, and Hanna Mazzawi. On optimal learning algorithms for multiplicity automata. In Gábor Lugosi and Hans Ulrich Simon, editors, *Learning Theory, 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006, Proceedings*, volume 4005 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2006. doi:10.1007/11776420_16.
6. Avrim Blum and Mona Singh. Learning functions of k terms. In Mark A. Fulk and John Case, editors, *Proceedings of the Third Annual Workshop on Computational Learning Theory, COLT 1990, University of Rochester, Rochester, NY, USA, August 6-8, 1990*, pages 144–153. Morgan Kaufmann, 1990. URL: http://dl.acm.org/citation.cfm?id=92620.
7. Nader H. Bshouty. On learning multivariate polynomials under the uniform distribution. *Inf. Process. Lett.*, 61(6):303–309, 1997. doi:10.1016/S0020-0190(97)00021-5.
8. Nader H. Bshouty. Testers and their applications. *Electron. Colloquium Comput. Complex.*, page 11, 2012. URL: https://eccc.weizmann.ac.il/report/2012/011, arXiv:TR12-011.
9. Nader H. Bshouty. Testers and their applications. In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 327–352. ACM, 2014. doi:10.1145/2554797.2554828.

10 Nader H. Bshouty. Almost optimal testers for concise representations. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:156, 2019. URL: https://eccc.weizmann.ac.il/report/2019/156, arXiv:TR19-156.

11 Nader H. Bshouty. Almost optimal proper learning and testing polynomials. In Armando Castañeda and Francisco Rodríguez-Henríquez, editors, *LATIN 2022: Theoretical Informatics - 15th Latin American Symposium, Guanajuato, Mexico, November 7-11, 2022, Proceedings*, volume 13568 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2022. doi:10.1007/978-3-031-20624-5_19.

12 Nader H. Bshouty and Yishay Mansour. Simple learning algorithms for decision trees and multivariate polynomials. *SIAM J. Comput.*, 31(6):1909–1925, 2002. doi:10.1137/S009753979732058X.

13 Michael Clausen, Andreas W. M. Dress, Johannes Grabmeier, and Marek Karpinski. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theor. Comput. Sci.*, 84(2):151–164, 1991. doi:10.1016/0304-3975(91)90157-W.

14 Arne Dür and Johannes Grabmeier. Applying coding theory to sparse interpolation. *SIAM J. Comput.*, 22(4):695–704, 1993. doi:10.1137/0222046.

15 Paul Fischer and Hans Ulrich Simon. On learning ring-sum-expansions. *SIAM J. Comput.*, 21(1):181–192, 1992. doi:10.1137/0221014.

16 Lisa Hellerstein and Rocco A. Servedio. On PAC learning algorithms for rich boolean function classes. *Theor. Comput. Sci.*, 384(1):66–76, 2007. doi:10.1016/j.tcs.2007.05.018.

17 Ron M. Roth and Gyora M. Benedek. Interpolation and approximation of sparse multivariate polynomials over GF(2). *SIAM J. Comput.*, 20(2):291–314, 1991. doi:10.1137/0220019.

18 Robert E. Schapire and Linda Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. *J. Comput. Syst. Sci.*, 52(2):201–213, 1996. doi:10.1006/jcss.1996.0017.

19 Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi:10.1145/1968.1972.

A Proof of Lemma 2

Recall that $\mathbb{HP}_d$ is the set of homogeneous polynomial F of degree d over the variables $Y = \{y_{i,j}\}_{i\in[n],j\in[d]}$ where each monomial of F is of the form $y_{1,i_1}y_{2,i_2}\cdots y_{d,i_d}$ where $\{i_1,i_2,\ldots,i_d\} \in \binom{[n]}{d}$. Throughout this section, q will be a power of two. All the arithmetic operations in the field $\mathrm{GF}(q^t)$ have time complexity $poly(t, \log q)$. Therefore, each arithmetic operation will be considered as one unit-time.

Let $d > 1$ be an integer. We write $\mathrm{GF}(q^t) \xrightarrow{N}_d \mathrm{GF}(q)$ if there are Nd linear maps $M_{i,j} : \mathrm{GF}(q^t) \to \mathrm{GF}(q)$, $i \in [N]$, $j \in [d]$ and elements $\{\alpha_i\}_{i\in[N]}$ in $\mathrm{GF}(q^t)$ such that for every $a_1, a_1, \ldots, a_d \in \mathrm{GF}(q^t)$

$$\prod_{j=1}^{d} a_j = \sum_{k=1}^{N} \left(\alpha_k \prod_{j=1}^{d} M_{k,j}(a_j) \right). \tag{8}$$

We say that $\mathrm{GF}(q^t) \xrightarrow{N}_d \mathrm{GF}(q)$ *in time* T if such maps can be found in time T.

We first prove

▶ **Lemma 18.** *If* $\mathrm{GF}(2^t) \xrightarrow{N}_d \mathrm{GF}(2)$ *in time* T*, then we can find maps* $M_k^* : \mathrm{GF}(2^t)^{dn} \to \mathrm{GF}(2)^{dn}$, $k \in [N]$ *in time* $O(Tn)$ *that are computable in time* $O(dnt)$ *such that, for every* $F \in \mathbb{HP}_d$ *and every* $\beta \in \mathrm{GF}(2^t)^{dn}$*, we have*

$$F(\beta) = \sum_{i=1}^{N} \alpha_i F(M_i^*(\beta)).$$

Proof. Since $\mathrm{GF}(2^t)\xrightarrow{N}_d\mathrm{GF}(2)$ in time T, we can find Nd linear maps $M_{i,j}:\mathrm{GF}(2^t)\to\mathrm{GF}(2)$, $i\in[N]$, $j\in[d]$ and elements $\{\alpha_i\}_{i\in[N]}$ in $\mathrm{GF}(2^t)$ such that for every $a_1,a_1,\ldots,a_d\in\mathrm{GF}(q^t)$, (8) holds.

We define $M_k^*((y_{j,i})_{i\in[n],j\in[d]})=(M_{k,j}(y_{j,i}))_{i\in[n],j\in[d]}$. Let F be any function in $\mathbb{HP}_d$. Suppose

$$F((y_{j,i})_{i\in[n],j\in[d]})=\sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I\prod_{j=1}^{d}y_{j,i_j}\right)$$

where $S\subseteq\binom{[n]}{d}$ and $\gamma_I\in\mathrm{GF}(2)$ for all $I\in S$. Then for every $\beta\in\mathrm{GF}(2^t)^{dn}$, by 8, we have

$$\begin{aligned}
F(\beta)&=\sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I\prod_{j=1}^{d}\beta_{j,i_j}\right)\\
&=\sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I\sum_{k=1}^{N}\left(\alpha_k\prod_{j=1}^{d}M_{k,j}(\beta_{j,i_j})\right)\right)\\
&=\sum_{k=1}^{N}\left(\alpha_k\sum_{I=\{i_1,i_2,\ldots,i_d\}\in S}\left(\gamma_I\left(\prod_{j=1}^{d}M_{k,j}(\beta_{j,i_j})\right)\right)\right)\\
&=\sum_{k=1}^{N}\alpha_kF((M_{k,j}(\beta_{j,i}))_{i\in[n],j\in[d]})=\sum_{k=1}^{N}\alpha_kF(M_k^*(\beta)).
\end{aligned}$$

◀

In particular, to prove Lemma 2, it is enough to prove.

▷ Claim 19. For any integer power of two $t>1$, we have $\mathrm{GF}(2^t)\xrightarrow{\tilde{O}(2^{1.66d})t}_d\mathrm{GF}(2)$ in time $poly(t,2^d)$.

We now prove some Lemmas which lead to this result.

Using exhaustive search for the linear maps $M_{i,j}$ and $\{\alpha_i\}_{i\in N}$, we have

▶ **Lemma 20.** *If* $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ *then* $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ *in time* $q^{O(tdN)}$.

The following four Lemmas are easy to prove.

▶ **Lemma 21.** *If* $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ *in time* T *then for every* $N'\geq N$, $\mathrm{GF}(q^t)\xrightarrow{N'}_d\mathrm{GF}(q)$ *in time* T.

▶ **Lemma 22.** *If* $\mathrm{GF}(q^t)\xrightarrow{N}_d\mathrm{GF}(q)$ *in time* T *then for every* $d'\leq d$, $\mathrm{GF}(q^t)\xrightarrow{N}_{d'}\mathrm{GF}(q)$ *in time* $O(T)$.

▶ **Lemma 23.** *If* $\mathrm{GF}(q^{t_1t_2})\xrightarrow{N_2}_d\mathrm{GF}(q^{t_1})$ *in time* T_1 *and* $\mathrm{GF}(q^{t_1})\xrightarrow{N_1}_d\mathrm{GF}(q)$ *in time* T_2 *then*

$\mathrm{GF}(q^{t_1t_2})\xrightarrow{N_1N_2}_d\mathrm{GF}(q)$

in time $O(T_1+T_2+N_1N_2t_1t_2)$.

▶ **Lemma 24.** *If* $\mathrm{GF}(q^t)\xrightarrow{N_1}_{d_1}\mathrm{GF}(q)$ *in time* T_1 *and* $\mathrm{GF}(q^t)\xrightarrow{N_2}_{d_2}\mathrm{GF}(q)$ *in time* T_2 *then*

$\mathrm{GF}(q^t)\xrightarrow{N_1N_2}_{d_1+d_2}\mathrm{GF}(q)$

in time $O(T_1+T_2+N_1N_2)$.

In particular, by Lemmas 22 and 24, we get

▶ **Lemma 25.** *If* $\mathrm{GF}(q^t) \xrightarrow{N}_{d'} \mathrm{GF}(q)$ *in time* T *then* $\mathrm{GF}(q^t) \xrightarrow{N^{\lceil d/d' \rceil}}_{d} \mathrm{GF}(q)$ *in time* $O(dT/d' + N^{\lceil d/d' \rceil})$.

Now we prove

▶ **Lemma 26.** *If* $q \geq d(t-1)$*, then* $\mathrm{GF}(q^t) \xrightarrow{d(t-1)+1}_{d} \mathrm{GF}(q)$ *in time* $poly(dt)$.

Proof. Let $f^{(1)}, \ldots, f^{(d)} \in \mathrm{GF}(q)[x]$ be arbitrary d polynomials of degree $t-1$ where $f^{(i)} = \sum_{j=0}^{t-1} f_j^{(i)} x^j$. Choose an element $\beta \in \mathrm{GF}(q^t)$ such that $\mathrm{GF}(q^t) \cong \mathrm{GF}(q)[\beta]$. Then $a_1 := f^{(1)}(\beta), \ldots, a_d := f^{(d)}(\beta)$ are arbitrary d elements in $\mathrm{GF}(q^t)$. So it is enough to show that $\prod_{i=1}^{d} f^{(i)}(\beta)$ can be expressed as in (8) with $N = d(t-1)+1$.

Let $f = \prod_{i=1}^{d} f^{(i)}(x) = f_0 + f_1 x + \ldots + f_{d(t-1)} x^{d(t-1)}$. One way to express the coefficients of f as functions of $f_j^{(i)}$, $i \in [d]$, $j = 0, 1, \ldots, d(t-1)$, is the following: First, we have $f_{d(t-1)} = \prod_{i=1}^{d} f_t^{(i)}$. Then substitute $d(t-1)$ distinct elements $\eta_1, \ldots, \eta_{d(t-1)}$ of the field $\mathrm{GF}(q)$ in $\prod_{i=1}^{d} f^{(i)}(x)$ and interpolate to find coefficients $f_0, \ldots, f_{d(t-1)-1}$. This shows that for every $i = 0, 1, \ldots, d(t-1)$, there are $\omega_{i,0}, \omega_{i,1}, \ldots, \omega_{i,d(t-1)}$ independent of $f_j^{(i)}$, $i \in [d]$, $j = 0, 1, \ldots, d(t-1)$ such that

$$f_i = \omega_{i,0} \prod_{k=1}^{d} f_t^{(k)} + \sum_{j=1}^{d(t-1)} w_{i,j} f(\eta_j) = \omega_{i,0} \prod_{k=1}^{d} f_t^{(k)} + \sum_{j=1}^{d(t-1)} w_{i,j} \prod_{k=1}^{d} f^{(k)}(\eta_j).$$

Then

$$\prod_{i=1}^{d} f^{(i)}(x) = \sum_{i=0}^{d(t-1)} f_i x^i = \left(\sum_{i=0}^{d(t-1)} \omega_{i,0} x^i \right) \prod_{k=1}^{d} f_t^{(k)} + \sum_{j=1}^{d(t-1)} \left(\sum_{i=0}^{d(t-1)} w_{i,j} x^i \right) \prod_{k=1}^{d} f^{(k)}(\eta_j).$$

Let

$$\alpha_i = \sum_{i=0}^{d(t-1)} w_{i,j} \beta^i$$

for $j = 0, 1, \ldots, d(t-1)$. Then

$$\prod_{i=1}^{d} f^{(i)}(\beta) = \alpha_0 \prod_{k=1}^{d} f_t^{(k)} + \sum_{j=1}^{d(t-1)} \alpha_j \prod_{k=1}^{d} f^{(k)}(\eta_j).$$

Now recall that $f^{(i)}(\beta)$, $i \in [d]$ are d (arbitrary) elements in $\mathrm{GF}(q^t)$. Since $\eta_j \in \mathrm{GF}(q)$ we have $f_i^{(k)}$ and $f^{(k)}(\eta_j)$ are linear functions from $\mathrm{GF}(q^t)$ to $\mathrm{GF}(q)$. This implies the result. ◀

The following lemma proves Claim 19 for a power of two $t = O(\log d)$.

▶ **Lemma 27.** *Let* c *be a constant. Let* t *be a power of two such that* $t < c \log d$*. We have* $\mathrm{GF}(2^t) \xrightarrow{2^{1.66d}}_{d} \mathrm{GF}(2)$ *in time* $poly(2^d)$.

Proof. By Lemma 26, $\mathrm{GF}(2^2) \xrightarrow{3}_{2} GF(2)$ in time $O(1)$. By Lemma 25, $\mathrm{GF}(2^2) \xrightarrow{N_1 = 3^{\lceil d/2 \rceil}}_{d} GF(2)$ in time $O(N_1)$. Now for any i, by Lemma 26, $\mathrm{GF}(2^{2^{i+1}}) \xrightarrow{2^{2^i}+1}_{2^{2^i}} GF(2^{2^i})$ in time $poly(2^{2^i})$. By Lemma 25, $\mathrm{GF}(2^{2^{i+1}}) \xrightarrow{N_i}_{d} \mathrm{GF}(2^{2^i})$ in time $poly(2^{2^i}) + O(N_i)$ where $N_i = (2^{2^i}+1)^{\lceil d/2^{2^i} \rceil}$. By Lemma 23, we have $\mathrm{GF}(2^t) \xrightarrow{N}_{d} \mathrm{GF}(2)$ in time $poly(2^t, 2^{\log^2 d} N) = poly(2^d)$ where

$$N = 3^{\lceil d/2 \rceil} 5^{\lceil d/4 \rceil} \cdots (2^{t/2}+1)^{\lceil d/2^{t/2} \rceil} \leq 2^{O(\log^2 d)} 2^{\left(\frac{\log 3}{2} + \frac{\log 5}{4} + \cdots\right) d} \leq 2^{1.66d}.$$ ◀

The following lemma will be frequently used to prove Claim 19 for larger values of d.

▶ **Lemma 28.** *Let t and t' be powers of 2, where $2\log(dt) > t' \geq \log(dt)$. Then $\mathrm{GF}(2^t) \xrightarrow{dt/t'}_d \mathrm{GF}(2^{t'})$ in time $poly(dt)$.*

Proof. We have $q = 2^{t'} \geq dt \geq d(t/t' - 1)$. By Lemma 21 and 26, the result follows. ◀

The following lemma proves Claim 19 for a power of two $t = d^{O(1)}$.

▶ **Lemma 29.** *Let c be a constant. Let t be a power of two such that $t < d^c$. We have $\mathrm{GF}(2^t) \xrightarrow{\tilde{O}(2^{1.66d})t}_d \mathrm{GF}(2)$ in time $poly(2^d)$.*

Proof. By Lemma 28, $\mathrm{GF}(2^t) \xrightarrow{dt/t'}_d \mathrm{GF}(2^{t'})$ in time $poly(dt)$ for some power of two t', where $t' \leq 2(c+1)\log d$. By Lemma 27, $\mathrm{GF}(2^{t'}) \xrightarrow{2^{1.66d}}_d \mathrm{GF}(2)$ in time $poly(2^d)$. By Lemma 23, $\mathrm{GF}(2^t) \xrightarrow{\tilde{O}(2^{1.66d})t}_d \mathrm{GF}(2)$ in time $poly(2^d)$. ◀

The following lemma proves Claim 19 for a power of two $t = 2^{d^{O(1)}}$.

▶ **Lemma 30.** *Let c be a constant. Let t be a power of two such that $t < 2^{d^c}$. We have $\mathrm{GF}(2^t) \xrightarrow{\tilde{O}(2^{1.66d})t}_d \mathrm{GF}(2)$ in $poly(t, 2^d)$.*

Proof. By Lemma 28, $\mathrm{GF}(2^t) \xrightarrow{dt/t'}_d \mathrm{GF}(2^{t'})$ in time $poly(dt)$ for some power of two t', where $t' \leq d^{c+2}$. By Lemma 29, $\mathrm{GF}(2^{t'}) \xrightarrow{\tilde{O}(2^{1.66d})t'}_d \mathrm{GF}(2)$ in time $poly(2^d)$. By Lemma 23, $\mathrm{GF}(2^t) \xrightarrow{\tilde{O}(2^{1.66d})t}_d \mathrm{GF}(2)$ in time $poly(t, 2^d)$. ◀

The following is from [9], Corollary 17 item 7

▶ **Lemma 31.** *For any $q \geq d$ and t, we have $\mathrm{GF}(q^t) \xrightarrow{O(d^4 t)}_d \mathrm{GF}(q)$.*

By Lemma 31 and Lemma 20 we get

▶ **Lemma 32.** *For any $q \geq d$ and t, we have $\mathrm{GF}(q^t) \xrightarrow{O(d^4 t)}_d \mathrm{GF}(q)$ in time $q^{O(t^2 d^5)}$.*

The following lemma proves Claim 19 for a power of two $t > 2^{d^{12}}$.

▶ **Lemma 33.** *Let $t \geq 2^{d^{12}}$ be a power of two. We have $\mathrm{GF}(2^t) \xrightarrow{\tilde{O}(2^{1.66d})t}_d \mathrm{GF}(2)$ in time $poly(t, 2^d)$.*

Proof. Let $2d > 2^\ell \geq d$ power of two. By Lemma 28, we have $\mathrm{GF}(2^t) \xrightarrow{dt/t'}_d \mathrm{GF}(2^{t'})$ in time $poly(dt)$ for a power of two $2\log(dt) > t' \geq \log(dt)$. Then again, by Lemma 28, we have $\mathrm{GF}(2^{t'}) \xrightarrow{dt'/t''}_d \mathrm{GF}(2^{t''})$ in time $poly(dt')$ for a power of two $2\log(dt') > t'' \geq \log(dt')$. By Lemma 32, $\mathrm{GF}(2^{t''}) \xrightarrow{O(d^4 t''/\ell)}_d \mathrm{GF}(2^\ell)$ in time

$$(2^\ell)^{O((t''/\ell)^2 d^5))} \leq 2^{O(d^6 \log^2 \log t)} \leq 2^{O(\log^{1/2} t \log^2 \log t)} < t.$$

By Lemma 29, $\mathrm{GF}(2^\ell) \xrightarrow{\tilde{O}(2^{1.66d})\ell}_d \mathrm{GF}(2)$ in time $poly(2^d)$. By Lemma 23, $\mathrm{GF}(2^t) \xrightarrow{\tilde{O}(2^{1.66d})t}_d \mathrm{GF}(2)$ in time $poly(t, 2^d)$. ◀

Now Claim 19 follows immediately from Lemma 29 and 33.

B Proof of Lemma 8

In this Section, we prove

▶ **Lemma 7.** *Let $t = d'm$ be an integer where $d < d' = O(d)$ and $\log n < m = O(\log n)$. There is a deterministic non-adaptive proper exact learning algorithm that learns $\mathbb{P}_{d,s}[2^t]$ with $2s$ queries in $\mathrm{GF}(2^t)^n$ in time $poly(d,s) \cdot \tilde{O}(n)$.*

Proof. The proof is the same as in [3].

Let $w \in \mathrm{GF}(2^t)$ such that $\mathrm{GF}(2^t) \simeq \mathrm{GF}(2^m)[w]$. The minimal polynomial $p \in \mathrm{GF}(2^m)[x]$ of w is of degree $d' - 1 \geq d$. Each element in $\mathrm{GF}(2^t)$ can be represented as $\gamma_0 + \gamma_1 w + \cdots + \gamma_{d'-1} w^{d'-1}$ where $\gamma_i \in \mathrm{GF}(2^m)$ for all i.

Let $f(x) = \sum_{i=1}^{s} a_i M_i$, where $M_i = x_{r_{i,1}} x_{r_{i,2}} \cdots x_{r_{i,d_i}}$ and $d_i \leq d$ for all $i \in [s]$. Here, we assume that the number of monomials in f is exactly s. We will show later the case when the size is less than s.

We choose n distinct elements $\alpha_1, \alpha_2, \cdots, \alpha_n$ in $\mathrm{GF}(2^m)$. This is possible because $2^m \geq n$. The queries of the algorithm are

$$u_i = ((w-\alpha_1)^i, (w-\alpha_2)^i, \ldots, (w-\alpha_n)^i), \quad i = 0, 1, 2, \ldots, 2s-1.$$

Let $v_i = f(u_i)$.

We now show how to find M_i, $i \in [s]$, and then find the coefficients a_i, $i \in [s]$.

We first give some observations. Let $m_i = M_i(u_1)$ and $\Lambda(x) = \prod_{i \in [s]} (x - m_i) = \sum_{i=0}^{s} \lambda_i x^i$. Notice that $M_i(u_j) = m_i^j$ and $\Lambda(m_i) = 0$ for every i. Now, for every $\ell = 0, 1, 2, \ldots, s-1$

$$\begin{aligned} 0 &= \sum_{i=1}^{s} a_i m_i^\ell \Lambda(m_i) = \sum_{i=1}^{s} a_i m_i^\ell \sum_{j=0}^{s} \lambda_j m_i^j = \sum_{j=0}^{s} \lambda_j \sum_{i=1}^{s} a_i m_i^{\ell+j} \\ &= \sum_{j=0}^{s} \lambda_j \sum_{i=1}^{s} a_i M_i(u_{\ell+j}) = \sum_{j=0}^{s} \lambda_j v_{\ell+j}. \end{aligned}$$

Since $\lambda_s = 1$, we get the linear equation $V\lambda = v$, where

$$V = \begin{pmatrix} v_0 & v_1 & v_2 & \cdots & v_{s-1} \\ v_1 & v_2 & v_3 & \cdots & v_s \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_{s-1} & v_s & v_{s+1} & \cdots & v_{2s-1} \end{pmatrix}, \lambda = \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{s-1} \end{pmatrix} \text{ and } v = \begin{pmatrix} -v_s \\ -v_{s+1} \\ \vdots \\ -v_{2s-1} \end{pmatrix}.$$

Ben-Or and Tiwari show in [3] that if f has s monomials, then V is a non-singular matrix. Then we can find $\lambda = V^{-1}v$ and therefore $\Lambda(x)$. By factoring $\Lambda(x)$, we get $m_i = M_i(u_1)$. Let $m_i = m_{i,0} + m_{i,1} w + \cdots + m_{i,d'-1} w^{d'-1}$ where $m_{i,j} \in \mathrm{GF}(2^m)$ for all j. Now since $m_i = M_i(u_1) = (w - \alpha_{r_{i,1}}) \cdots (w - \alpha_{r_{i,d_i}})$ and the minimal polynomial of w is of degree $d' - 1 \geq d \geq d_i$, we have $m_{i,0} + m_{i,1} x + \cdots + m_{i,d'-1} x^{d'-1} = (x - \alpha_{r_{i,1}}) \cdots (x - \alpha_{r_{i,d_i}})$. So by factoring $m_{i,0} + m_{i,1} x + \cdots + m_{i,d'-1} x^{d'-1}$, we get $\alpha_{r_{i,1}}, \ldots, \alpha_{r_{i,d_i}}$ and therefore $M_i(x)$. To find the coefficients a_i, we solve the linear equation $f(u_i) = \sum_{i=1}^{s} a_i M_i(u_i) = v_i, \quad i = 0, 1, \ldots, s-1$. This finishes the case when the number of monomials is s. When the number of monomial is at most s, then it is known that the number of monomials is equal to the maximum r such that the upper left $r \times r$ sub-matrix of V is non-singular. So we find r and continue as above. ◀

Cut Paths and Their Remainder Structure, with Applications

Massimo Cairo
Department of Computer Science, University of Helsinki, Finland

Shahbaz Khan[1]
Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India

Romeo Rizzi
Department of Computer Science, University of Verona, Italy

Sebastian Schmidt
Department of Computer Science, University of Helsinki, Finland

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland

Elia C. Zirondelli
Department of Mathematics, University of Trento, Italy

Abstract

In a strongly connected graph $G = (V, E)$, a *cut arc* (also called *strong bridge*) is an arc $e \in E$ whose removal makes the graph no longer strongly connected. Equivalently, there exist $u, v \in V$, such that all u-v walks contain e. Cut arcs are a fundamental graph-theoretic notion, with countless applications, especially in reachability problems.

In this paper we initiate the study of *cut paths*, as a generalisation of cut arcs, which we naturally define as those paths P for which there exist $u, v \in V$, such that all u-v walks contain P as subwalk. We first prove various properties of cut paths and define their remainder structures, which we use to present a simple $O(m)$-time verification algorithm for a cut path ($|V| = n$, $|E| = m$).

Secondly, we apply cut paths and their remainder structures to improve several reachability problems from bioinformatics, as follows. A walk is called *safe* if it is a subwalk of every node-covering closed walk of a strongly connected graph. *Multi-safety* is defined analogously, by considering node-covering *sets* of closed walks instead. We show that cut paths provide *simple* $O(m)$-time algorithms verifying if a walk is safe or multi-safe. For multi-safety, we present the first linear time algorithm, while for safety, we present a simple algorithm where the state-of-the-art employed complex data structures. Finally we show that the simultaneous computation of remainder structures of all subwalks of a cut path can be performed in linear time, since they are related in a structured way. These properties yield an $O(mn)$-time algorithm outputting all maximal multi-safe walks, improving over the state-of-the-art algorithm running in time $O(m^2 + n^3)$.

The results of this paper only scratch the surface in the study of cut paths, and we believe a rich structure of a graph can be revealed, considering the perspective of a path, instead of just an arc.

2012 ACM Subject Classification Applied computing → Computational biology; Mathematics of computing → Paths and connectivity problems; Theory of computation → Graph algorithms analysis

Keywords and phrases reachability, cut arc, strong bridge, covering walk, safety, persistence, essentiality, genome assembly

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.17

Related Version *Previous Version*: https://arxiv.org/abs/2210.07530

[1] Most of the work by the author was done while he was affiliated to University of Helsinki.

© Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian Schmidt, Alexandru I. Tomescu, and Elia C. Zirondelli;
licensed under Creative Commons License CC-BY 4.0

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 17; pp. 17:1–17:17

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

Funding *Alexandru I. Tomescu*: This work was partially funded by the Academy of Finland (grants No. 322595, 328877). *Shahbaz Khan, Sebastian Schmidt and Alexandru I. Tomescu*: This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO).

1 Introduction

1.1 Motivation

Connectivity problems are a fundamental aspect of graph theory and graph algorithms. In directed graphs, *cut arcs* (also known as *strong bridges*) are a basic structure to characterise the reachability properties of the graph. They are defined as arcs whose removal makes the graph no longer strongly connected, or equivalently, as arcs e such that there exists a pair of nodes u, v such that each u-v walk contains e. Cut arcs and the related strongly connected components are nowadays part of any lecture about graph theory. Moreover, significant work has been done to investigate the properties of a graph in relation to its cut arcs, e.g. by finding all cut arcs in linear time [14], and, after linear-time preprocessing, answering connectivity queries in constant time under the removal of any single arc [11]. These gave rise to further theoretical advances, and are used in algorithms to compute e.g. 2-vertex connected components in directed graphs [10] or 2-edge connected components in directed graphs [9]. The results are also useful for more practical works, like in the analysis of non-equilibrium biochemical reaction networks [22] or when analysing real-world graphs such as social networks or the world wide web [15]. Naturally, cut arcs play a crucial role in various practical networks, as they represent critical links in e.g. communication networks, road networks or transportation networks.

A natural generalisation of a cut arc is a *cut path*[2], similarly defined as a walk (not a single arc) W such that there exists a pair of nodes u, v such that each u-v walk has W as subwalk. While there is (to the best of our knowledge) no research around cut paths as such, they seem to be equally fundamental as cut arcs. On the practical side, one can view cut paths as representing critical *routes* through social networks, the world wide web, communication networks, road networks or transportation networks. Additionally, in practical applications, when nodes represent street crossings or network routers, cut paths imply critical *links* within these objects. On the more theoretical side, in this paper we show that cut paths are a useful tool in some reachability problems theoretically modelling the genome assembly problem in bioinformatics. In addition to exhibiting interesting properties on their own, cut paths also allows us to improve several of these results, as we discuss in Section 1.3.

1.2 Overview of cut paths

To give an overview of cut paths, we start with some basic definitions. A *graph* $G = (V, E)$ with n nodes and m arcs is directed and may have self-loops. For an arc $e = (u, v)$, we call $u = \text{TAIL}(e)$ its *tail* and $v = \text{HEAD}(e)$ its *head*. Sets of nodes can induce subgraphs in the standard manner. A graph is *strongly connected* if each pair of nodes is connected by a directed path in both directions. A *strongly connected component (SCC)* is a maximal induced subgraph that is strongly connected. A *cut arc* is an arc that upon removal increases the number of strongly connected components in G.

[2] Note that walks that contain a cycle cannot be cut paths, hence the name *cut path* and not *cut walk*.

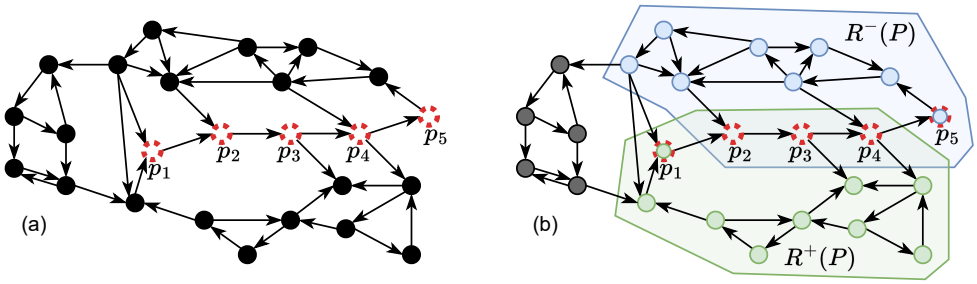


Figure 1 (a) A cut path $P = (p_1, p_2, p_3, p_4, p_5)$, highlighted by red dashed nodes. Each walk from p_1 to p_5 has P as a subwalk. (b) The remainder structure of P. The set $R^+(P)$ is enclosed by the green area, and the set $R^-(P)$ is enclosed by the blue area. Its source component $S^-(P)$ is highlighted by blue nodes, its sink component $S^+(P)$ is highlighted by green nodes and the inner component $\bar{S}(P)$ is highlighted by grey nodes. The path component $\bar{P}(P)$ is the intersection of $R^+(P)$ and $R^-(P)$. Note that, to get from $S^+(P)$ to $S^-(P)$, one needs to traverse P completely.

An equivalent definition of a cut arc in strongly connected graphs is an arc that is part of all walks from some node to some other node. Generalising, a *cut path* is a walk that is a subwalk of all walks from some node to some other node. We assume a graph to be strongly connected from here on.

When removing a cut arc (u, v) from a graph, the graph is not strongly connected anymore, but instead contains multiple SCCs. There is exactly one source SCC (containing v), which is an SCC without any incoming arcs from other SCCs, and exactly one sink SCC (containing u), which is an SCC without any outgoing arcs to other SCCs. The source is connected to the sink via direct arcs or via other SCCs.

A similar structure exists for cut paths, which we call their *remainder structure*. This structure is helpful both to efficiently check whether a walk is a cut path, and for applying cut paths to other problems. We define the remainder structure of a cut path $P = (p_1, \ldots, p_\ell)$ (where p_i are nodes), which we denote $R(P) = (S^-(P), \bar{S}(P), S^+(P), \bar{P}(P))$, as follows. Let $R^+(P)$ be the set of nodes reachable by walks starting from the first node of P without using $(p_{\ell-1}, p_\ell)$. Symmetrically, let $R^-(P)$ be the set of nodes reaching the last node of P without using (p_1, p_2). With this definition, $R^+(P)$ and $R^-(P)$ may intersect, so we define the *source component* $S^-(P) := R^-(P) \setminus R^+(P)$, the *sink component* $S^+(P) := R^+(P) \setminus R^-(P)$, the *inner component* $\bar{S}(P) := V \setminus (S^- \cup S^+)$, and the *path component* $\bar{P}(P) := R^-(P) \cap R^+(P)$. In the remainder structure $R(P)$, for each node u in the subgraph induced by $S^-(P)$ and each node v in the subgraph induced by $S^+(P)$, the walk P is on every u-v walk in G. Computing the remainder structure is trivial given its definition, and it additionally allows for a very simple way to efficiently check if a walk is a cut path.

▶ **Theorem 1** (Efficient verification of cut paths). *Let $P = (p_1, \ldots, p_\ell)$ be a walk of length $\ell - 1 \geq 1$. P is a cut path if and only if $\bar{P}(P) = \{p_2, \ldots, p_{\ell-1}\}$ (specifically, for $\ell = 2$, $\bar{P}(P)$ is empty). This property can be verified in $O(m)$ time.*

1.3 Applications of cut paths

Background. We apply cut paths to improve several reachability-based results that have been given over the last years for "safe walks", motivated by the genome assembly problem in bioinformatics [20, 5, 6, 1, 17, 16]. We give here a minimal self-contained description, and refer the reader to these papers for motivation and applications. One can formulate the genome assembly problem as finding one closed node-covering walk (i.e., passing through

every node at least once)[3] in a given strongly connected graph built from the input sequencing data. Since such graphs may admit multiple such walks, one can define a *safe walk* as one appearing in any closed node-covering walk of a strongly connected graph. Formally:

▶ **Definition 2** (Safe walk [20]). *Given a graph G, a walk is* safe *if it is a subwalk of each possible closed node-covering walk of G.*

We are interested in enumerating all *maximal* safe walks, namely all those that are not a proper subwalk of another safe walk. These can be thought as representing the maximal correct (partial) answers to the genome assembly problem. In [20] it is argued that some specific types of walks used by genome assembly programs are safe walks, and thus finding *all* maximal safe walks can be relevant in practice, since it can lead to longer parts of the genome being reconstructed. Similar problems have been previously studied without the covering constraint but instead by considering subwalks of all possible walks from a given node s to a given node t [4], or with the covering constraint set to *cover exactly once* (i.e., closed Eulerian walks) [16, 2].

Safe walks can be characterised as follows. Let $(w_1, \ldots, w_\ell)$ be a walk. A path from w_i to w_j, with $1 < i \leq j < \ell$, with first arc different from (w_j, w_{j+1}), and last arc different from (w_{i-1}, w_i), is called a *forbidden path*. Tomescu and Medvedev [20] proved that a walk is safe if and only if it has no forbidden path and all its arcs are cut arcs. For a walk made up only of cut arcs, such a forbidden path can be seen as a NO-certificate, since it testifies that the walk is not safe. Even though NO-certificates are usually harder to check, Cairo et al. [6] showed that the absence of a forbidden path can be checked in $O(m)$ time, but using complex data structures. Moreover, all maximal walks without forbidden paths can still be enumerated in $O(mn)$ time [5]. By appropriately splitting such walks at non-cut arcs, and removing duplicates, also maximal safe walks can be enumerated in $O(mn)$ time.[4]

One can also consider another variant of the problem, where one needs to assemble an unknown number of genomes from a graph, or a genome with an unknown number of chromosomes [1]. For this, one can formulate the genome assembly problem as finding a node-covering *set* of closed walks (i.e., such that every node appears in at least one walk in the set). In this setting, the notion of safety is adapted as follows:

▶ **Definition 3** (Multi-safe walk [20, 1]). *Given a graph G, a walk is* multi-safe *if it is a subwalk of some walk in each possible node-covering set of proper closed walks of G.*

The theory around multi-safe walks is less developed, the only algorithmic result being by Obscura Acosta et al. [1], who showed that all maximal multi-safe walks can be enumerated in $O(m^2 + n^3)$ time. This algorithm is also based on forbidden paths, with some additional conditions. One reason behind this lack of overall progress around multi-safe walks can be due to the lack of a YES-certificate, which requires building new machinery from scratch.

Safety-related previous works. The idea of partial solutions common to all solutions to a problem is very natural and has appeared in several other contexts. For example, Costa [7] studied *persistent edges* belonging to all maximum matching of a bipartite graph, and Hammer

[3] To be precise, [20, 5, 6, 1] focus mostly on the arc-covering case, where the closed walks have to pass through all *arcs* at least once. In this paper we focus on the node-covering case, for two reasons: first, it has a more direct relation to cut paths, and second, the arc-covering case can be reduced to it in linear time by subdividing every arc (i.e., introducing a node in the middle of every arc).

[4] This fact was not observed previously in the literature (recall that in this paper we are defining safety in terms of node-covering walks), but follows by standard techniques of removing duplicates using a suffix tree. For completeness, we explain this in Section 4 and Appendix A.

et al. [13] studied *persistent nodes* belonging to all maximum stable sets. Recently, Bumpus et al. [3] studied *c-essential vertices*, defined as those contained in all c-approximate solutions to e.g. Odd Cycle Transversal and Directed Feedback Vertex Set problems. See also Table 1 in [3] for algorithms detecting *some* c-essential vertices for several other NP-hard problems. As opposed to the latter problems, in this paper we tackle polynomially solvable problems (computing closed node-covering walks is trivial), and thus their safe partial solutions admit rich structures which can be exploited in getting efficient algorithms enumerating *all* of them.

Our results. We show that cut paths and their remainder structure provide a *flexible* technique to study both safe and multi-safe walks. For example, they can be used to derive natural YES-certificates for both types of walks.

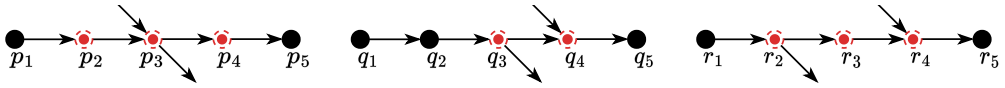


Figure 2 Walks P, Q, R and their cores highlighted by red dashed nodes. Nodes p_3, q_3 and r_2 are splits, and nodes p_3, q_4 and r_4 are joins. The walks P and Q are interleaved, and the walk R is non-interleaved. Note that $(q_1, \dots, q_5)$ is defined to be interleaved, since even though it has separate split-free and join-free parts, they are not trivially safe since (q_3, q_4) is only safe if it is a cut arc.

To describe our results, we need additional definitions for walks. Examples for these definitions are given in Figure 2. A *split* is a node with at least two outgoing arcs and a *join* is a node with at least two incoming arcs. Let $W = (w_1, \dots, w_\ell)$ be a walk with $\ell \geq 2$. The *inner* nodes of W are $w_2, \dots, w_{\ell-1}$. Let w_i be its first inner join, or w_ℓ if W has no inner join. Let w_j be its last inner split, or w_1 if W has no inner split. Then W is an *interleaved walk* if $i \leq j + 1$ and a *non-interleaved walk* otherwise. The *core* of an interleaved walk is its subwalk from w_{i-1} to w_{j+1}. The *core* of a non-interleaved walk is its subwalk from w_j to w_i.

A first consequence is that verifying whether a walk is safe can now be done by a simple check whether the *core* of a walk is a cut path (after excluding trivial cases). Since this can be computed in linear-time using simple graphs traversals (Theorem 1), we obtain a verification algorithm much simpler than the one in [6] (which uses complex data structures from [11], which in turn uses dominator trees [8] and loop-nesting forests [19]).

▶ **Theorem 4** (Safety characterisation and verification). *Let W be a walk, and let $C(W)$ be its core. W is safe if and only if it is a non-interleaved walk or $C(W)$ is a cut path. This property can be verified in $O(m)$ time.*

In our proof, we use the properties of the remainder structure to show that if the core of a walk W is not a cut path, then it can be replaced by a walk avoiding the core in any closed node-covering walk. On the other hand, if the core is a cut path, then there is a pair of nodes that can only be connected via the core. Since a closed node-covering walk contains a subwalk between each pair of nodes, that makes W safe.

For the multi-safe case we obtain a YES-certificate based on checking a property of the remainder structure. This leads to the first linear-time algorithm verifying whether a walk is multi-safe. In contrast to safe walks, the characterisation depends on the existence of certain SCCs of size one. Intuitively, a multi-safe walk must be safe, since otherwise there would be a closed node-covering walk avoiding it, which then also disproves multi-safety. Moreover, if $R^+(C(W))$, $R^-(C(W))$ and $\bar{S}(C(W))$ of a core $C(W)$ contain only SCCs of size at least two, then they can all be covered by proper closed walks without leaving the respective component, and thus without using $C(W)$ as subwalk, disproving the multi-safety of W. If

however one of them contains an SCC of size one, then to cover this SCC, the respective component needs to be left and reentered. This can only happen by using $C(W)$ as subwalk, hence W is multi-safe.

▶ **Theorem 5** (Multi-safety characterisation and verification). *Let W be a walk, and let $C(W)$ be its core. If W is non-interleaved, then it is multi-safe. Otherwise, it is multi-safe if and only if it is safe and any of $G[\bar{S}(C(W))], G[R^+(C(W))]$ or $G[R^-(C(W))]$ contains an SCC of size one. This property can be verified in $O(m)$ time.*

Lastly, we improve the existing $O(m^2 + n^3)$-time algorithm enumerating all maximal multi-safe walks. A naive application of Theorem 5 would lead to an $O(m^2 n)$-time algorithm for this enumeration problem, already improving the previous one for dense graphs. However, by proving several additional properties of the remainder structure, we can amortise the time to just $O(mn + o)$, where o is the size of the output. First, the remainder structure of all subwalks P' of a given cut path P can be precomputed in linear time. This works because when shifting either the start or end of the subwalk to the right (along the walk), then the set $R^+(P')$ monotonously grows and the set $R^-(P')$ monotonously shrinks (except for some subwalks that are trivial to handle without the remainder structure). Further, when growing $R^+(P')$, only complete SCCs get removed from the inner component, and when shrinking $R^-(P')$, the existing SCCs in the inner component are not altered.

▶ **Theorem 6** (Enumerating maximal multi-safe walks). *All maximal multi-safe walks can be identified in $O(mn)$ time and enumerated in $O(mn + o)$ time, where o is the total length of the output and it holds that $o \in O(n^3)$.*

In a nutshell, using cut paths and the remainder structure, we gain a deeper understanding of critical structures for the connectivity of directed graphs. In bioinformatics applications, this allows us to get better characterisations for two problems that for the first time admit a simple to compute and verify YES-certificate. Moreover, meticulously investigating the properties of the remainder structure, we improve over the complexity of the best known enumeration algorithm for maximal multi-safe walks.

Notation. A *split* is a node with at least two outgoing arcs, and a *join* is a node with at least two incoming arcs. Let $G = (V, E)$ be a strongly connected graph, with $|V| = n$ and $|E| = m \geq n$. We denote the removal of an arc $e \in E$ by $G - e$. Given a subset of nodes $V' \subseteq V$, the subgraph induced by V' subgraph is defined as $G[V'] = (V', \{(u, v) \in E \mid u, v \in V'\})$.

For two nodes $u, v \in V$, a *u-v walk* of *length* $\ell - 1$ is a sequence of nodes $W = (w_1, \ldots, w_\ell)$ with $w_1 = u$ and $w_\ell = v$ and such that for each $i \in \{1, \ldots, \ell - 1\}$ it holds that $(w_i, w_{i+1}) \in E$. The *tail* $\textsc{tail}(W)$ of W is w_1, and the *head* $\textsc{head}(W)$ of a W is w_ℓ. W is *closed* if $u = v$ and *open* otherwise. W is a *path* if all nodes are unique except that $w_1 = w_\ell$ is allowed. The *inner nodes* of a W are the nodes $w_2, \ldots, w_{\ell-1}$, where a walk of length $\ell - 1 \leq 1$ has no inner nodes. The notation WW' denotes the *concatenation* of walks W and W' if $\textsc{head}(W) = \textsc{tail}(W')$. *Subwalks* of walks are defined in the standard manner, where subwalks of closed walks may run over the end (e.g. (c, a, b) is a subwalk of (a, b, c, a)). A *proper subwalk* of W is a subwalk that is shorter than W.

2 Cut paths and their remainder structure

In this section, we formally define cut paths and the remainder structure and prove their main properties. See Appendix B for all formal proofs that we omitted here.

▶ **Definition 7.** *A walk W in a strongly connected graph $G = (V, E)$ is a* cut path *if there is a pair of nodes $u, v \in V$ such that all u-v walks in G have W as subwalk.*

One intuitive property of a cut path is that it is an open path.

▶ **Lemma 8** (Open path property). *A cut path is an open path.*

Further, for any cut path P, the pair of nodes $\text{TAIL}(P), \text{HEAD}(P)$ is a *witness* for P being a cut path, i.e. all $\text{TAIL}(P)$-$\text{HEAD}(P)$ walks contain P as a subwalk.

▶ **Lemma 9** (Witness property). *A walk W is a cut path if and only if it is subwalk of all $\text{TAIL}(W)$-$\text{HEAD}(W)$ walks.*

2.1 Restricted reachabilities

The remainder structure is based on the restricted reachabilities of a walk. Even though only open paths can be cut paths, we define the remainder structure here on arbitrary walks. As we see below, the remainder structure makes it easy to check if a walk is a cut path.

▶ **Definition 10.** *The* restricted forward and backward reachability *of a walk $W = (w_1, \ldots, w_\ell)$ in a strongly connected graph $G = (V, E)$ are*

$$R^+(W) := \{v \in V \mid \exists\, \text{TAIL}(W)\text{-}v \text{ walk in } G - (w_{\ell-1}, w_\ell)\},$$
$$R^-(W) := \{v \in V \mid \exists\, v\text{-HEAD}(W) \text{ walk in } G - (w_1, w_2)\}.$$

The restricted reachabilities exhibit the following property, which makes them simple to work with.

▶ **Lemma 11** (Bottleneck property). *For a walk $W = (w_1, \ldots, w_\ell)$, the only arc leaving $R^+(W)$ is $(w_{\ell-1}, w_\ell)$ and the only arc entering $R^-(W)$ is (w_1, w_2).*

Proof. Assume for a contradiction that an arc (u, v) different from $(w_{\ell-1}, w_\ell)$ would leave $R^+(W)$. Since $u \in R^+(W)$, there is a $\text{TAIL}(W)$-u walk without $(w_{\ell-1}, w_\ell)$. Appending (u, v) to such a walk cannot introduce $(w_{\ell-1}, w_\ell)$ as subwalk, because (u, v) is not $(w_{\ell-1}, w_\ell)$. Therefore, $v \in R^+(W)$, contradicting (u, v) leaving $R^+(W)$.

By symmetry, the only arc entering $R^-(W)$ is (w_1, w_2). ◀

Note that by leaving $R^+(W)$, one always ends up in $R^-(W)$ (or one was in $R^-(W)$ already, if one leaves $R^+(W)$ from a node in $R^+(W) \cap R^-(W)$), and by entering $R^-(W)$, one always comes from $R^+(W)$ (and possibly ends up in $R^+(W)$ again, if one enters $R^-(W)$ at a node in $R^+(W) \cap R^-(W)$). Further, the restricted reachabilities are strongly connected in certain cases.

▶ **Lemma 12** (Strong connectivity property). *Let $P = (p_1, \ldots, p_\ell)$ be a cut path. If the last inner node of P is a split, then $G[R^+(P)]$ is strongly connected. If the first inner node of P is a join, then $G[R^-(P)]$ is strongly connected.*

Proof. Let the last inner node of P be a split. By definition, p_1 reaches all nodes in $R^+(P)$ via walks not leaving $R^+(P)$. Assume for a contradiction that there is a node $v \in R^+(P)$ that cannot reach p_1 without leaving $R^+(P)$. Then by Lemma 11, each v-p_1 walk contains $(p_{\ell-1}, p_\ell)$, so each $p_{\ell-1}$-p_1 walk contains $(p_{\ell-1}, p_\ell)$. Let $v' \neq v_\ell$ be a node with $(p_{\ell-1}, v') \in E$. Then since v' is reachable from $p_{\ell-1}$, each v'-p_1 walk contains $(p_{\ell-1}, p_\ell)$. So there is a p_1-p_ℓ walk W via v' that does not have p_1 or p_ℓ as inner nodes, so it does not have P as subwalk. By Lemma 9, this contradicts P being a cut path. ◀

Note that, whenever a restricted reachability is not strongly connected, then it consists of a strongly connected component, plus nodes from P that form SCCs of size one.

2.2 The remainder structure

▶ **Definition 13.** *The* remainder structure $R(W) = (S^-(W), \bar{S}(W), S^+(W), \bar{P}(W))$ *of a walk W in a strongly connected graph $G = (V, E)$ is defined as*

$$
\begin{aligned}
S^-(W) &:= R^-(W) \setminus R^+(W) \text{ (the source component),}\\
\bar{S}(W) &:= V \setminus (R^+(W) \cup R^-(W)) \text{ (the inner component),}\\
S^+(W) &:= R^+(W) \setminus R^-(W) \text{ (the sink component),}\\
\bar{P}(W) &:= R^+(W) \cap R^-(W) \text{ (the path component).}
\end{aligned}
$$

Note that the remainder structure is a decomposition of the nodes of G. For checking if a walk is a cut path, we can use the *inner path property* of the remainder structure. The inner path property can be checked in linear time with trivial algorithms that directly follow from the definition of the remainder structure and the property.

▶ **Theorem 1** (Efficient verification of cut paths)**.** *Let $P = (p_1, \ldots, p_\ell)$ be a walk of length $\ell - 1 \geq 1$. P is a cut path if and only if $\bar{P}(P) = \{p_2, \ldots, p_{\ell-1}\}$ (specifically, for $\ell = 2$, $\bar{P}(P)$ is empty). This property can be verified in $O(m)$ time.*

Proof. By definition, it holds that $\{p_2, \ldots, p_{\ell-1}\} \subseteq \bar{P}(P)$. Assume for a contradiction that there was a node $v \in \bar{P}(P) \setminus \{p_2, \ldots, p_{\ell-1}\}$. If $v = p_\ell$, then $p_\ell \in R^+(P)$, so there is a $\text{TAIL}(P)$-$\text{HEAD}(P)$ walk that does contain $(p_{\ell-1}, p_\ell)$, which by Lemma 9 contradicts P being a cut path. In the same way, if $v = p_1$, then $p_1 \in R^-(P)$, which again contradicts P being a cut path. Therefore, $v \notin \{p_1, \ldots, p_\ell\}$.

Since $v \in R^+(P)$, there is a $\text{TAIL}(P)$-v walk W_1 in G that does not contain $(p_{\ell-1}, p_\ell)$. Further, since $v \in R^-(P)$, there is a v-$\text{HEAD}(P)$ walk W_2 in G that does not contain (p_1, p_2). Then, $W = W_1W_2$ is a $\text{TAIL}(P)$-$\text{HEAD}(P)$ walk. Since $v \notin \{p_1, \ldots, p_\ell\}$, it holds that concatenating W_1W_2 does not introduce P as subwalk. So by Lemma 9 it holds that P is not a cut path, which completes the contradiction.

If $\ell > m$, then W contains a cycle, so by Lemma 8 it is not a cut path. The sets $R^+(W)$ and $R^-(W)$ can be computed in linear time and hence $R^+(W) \cap R^-(W)$ can be computed in linear time. Resulting, P being a cut path can be verified in $O(m)$ time. ◀

The remainder structure exhibits two more properties useful for other problems.

▶ **Lemma 14** (Extended witness property)**.** *Let $P = (p_1, \ldots, p_\ell)$ a cut path of length $\ell - 1 \geq 1$. Let $u \in S^+(P)$ and $v \in S^-(P)$ be nodes. It holds that all u-v walks contain P as subwalk.*

▶ **Lemma 15** (Nonemptiness property)**.** *For a cut path $P = (p_1, \ldots, p_\ell)$ of length $\ell - 1 \geq 1$, it holds that $p_1 \in S^+(P)$ and $p_\ell \in S^-(P)$.*

3 Linear-time verifiable characterisations of (multi-)safe walks

We apply the remainder structure of a cut path to give easily and efficiently verifiable characterisations of safe and multi-safe walks. From here on we assume that our strongly connected graph $G = (V, E)$ is not a cycle. All missing formal proofs are in Appendix B.

First note that the univocal extension of a walk always needs to be traversed when traversing the walk itself with a closed walk. The *univocal extension* $U(W) = (l_1, \ldots, l_a, w_1, \ldots, w_\ell, r_1, \ldots, r_b)$ of W is a maximal walk where $l_2, \ldots, l_a, w_1$ are not joins and $w_\ell, r_1, \ldots, r_{b-1}$ are not splits.

▶ **Lemma 16** (Safety of univocal extensions). *The univocal extension $U(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

Since walks are (not necessarily maximal) univocal extensions of their cores, we get the following property useful for characterising safe and multi-safe walks.

▶ **Lemma 17** (Safety of cores). *The core $C(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

The difficulty in characterising safe and multi-safe walks lies in characterising interleaved walks. Non-interleaved walks are very simple to handle.

▶ **Lemma 18** (Non-interleaved walks). *A non-interleaved walk $W = (w_1, \dots, w_\ell)$ with core $C(W) = (w_j, \dots, w_i)$, $j + 2 \leq i$, is both safe and multi-safe. This property can be verified in $O(m)$ time.*

Safe walks can be characterised as follows.

▶ **Theorem 4** (Safety characterisation and verification). *Let W be a walk, and let $C(W)$ be its core. W is safe if and only if it is a non-interleaved walk or $C(W)$ is a cut path. This property can be verified in $O(m)$ time.*

Proof. If W is non-interleaved, the statement follows by Lemma 18.

If the core $C(W)$ of W is not a cut path, then by Lemma 9, there is a $\textsc{tail}(C(W))$-$\textsc{head}(C(W))$ walk that does not have $C(W)$ as subwalk. This can be used to replace all occurrences of $C(W)$ in a node-covering closed walk. To ensure that the resulting walk C covers all nodes, we insert two closed walks C_1 and C_2 into it, constructed as follows. Let v be the last inner split in $C(W)$ and v' one of its successors outside of $C(W)$. C_1 starts in $\textsc{tail}(C(W))$ and walks $C(W)$ until v and then v'. From v' it walks back to $\textsc{tail}(C(W))$, which is possible because the graph is strongly connected, and also possible without $C(W)$ as subwalk since it ends in $\textsc{tail}(C(W))$. C_2 is constructed symmetrically through $\textsc{head}(C(W))$. Since W is interleaved, it holds that $C(W)$ is interleaved, so C_1 and C_2 together cover $C(W)$. Further, because C contains $\textsc{tail}(C(W))$ and $\textsc{head}(C(W))$, we can insert C_1 and C_2 into it. Since the insertions happen at the first/last node of $C(W)$ they do not introduce it as subwalk. Thus, W is not safe.

If the core $C(W)$ of W is a cut path, then there is a pair of nodes $u, v \in V$ such that all u-v walks have $C(W)$ as a subwalk. A closed node-covering walk contains a subwalk between each pair of nodes, so also between u and v. Therefore, each closed node-covering walk contains $C(W)$ as subwalk, so by Lemma 17, W is safe.

Finally, the core $C(W)$ can be computed in linear time, and by Theorem 1 it can be checked for being a cut path in linear time. Hence, verifying if W is safe takes $O(m)$ time. ◀

Multi-safe walks can be characterised as follows. See Figure 3 for an example.

▶ **Theorem 5** (Multi-safety characterisation and verification). *Let W be a walk, and let $C(W)$ be its core. If W is non-interleaved, then it is multi-safe. Otherwise, it is multi-safe if and only if it is safe and any of $G[\bar{S}(C(W))], G[R^+(C(W))]$ or $G[R^-(C(W))]$ contains an SCC of size one. This property can be verified in $O(m)$ time.*

Proof. If W is non-interleaved, the statement follows by Lemma 18.

If $C(W)$ is not a cut path, then by Theorem 4, W is not safe, so it is also not multi-safe.

If none of $G[\bar{S}(C(W))], G[R^+(C(W))]$ and $G[R^-(C(W))]$ contain an SCC of size one, then all nodes can be covered by proper closed walks that do not leave a single one of $G[\bar{S}(C(W))], G[R^+(C(W))]$ or $G[R^-(C(W))]$. By definition, $G[\bar{S}(C(W))]$ contains no

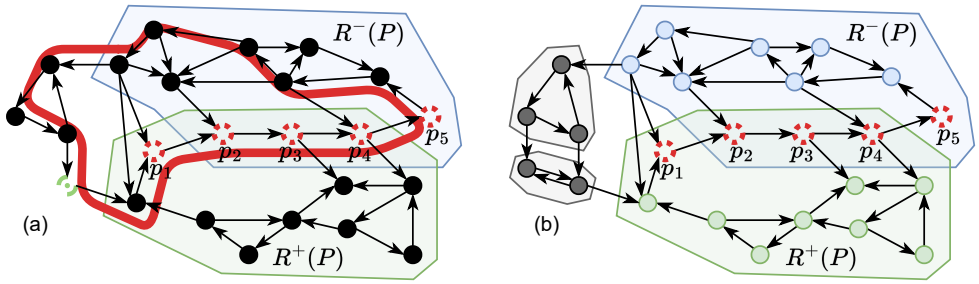


Figure 3 (a) A walk $(p_1, p_2, p_3, p_4, p_5)$ (as red dashed nodes) that is multi-safe, because the inner component of its remainder structure contains an SCC of size one (the green dashed node). The red walk is an example proper closed walk that covers the marked SCC. (b) Neither the restricted reachabilities nor the inner component contain an SCC of size one, so the two SCCs in the inner component and the restricted reachabilities can be covered by separate closed walks, without traversing the walk.

node of $C(W)$. Further, by Lemma 11, $G[R^+(C(W))]$ misses the last node of $C(W)$ and $G[R^-(C(W))]$ misses the first node of $C(W)$. Thus, none of the proper closed walks can have $C(W)$ as subwalk, so W is not multi-safe.

If $C(W)$ is a cut path and $G[\bar{S}(C(W))]$ contains an SCC of size one, then this SCC cannot be covered by a proper closed walk without leaving $\bar{S}(C(W))$. By Lemmas 11 and 15, when leaving $\bar{S}(C(W))$, a walk ends up in $S^+(C(W))$, and when entering $\bar{S}(C(W))$, a walk comes from $S^-(C(W))$. By Lemma 14, walking from $S^+(C(W))$ to $S^-(C(W))$ requires having $C(W)$ as a subwalk. Therefore, by Lemma 17, W is multi-safe.

If $C(W)$ is a cut path and $G[R^+(C(W))]$ contains an SCC of size one, then by Lemma 12, $R^+(C(W))$ contains exactly one node v. By definition, v is not contained in $G[\bar{S}(C(W))]$ and by Lemma 15, v is not contained in $G[R^-(C(W))]$. So to cover v, a proper closed walk X would leave $R^+(C(W))$. By Lemmas 11 and 15, this implies that X would have a subwalk from $\textsc{tail}(C(W))$ to $\textsc{head}(C(W))$. By Lemma 9 this implies X has $C(W)$ as subwalk, so by Lemma 17, W is multi-safe. By symmetry, if $C(W)$ is a cut path and $G[R^-(C(W))]$ contains an SCC of size one, then W is multi-safe.

Finally, the core $C(W)$ can be computed in linear time, and by Theorem 1 it can be checked for being a cut path in linear time. Moreover, the strongly connected components can be computed in linear time by [18]. Hence, W being safe can be verified in $O(m)$ time. ◀

4 Amortised enumeration of all maximal multi-safe walks

In this section we give the algorithm supporting Theorem 6. Since every multi-safe walk is also safe, by definition, we start by enumerating all safe walks, and then finding their subwalks that are also multi-safe, using further properties of their remainder structure, including an additional monotonicity property of it.

Enumerating all maximal safe walks. Recall that all walks without forbidden paths can be enumerated in time $O(mn)$ with the algorithm from Cairo et al. [5]. From these, it is simple to get the safe walks using the following property from [20, Theorem 3]:

▶ **Lemma 19** (Safe walks [20]). *A walk is safe if and only if it has no forbidden paths and has no cut arc.*

Algorithm 1 MultiSafe.

Input: Strongly connected graph $G = (V, E)$, all maximal safe walks $\mathcal{W}$.
Output: All maximal multi-safe walks $\mathcal{W}'$.

```
1  W' ← () // empty list
2  for W ∈ W do
3      if W is a non-interleaved walk then
4          append W to W', continue
5      (w_1, ..., w_ℓ) ← C(W), start ← 1, end ← 1
6      while end ≤ ℓ do
7          if (w_start, ..., w_end) is multi-safe then
8              end ← end + 1
9          else
10             append U((w_start, ..., w_{end-1})) to W'
11             start ← start + 1
12     while (w_start, ..., w_{end-1}) is not multi-safe do
13         start ← start + 1
14     append U((w_start, ..., w_{end-1})) to W'
15 Remove duplicates and subwalks from W' using e.g. a suffix tree
```

We compute the cut arcs in linear time [14] and then break all the walks without forbidden paths at arcs that are not cut arcs. Then we remove duplicates and proper subwalks from the result using standard methods and suffix trees. See Appendix A for details.

▶ **Lemma 20** (Enumeration of safe walks). *All maximal safe walks can be enumerated in $O(mn)$ time.*

Enumerating all maximal multi-safe walks. Using Theorem 5, we are able to derive an algorithm that enumerates maximal multi-safe walks. It works by iterating all safe walks and enumerating all their maximal multi-safe subwalks. We start with the subwalk of a safe walk consisting of its first core arc, and then extend it to the right whenever it is safe, while removing its first node (only from the subwalk, not the graph) whenever it is not safe. Afterwards, we deduplicate and remove proper subwalks again, as described in Appendix A. See Algorithm 1 for pseudocode. To analyse the runtime of this algorithm without amortisation, we use the following property.

▶ **Lemma 21** (Amount of interleaved safe walks). *A strongly connected graph contains at most $O(n)$ interleaved maximal safe walks.*

By Cairo et al. [5], the total length of the maximal walks without forbidden paths is $O(mn)$, and hence the total length of the maximal safe walks is $O(mn)$. Thus, if there are no interleaved walks, the algorithm runs in $O(mn)$ time. However, there may be up to $O(n)$ interleaved walks, and since their cores can not have cycles, for each interleaved walk, Algorithm 1 performs up to $O(n)$ multi-safety checks. Each such check takes $O(m)$ by Theorem 5. Further, in a strongly connected graph that is not a cycle, a univocal extension increases the length of a walk by at most $O(n)$. So the total length of the maximal multi-safe walks produced by a maximal safe walk is $O(n^2)$. Hence, including linear-time deduplication and removal of proper subwalks, the runtime of Algorithm 1 is $O(mn)$ for non-interleaved walks plus $O(mn + n^2)$ for each of the up to $O(n)$ interleaved walks. Summed up, that is $O(mn^2 + n^3) = O(mn^2)$. However, using amortisation, we get $O(mn)$, as shown below.

▶ **Theorem 22** (Amortised computation). *Let $P = (p_1, \dots, p_\ell)$ be a cut path and let $P' = (p_i, \dots, p_j)$ be a subwalk of P with $i < j$. If p_{j-1} and $p_{\ell-1}$ are splits, then $R^+(P') \subseteq R^+(P)$. If p_2 and p_{i+1} are joins, then $R^-(P') \subseteq R^-(P)$.*

Proof. Let p_{j-1} and $p_{\ell-1}$ be splits. Let $v \notin R^+(P)$. Then by definition each p_1-v walk contains $(p_{\ell-1}, p_\ell)$. Further, since $i \neq \ell$ and by Lemma 8 it holds that P is an open path, it holds that $p_i \in R^+(P)$. So each p_i-v walk contains $(p_{\ell-1}, p_\ell)$. Further, since P is a cut path, also its subwalk $P_C = (p_i, \dots, p_{\ell-1})$ is a cut path. By Lemma 9, this implies that each p_i-$p_{\ell-1}$ walk has P_C as subwalk, which especially means that it contains (p_{j-1}, p_j). Therefore, each p_i-v walk contains (p_{j-1}, p_j), so $v \notin R^+(P')$. Resulting, $R^+(P') \subseteq R^+(P)$.

By symmetry, if p_2 and p_{i+1} are joins, then $R^-(P') \subseteq R^-(P)$. ◀

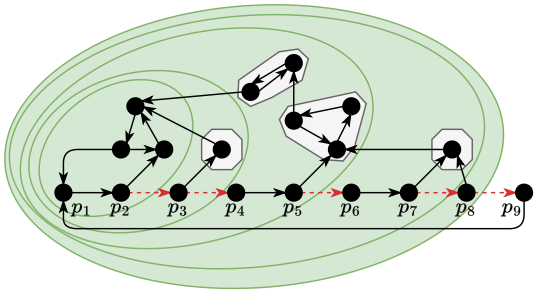


Figure 4 Amortised computation of $R^+(P')$ where P' is a prefix of $P = (p_1, \dots, p_9)$. The red dashed arcs mark the ends of the prefixes P'. They are also the only arc leaving their respective $R^+(P')$. The SCCs of $G \setminus R^+(P')$ are enclosed in grey areas, and the different $R^+(P')$ are enclosed in green areas.

An example for Theorem 22 is given in Figure 4. Using Theorem 22 to implement Algorithm 1, we can answer all multi-safety queries for a single walk $W \in \mathcal{W}$ in $O(m)$ time. For this, we observe that the boundaries of the subwalk W' of $C(W)$ that is tested for safety only get shifted towards the end of $C(W)$. Further, whenever W' contains no split or no join as inner node, then it is either a non-interleaved walk or a univocal extension of a cut arc (by Lemma 19) and hence safe. So we only need to compute the remainder structure for subwalks that contain at least one split and one join. For such subwalks, by Theorem 22 it holds that $R^+(W')$ is monotonically increasing within each execution of the body of the loop in Algorithm 1, and $R^-(W')$ is monotonically decreasing. Therefore, all $R^+(W')$ and $R^-(W')$ can be precomputed by computing them for all prefixes and suffixes of W that contain splits or joins, respectively. The computation of the $R^+(W')$ is done in forward order, and for each node, the search is started from the node itself and nodes visited by earlier searches are pruned. With this strategy, the $R^+(W')$ of all prefixes W' of $C(W)$ can be computed in $O(m)$ time. By computing the $R^-(W')$ in reverse order, they can also be computed in $O(m)$ time.

To check the safety of all subwalks W' based on the precomputed remainder structure in linear time, note the following. When executing the body of the loop in Algorithm 1, $R^+(W')$ only increases and $R^-(W')$ only decreases for the relevant subwalks (those that are interleaved). Further, by Lemma 12, $G[R^+(W')]$ and $G[R^-(W')]$ are strongly connected if they have a split as last inner node or a join as first inner node, respectively. If they are not strongly connected then the inner nodes after the last inner split or the inner nodes before the first inner join, respectively, form SCCs of size 1. Therefore, the query if $G[R^+(W')]$ or $G[R^-(W')]$ contain an SCC of size 1 can be answered in constant time if the size of $R^+(W')$

and $R^-(W')$ as well as the joins and splits of W' are tracked within the body of the loop in Algorithm 1. Further, by definition of $R^+(W')$ and $R^-(W')$, when $R^+(W')$ grows, SCCs of the inner component enter it as a whole, so by growing $R^+(W')$, the remaining SCCs of the inner component remain unchanged. Symmetrically, when shrinking $R^-(W')$, the new nodes do not alter the existing SCCs in the inner component, so to check whether an SCC of size one was added, only the SCCs of the induced subgraph of the newly added nodes need to be computed. Since the SCCs of a graph can be computed in linear time [18], this means that tracking whether there are SCCs of size 1 in any of $G[\bar{S}(W')]$, $G[R^+(W')]$ or $G[R^-(W')]$ can be implemented in $O(m)$ time per execution of the body of the loop in Algorithm 1.

Hence, we get a runtime of $O(m)$ for the multi-safety checks of each of the $O(n)$ interleaved safe walks. However, each interleaved maximal safe walk may produce maximal multi-safe walks of total length $O(n^2)$. But, amortising over all interleaved maximal safe walk, we see that each core of a multi-safe walk is uniquely identified by its first and last node, since otherwise it would not be a cut path by Lemma 9. So, we can use a flag for each pair of nodes and thus avoid repetitions in multi-safe walks produced by interleaved maximal safe walks in constant time. In total, at most $O(n^2)$ such checks happen, so the interleaved walks take $O(mn + o)$ time, where o is the total length of the interleaved walks. This results in a total time of $O(mn + o)$ for Algorithm 1, and since there are at most n interleaved safe walks by Lemma 21, $o \in O(n^3)$. By reporting the maximal multi-safe walks as start and end index in their respective maximal safe walks, we get an output size of $O(n^2)$ interleaved maximal multi-safe walks, plus $O(mn)$ non-interleaved maximal multi-safe walks. So if we are only interested in identifying maximal multi-safe walks and not in an explicit enumeration, we have an algorithm that runs in $O(mn)$ time.

▶ **Theorem 6** (Enumerating maximal multi-safe walks). *All maximal multi-safe walks can be identified in $O(mn)$ time and enumerated in $O(mn + o)$ time, where o is the total length of the output and it holds that $o \in O(n^3)$.*

5 Conclusions and future work

We introduced cut paths as a generalisation of cut arcs, as well as the remainder structure of cut paths. Using properties of the remainder structure, we applied cut paths to some well-studied reachability problems from bioinformatics. In the same way as the remainder structure gave a simple YES-certificate for a path to be a cut path (Theorem 1), the remainder structure led to easily verifiable YES-certificates for walk safety (Theorem 4) and multi-safety (Theorem 5), which were open questions. By proving an additional monotonicity property (Theorem 22), we improved the state-of-the-art of enumeration of all maximal multi-safe walks (Theorem 6).

There are central structural questions about cut paths that remain open. It is known that there are at most $O(n)$ cut arcs which can be enumerated in $O(m)$ time [14]. But for cut paths, there is no known upper bound to their amount or total length, and it is open how they can be enumerated efficiently. Further, it is open how they can overlap and intersect.

For our applications, it is open if the total length of safe and multi-safe walks is really $O(mn)$, or if our enumeration algorithm for safe and multi-safe walks is not optimal. Further, it is open if there is a linear output-sensitive algorithm for safe or multi-safe walks, as the one for walks without forbidden paths from [6].

References

1 Nidia Obscura Acosta, Veli Mäkinen, and Alexandru I. Tomescu. A safe and complete algorithm for metagenomic assembly. *Algorithms for Molecular Biology*, 13(1):3:1–3:12, 2018. doi:10.1186/s13015-018-0122-7.

2 Nidia Obscura Acosta and Alexandru I. Tomescu. Simplicity in eulerian circuits: Uniqueness and safety. *arXiv*, abs/2208.08522, 2022. arXiv:2208.08522.

3 Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-Space Reduction via Essential Vertices. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.30.

4 Massimo Cairo, Shahbaz Khan, Romeo Rizzi, Sebastian Schmidt, and Alexandru I. Tomescu. Safety in *s-t* Paths, Trails and Walks. *Algorithmica*, 84:719–741, 2022. doi:10.1007/s00453-021-00877-w.

5 Massimo Cairo, Paul Medvedev, Nidia Obscura Acosta, Romeo Rizzi, and Alexandru I. Tomescu. An Optimal $O(nm)$ Algorithm for Enumerating All Walks Common to All Closed Edge-covering Walks of a Graph. *ACM Trans. Algorithms*, 15(4):48:1–48:17, 2019. doi:10.1145/3341731.

6 Massimo Cairo, Romeo Rizzi, Alexandru I. Tomescu, and Elia C. Zirondelli. Genome assembly, from practice to theory: Safe, complete and linear-time. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 43:1–43:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

7 Marie Costa. Persistency in maximum cardinality bipartite matchings. *Oper. Res. Lett.*, 15(3):143–9, 1994. doi:10.1016/0167-6377(94)90049-3.

8 Wojciech Fraczak, Loukas Georgiadis, Andrew Miller, and Robert E Tarjan. Finding dominators via disjoint set union. *Journal of Discrete Algorithms*, 23:2–20, 2013.

9 Loukas Georgiadis, Giuseppe F Italiano, Luigi Laura, and Nikos Parotsidis. 2-edge connectivity in directed graphs. *ACM Transactions on Algorithms (TALG)*, 13(1):1–24, 2016.

10 Loukas Georgiadis, Giuseppe F Italiano, Luigi Laura, and Nikos Parotsidis. 2-vertex connectivity in directed graphs. *Information and Computation*, 261:248–264, 2018.

11 Loukas Georgiadis, Giuseppe F Italiano, and Nikos Parotsidis. Strong connectivity in directed graphs under failures, with applications. *SIAM Journal on Computing*, 49(5):865–926, 2020.

12 Dan Gusfield. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News*, 28(4):41–60, 1997.

13 P. L. Hammer, P. Hansen, and B. Simeone. Vertices belonging to all or to no maximum stable sets of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):511–522, 1982. doi:10.1137/0603052.

14 Giuseppe F Italiano, Luigi Laura, and Federico Santaroni. Finding strong bridges and strong articulation points in linear time. *Theoretical Computer Science*, 447:74–84, 2012.

15 Giuseppe F Italiano, Nikos Parotsidis, and Eugenia Perekhodko. What's inside a bow-tie: Analyzing the core of the web and of social networks. In *Proceedings of the 2017 International Conference on Information System and Data Mining*, pages 39–43, 2017.

16 Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908, 2009.

17 Amatur Rahman and Paul Medvedev. Assembler artifacts include misassembly because of unsafe unitigs and underassembly because of bidirected graphs. *Genome Research*, 2022. doi:10.1101/gr.276601.122.

18 Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

19 Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6(2):171–185, 1976.

20 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017. Preliminary version appeared in RECOMB 2016.

21 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

22 Pencho Yordanov and Jörg Stelling. Efficient manipulation and generation of kirchhoff polynomials for the analysis of non-equilibrium biochemical reaction networks. *Journal of the Royal Society Interface*, 17(165):20190828, 2020.

A Deduplication and removal of proper subwalks in linear time

Algorithm 2 RemoveDuplicatesAndProperSubwalks.

Input: List of walks $\mathcal{W} = (W_1, \ldots, W_{|\mathcal{W}|})$.
Output: Set of walks $\mathcal{W}'$ containing one copy of each unique walk in $\mathcal{W}$ that is not a proper subwalk of another walk in $\mathcal{W}$.

```
1 Sort 𝒲 by length descending
2 Build string S = W_1$W_2$...$W_|𝒲|$
3 Build suffix tree T on S
4 𝒲' ← ∅
5 for W_i ∈ 𝒲 do
6     (l, r) ← first occurrence of W_i in S
7     if (l, r) = coordinates of W_i in S then
8         𝒲' ← 𝒲' ∪ {W_i}
```

The removal of duplicates and subwalks is implemented in linear time using a suffix tree in Algorithm 2. It works by sorting the walks by length descending and only reporting a walk if it is no subwalk of a previous walk, meaning if it is no proper subwalk of a previous walk and it is the first occurrence of itself.

▶ **Lemma 23** (Deduplication). *Algorithm 2 is correct and works in time linear in the total length of $\mathcal{W}$.*

Proof. The algorithm sorts the walks by length descending and then reports walks only if their string of nodes does not occur any earlier than themselves in S. This reports only one copy of each walk since only the first occurrence of a walk in S is reported. Moreover, if a walk is a proper subwalk of another, then that subwalk will occur earlier in S, so the subwalk will never be reported. Therefore, Algorithm 2 is correct.

For the runtime, let $||\mathcal{W}||$ be the total length of $\mathcal{W}$. Sorting $\mathcal{W}$ by length descending can be done by bucket sort with $||\mathcal{W}||$ buckets containing each a dynamic array. Then it runs in time linear in $||\mathcal{W}||$. Building S and the suffix tree is linear in $||\mathcal{W}||$ [21]. The total cost of checking for the first occurrences of all W_i in S is linear in $||\mathcal{W}||$ [12]. Checking the coordinates of W_i can be done by storing the coordinates for each string while constructing S in a lookup table indexed by i. Then the branch runs in constant time. Therefore, Algorithm 2 runs in time linear in $||\mathcal{W}||$. ◀

B Omitted proofs

This section contains the proofs omitted from the main matter.

▶ **Lemma 8** (Open path property). *A cut path is an open path.*

Proof. Let W be a walk in a strongly connected graph $G = (V, E)$ that is not an open path, i.e. it repeats some node v. Then we can construct the walk $W' \neq W$ by removing all v-v subwalks from W (if W is a v-v walk, then W' is a single node). Assume for a contradiction that W was a cut path. Then there would be a pair of nodes $u, w \in V$ such that every u-w walk in G would have W as subwalk. But we could replace all occurrences of W by W' in any u-w walk, resulting in u-w walks that do not have W as subwalk. By contradiction, W is not a cut path. ◀

▶ **Lemma 9** (Witness property). *A walk W is a cut path if and only if it is subwalk of all $\textsc{tail}(W)$-$\textsc{head}(W)$ walks.*

Proof. Let $G = (V, E)$ be the strongly connected graph that contains W. If there is a $\textsc{tail}(W)$-$\textsc{head}(W)$ walk without W as subwalk, then for any pair of nodes $u, v \in V$, any u-v walk that contains W as subwalk can be transformed into a u-v walk that does not contain W as subwalk. Then W is not a cut path.

If all $\textsc{tail}(W)$-$\textsc{head}(W)$ walks have W as subwalk, then by definition W is a cut path. ◀

▶ **Lemma 14** (Extended witness property). *Let $P = (p_1, \ldots, p_\ell)$ a cut path of length $\ell - 1 \geq 1$. Let $u \in S^+(P)$ and $v \in S^-(P)$ be nodes. It holds that all u-v walks contain P as subwalk.*

Proof. Let $W_1W_2W_3$ be a $\textsc{tail}(P)$-$\textsc{head}(P)$ walk where W_1 is a $\textsc{tail}(P)$-u walk, W_2 is a u-v walk, and W_3 is a v-$\textsc{head}(P)$ walk. By Lemma 9, $W_1W_2W_3$ must have P as a subwalk since P is a cut path. By definition, $\textsc{tail}(P)$ reaches all nodes in $S^+(P) \subseteq R^+(P)$ without using $(p_{\ell-1}, p_\ell)$, so we can choose W_1 without using P as subwalk. Also, all nodes in $S^-(P) \subseteq R^-(P)$ reach $\textsc{head}(P)$ without using (p_1, p_2), so we can choose W_3 without using P as subwalk. Further, by definition, $u, v \notin \bar{P}(P)$, so by Theorem 1, neither u nor v are inner nodes of P. Therefore, concatenating $W_1W_2W_3$ cannot introduce P as a subwalk by crossing the boundary between either W_1 and W_2 or W_2 and W_3. Concluding, W_2 has P as subwalk. ◀

▶ **Lemma 15** (Nonemptiness property). *For a cut path $P = (p_1, \ldots, p_\ell)$ of length $\ell - 1 \geq 1$, it holds that $p_1 \in S^+(P)$ and $p_\ell \in S^-(P)$.*

Proof. By definition, $p_1 \in R^+(P)$ and $p_\ell \in R^-(P)$. It holds that $p_\ell \notin R^+(P)$ and $p_1 \notin R^-(P)$, since by Lemma 11 any of the two implies a $\textsc{tail}(P)$-$\textsc{head}(P)$ walk without P as subwalk, which by Lemma 9 contradicts P being a cut path. ◀

▶ **Lemma 16** (Safety of univocal extensions). *The univocal extension $U(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

Proof. Let $U(W) = (u_1, \ldots, u_\ell)$ and $W = (u_i, \ldots, u_j)$. Any closed walk having W as subwalk can only enter W via $(u_1, \ldots, u_i)$ since $(u_2, \ldots u_i)$ has no joins, and it can only leave W via $(u_j, \ldots, u_\ell)$ since $(u_j, \ldots, u_{\ell-1})$ has no splits. Hence, $U(W)$ is safe if and only if W is safe, and $U(W)$ is multi-safe if and only if W is multi-safe. ◀

Since walks are (not necessarily maximal) univocal extensions of their cores, we get the following property useful for characterising safe and multi-safe walks.

▶ **Lemma 17** (Safety of cores). *The core $C(W)$ of a walk W is safe if and only if W is safe. It is multi-safe if and only if W is multi-safe.*

Proof. Let $W = (w_1, \ldots, w_\ell)$ and $C(W) = (w_i, \ldots, w_j)$. By definition, independent of W being interleaved or non-interleaved, it holds that $(w_2, \ldots, w_i)$ contains no joins and $(w_j, \ldots, w_{\ell-1})$ contains no splits. Hence, W is subwalk of $U(C(W))$, so $U(W) = U(C(W))$. By Lemma 16, W is safe $\iff$ $U(W)$ is safe $\iff$ $U(C(W))$ is safe $\iff$ $C(W)$ is safe. The same equivalence holds for the multi-safe property. ◀

▶ **Lemma 18** (Non-interleaved walks). *A non-interleaved walk $W = (w_1, \ldots, w_\ell)$ with core $C(W) = (w_j, \ldots, w_i)$, $j + 2 \leq i$, is both safe and multi-safe. This property can be verified in $O(m)$ time.*

Proof. Since $j + 2 \leq i$, it holds that $C(W)$ has an inner node that can only be covered by a closed walk by using $C(W)$ as subwalk. Hence, by Lemma 17, it holds that W is both safe and multi-safe.

Finally, checking a walk for being interleaved can be done in $O(m)$ time. If $\ell > 3m$, then if the graph is a cycle, W is non-interleaved. If the graph is not a cycle, then assume for a contradiction that W is non-interleaved. Then W contains a join-free or split-free subwalk of length $m + 1$. Such a subwalk contains a cycle, because $m + 1 > m$. And such a cycle is then join-free or split-free. This contradicts the graph not being a cycle or the graph being strongly connected. ◀

▶ **Lemma 21** (Amount of interleaved safe walks). *A strongly connected graph contains at most $O(n)$ interleaved maximal safe walks.*

Proof. Note that each interleaved walk with a core of length 1 is safe only if its core is a cut arc. So there are at most $O(n)$ interleaved safe walks with a core of length 1. Further, by Cairo et al. [6], it holds that there are at most $O(n)$ walks without forbidden paths that are interleaved with a core of length at least 2. Assume for a contradiction that any such walk W could contain more than one non-cut arc. The non-cut arcs cannot be outside of the core, since all non-core arcs are the only outgoing or the only incoming arcs of some node. If there are at least two non-cut arcs in the core, we can construct an arc-covering closed walk W' that does not contain W. Start with any arc-covering closed walk and repeat it twice. In the first repetition, replace any occurrence of the first non-cut arc of W with a walk that avoids the non-cut arc. And in the second repetition, replace any occurrence of the last non-cut arc of W with a walk that avoids the non-cut arc. Such avoiding walks exist since the avoided arcs are not cut arcs. Further, by avoiding an arc of W, they do not have W as subwalk. The resulting walk W' is arc-covering and closed, but does not have W as subwalk. Hence, W has a forbidden path by Lemma 19. Finally, each walk without forbidden path that is interleaved with a core of length at least 2 produces at most two interleaved safe walks. Since non-interleaved walks without forbidden path cannot be broken into interleaved ones, there are at most $O(n)$ interleaved maximal safe walks. ◀

Geometric Amortization of Enumeration Algorithms

Florent Capelli
Université de Lille, CNRS, Inria, Centrale Lille, UMR 9189 – CRIStAL, F-59000 Lille, France

Yann Strozecki
Université Paris Saclay, UVSQ, DAVID, France

Abstract

In this paper, we introduce a technique we call geometric amortization for enumeration algorithms, which can be used to make the delay of enumeration algorithms more regular with little overhead on the space it uses. More precisely, we consider enumeration algorithms having incremental linear delay, that is, algorithms enumerating, on input x, a set $A(x)$ such that for every $t \leq \sharp A(x)$, it outputs at least t solutions in time $O(t \cdot p(|x|))$, where p is a polynomial. We call p the incremental delay of the algorithm. While it is folklore that one can transform such an algorithm into an algorithm with maximal delay $O(p(|x|))$, the naive transformation may use exponential space. We show that, using geometric amortization, such an algorithm can be transformed into an algorithm with delay $O(p(|x|)\log(\sharp A(x)))$ and space $O(s\log(\sharp A(x)))$ where s is the space used by the original algorithm. In terms of complexity, we prove that classes DelayP and IncP_1 with polynomial space coincide.

We apply geometric amortization to show that one can trade the delay of flashlight search algorithms for their average delay up to a factor of $O(\log(\sharp A(x)))$. We illustrate how this tradeoff is advantageous for the enumeration of solutions of DNF formulas.

2012 ACM Subject Classification Theory of computation → Complexity classes

Keywords and phrases Enumeration, Polynomial Delay, Incremental Delay, Amortization

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.18

Related Version *Full Version*: `https://arxiv.org/abs/2108.10208`

Supplementary Material
InteractiveResource (Visualization): `http://florent.capelli.me/coussinet/`

Funding This work was supported by project ANR KCODA, ANR-20-CE48-0004.

Acknowledgements We would like to thank anonymous reviewers for helpful comments on previous version of the paper and Nofar Carmeli and Stefan Mengel for proofreading earlier version of the paper.

1 Introduction

An enumeration problem is the task of listing a set of elements without redundancies. It is an important and old class of problems: the Baguenaudier game [28] from the 19th century can be seen as the problem of enumerating integers in Gray code order. Ruskey even reports [33] on thousand-year-old methods to list simple combinatorial structures such as the subsets or the partitions of a finite set. Modern enumeration algorithms date back to the 1970s with algorithms computing circuits or spanning trees of a graph [36, 32], while fundamental complexity notions for enumeration have been formalized 30 years ago by Johnson, Yannakakis and Papadimitriou [23]. The main specificity of enumeration problems is that the size of the enumerated set is typically exponential in the size of the input. Hence, a problem is considered tractable and said to be *output polynomial* when it can be solved in

© Florent Capelli and Yann Strozecki;
licensed under Creative Commons License CC-BY 4.0

time polynomial in the size of the *input and the output*. This measure is relevant when one wants to generate and store all elements of a set, for instance to build a library of objects later to be analyzed by experts, as it is done in biology, chemistry, or network analytics [2, 6, 9].

For most problems, the set to enumerate is too large, or may not be needed in its entirety. It is then desirable to efficiently generate a part of the set for statistical analysis or on the fly processing. In this case, a more relevant measure of the complexity and hence of the quality of the enumeration algorithm is its *delay*, that is, the time spent between two consecutive outputs. One prominent focus has been to design algorithms whose delay is bounded by a polynomial in the size of the input. Problems admitting such algorithms constitute the class DelayP and many problems are in this class, for example the enumeration of the maximal independent sets of a graph [23], or answer tuples of restricted database queries [19] (see [39] for many more examples).

It also happens that new elements of the output set, also called solutions, become increasingly difficult to find. In this case, polynomial delay is usually out of reach but one may still design algorithms with *polynomial incremental time*. An algorithm is in polynomial incremental time if for every i, the delay between the output of the i^{th} and the $(i+1)^{\text{st}}$ solution is polynomial in i and in the size of the input. Such algorithms naturally exist for saturation problems: given a set of elements and a polynomial time function acting on tuples of elements, produce the closure of the set by the function. One can generate such a closure by iteratively applying the function until no new element is found. As the set grows bigger, finding new elements becomes harder. The best algorithm to generate circuits of a matroid uses a closure property of the circuits [24] and is thus in polynomial incremental time. The fundamental problem of generating the minimal transversals of a hypergraph can also be solved in quasi-polynomial incremental time [21, 8] and some of its restrictions in polynomial incremental time [20]. In this paper, the class of problems which can be solved with polynomial incremental time is denoted by UsualIncP.

While the delay is a natural way of measuring the quality of an enumeration algorithm, it might sometimes be too strong of a restriction. Indeed, if the enumeration algorithm is used to generate a subset of the solutions, it is often enough to have guarantees that the time needed to generate i solutions is reasonable for every i. For example, one could be satisfied with an algorithm that has the property that after a time $i \cdot p(n)$, it has output at least i solutions, where p is a polynomial and n the input size. In this paper, we refer to this kind of algorithm as IncP_1-enumerators[1] and call $p(n)$ *the incremental delay* of the algorithm.

While polynomial delay enumerators are IncP_1-enumerator, the converse is not true. Indeed, IncP_1-enumerators do not have to output their solutions regularly. Take for example an algorithm that, on an input of size n, outputs 2^n solutions in 2^n steps, then nothing for 2^n steps and finally outputs the last solution. It can be readily verified that this algorithm outputs at least i solutions after $2i$ steps for every $i \leq 2^n + 1$, and it is thus an IncP_1-enumerator. However, the delay of such an algorithm is not polynomial as the time spent between the output of the last two solutions is 2^n. Instead of executing the output instruction of this algorithm, one could store the solutions that are found in a queue. Then, every two steps of the original algorithm, one solution is removed from the queue and output. The IncP_1 property ensures that the queue is never empty when dequeued and we now have polynomial delay. Intuitively, the solutions being dense in the beginning, they are used to pave the gap until the last solution is found. While this strategy may always be applied to turn an IncP_1-enumerator into a polynomial delay algorithm, the size of the queue may

[1] The 1 in IncP_1 stands for the linear dependency of the incremental time in the number of solutions.

become exponential in the size of the input. In the above example, after the simulation of 2^n steps, 2^n solutions have been pushed into the queue but only 2^{n-1} of them are output, so the queue still contains 2^{n-1} solutions. Unfortunately, an algorithm using exponential space may not be feasible. Therefore, much effort has been devoted to ensure that polynomial delay methods run with polynomial space [27, 3, 15, 17, 10].

Contributions. The main result of this paper is a proof that the regularization of an IncP_1-enumerator may be done without exponentially increasing the space used. More formally, we show that the class $\mathrm{DelayP}^{\mathrm{poly}}$ of problems that can be solved by a polynomial delay and polynomial space algorithm is the same as the class $\mathrm{IncP}_1^{\mathrm{poly}}$ of problems that can be solved by a polynomial space IncP_1-enumerator. In other words, we prove $\mathrm{DelayP}^{\mathrm{poly}} = \mathrm{IncP}_1^{\mathrm{poly}}$, answering positively a question we raised in [11] and where only special cases were proven. Our result relies on a constructive method that we call *geometric amortization*. It turns any IncP_1-enumerator into a polynomial delay algorithm whose delay and space complexity are the incremental delay and space complexity of the original enumerator multiplied by a factor of $\log(S)$, where S is the number of solutions (Theorem 3). Interestingly, we also show that the total time can be asymptotically preserved.

We also apply geometric amortization to transform the average delay of many DelayP-enumerators into a guaranteed delay. Indeed, we show that some widely used algorithmic techniques to design DelayP algorithms also have an incremental delay that matches their average delay. Thus, using geometric amortization, we show that the delay of such an enumerator can be traded for their average delay multiplied by the logarithm of the number of solutions. We apply this result to an algorithm solving Π_{DNF} [12], the problem of listing the models of a DNF formula. This gives an algorithm with sublinear delay and polynomial memory, answering an open question of [12].

The main consequence of our result is that it makes proving that an enumeration problem is in $\mathrm{DelayP}^{\mathrm{poly}}$ easier as one does not have to design an algorithm with polynomial delay but only with polynomial incremental delay. One side-effect of our transformation however is that the order the solutions are output in is changed which may have some practical consequences when used. However, we do not see this as a downside. Actually, we do not believe our method should be used in practice as we cannot see any advantages of having an algorithm with polynomial delay over one with polynomial incremental delay, a notion that we find more natural. This opinion may not be shared by everyone and the main point of our result is to show that from a purely theoretical point of view, it actually does not matter as both notions are – and it came as a surprise to us – the same.

Related work. The notion of polynomial incremental delay is natural enough to have appeared before in the literature. In her PhD thesis [22], Goldberg introduced the notion of *polynomial cumulative delay*, which exactly corresponds to our notion of polynomial incremental delay. We however decided to stick to the terminology of [11]. Goldberg mentions on page 10 that one can turn a linear incremental algorithm into a polynomial delay algorithm but at the price of exponential space. She argues that one would probably prefer in practice incremental delay and polynomial space to polynomial delay and exponential space. Interestingly, she also designs for every constant k, an IncP_1-algorithm with polynomial space to enumerate on input n, every graph that is k-colorable (Theorem 15 on page 112). She leaves open the question of designing a polynomial delay and polynomial space algorithm for this problem, which now comes as a corollary of our theorem applied to her IncP_1-algorithm.

In [37], Tziavelis, Gatterbauer and Riedewald introduce the notion of *Time-To-k* to describe the time needed to output the k best answers of a database query for every k. They design algorithms having a Time-To-k complexity of the form $\text{poly}(n)k$ where n is the size of the input, which hence corresponds to the notion of IncP_1. They argue that delay is sufficient but not necessary to get good Time-to-k complexity and they argue that in practice, having small Time-to-k complexity is better than having small delay. Observe however that in their case, our method does not apply well since they are interested in the *best* answers, meaning that the order is important in this context. Our method does not preserve order.

Organization of the paper. In Section 2, we introduce enumeration problems, the related computational model and complexity measures. Section 3 presents different techniques to turn an IncP_1-enumerator into a DelayP-enumerator using a technique called geometric amortization. Interactive visualization of how geometric amortization works can be found at `http://florent.capelli.me/coussinet/`. In Section 3.3 we apply geometric amortization to incremental polynomial algorithms, showing that our result generalizes to the IncP_i hierarchy. Section 4 gives a method to transform many DelayP-enumerators of average delay $a(n)$ into a DelayP-enumerator with maximal delay $a(n)\log(K)$ where K is the number of solutions. We use it to obtain an algorithm for the problem of enumerating the models of a DNF formula. To outline the main ideas of our algorithms, they are presented using pseudocode with instructions to simulate a given Random Access Machin (RAM). The details on the complexity of using such instructions with minimal overhead are given in the appendix.

2 Preliminaries

Enumeration problems. Let Σ be a finite alphabet and Σ^* be the set of finite words built on Σ. We denote by $|x|$ the length of $x \in \Sigma^*$. Let $A \subseteq \Sigma^* \times \Sigma^*$ be a binary predicat. We write $A(x)$ for the set of y such that $A(x, y)$ holds. The enumeration problem Π_A is the function which associates $A(x)$ to x. The element x is called the *instance* or the *input*, while an element of $A(x)$ is called a *solution*. We denote the cardinality of a set $A(x)$ by $\sharp A(x)$.

A predicate A is said to be *polynomially balanced* if for all $y \in A(x)$, $|y|$ is polynomial in $|x|$. It implies that $\sharp A(x)$ is bounded by $|\Sigma|^{\text{poly}(|x|)}$. Let $\textsc{Check}{\cdot}A$ be the problem of deciding, given x and y, whether $y \in A(x)$. The class EnumP, a natural analogous to NP for enumeration, is defined to be the set of all problems Π_A where A is polynomially balanced and $\textsc{Check}{\cdot}A \in \text{P}$. More details can be found in [11, 35].

Computational model. In this paper, we use the Random Access Machine (RAM) model introduced by Cook and Reckhow [18] with comparison, addition, subtraction and multiplication as its basic arithmetic operations augmented with an $\texttt{OUTPUT}(i, j)$ operation which outputs the content of the values of registers $R_i, R_{i+1}, \ldots, R_j$ as in [4, 34] to capture enumeration problems. We use an hybrid between *uniform cost model* and *logarithmic cost model* (see [18, 1]): output, addition, multiplication and comparison are in constant time on values less than n, where n is the size of the input. In first-order query problems, it is justified by bounding the values in the registers by n times a constant [19, 4]. However, it is not practical for general enumeration algorithms which may store and access 2^n solutions and thus need to deal with large integers. Hence, rather than bounding the register size, we define the cost of an instruction to be the sum of the size of its arguments *divided by* $\log(n)$. Thus, any operation on a value polynomial in n can be done in constant time, but unlike in the usual uniform cost model, we take into account the cost of dealing with superpolynomial values.

A RAM M on input $x \in \Sigma^*$ produces a sequence of outputs $y_1, \dots, y_S$. The set of outputs of M is denoted by $M(x)$ and its cardinality by $\sharp M(x)$. We say that M solves Π_A if, on every input $x \in \Sigma^*$, $A(x) = M(x)$ and for all $i \neq j$ we have $y_i \neq y_j$, that is *no solution is repeated*. All registers are initialized with zero. The space used by the M is the sum of the bitsize of the integers stored in its registers, up to the register of the largest index accessed.

We denote by $T_M(x, i)$ the time taken by the machine M on input x before the i^{th} `OUTPUT` instruction is executed. When the machine is clear from the context, we drop the subscript M and write $T(x, i)$. The *delay* of a RAM which outputs the sequence $y_1, \dots, y_S$ is the maximum over all $i \leq s$ of the time spent between the generation of y_i and y_{i+1}, that is $\max_{1 \leq i \leq S} T(x, i+1) - T(x, i)$. The *preprocessing* is $T_M(x, 1)$, the time spent before the first solution is output. The *postprocessing* is the time spent between the output of the last solution and the end of the computation. To simplify the presentation, we assume that there is no postprocessing, that is, a RAM solving an enumeration problem stops right after having output the last solution. This assumption does not affect the complexity classes studied in this paper, as the output of the last solution can be delayed to the end of the algorithm.

Pseudocode. In this paper, we describe our algorithms using pseudocode that is then compiled to a RAM with the usual complexity guarantees. In our algorithms, we freely use variables and the usual control flow instructions, arithmetic operations and data structures. We also assume that we have access to an $\mathsf{output}(s)$ instruction which outputs the value of variable s. When compiled, this instruction calls the `OUTPUT` instruction of the RAM on the registers holding the value of s.

As this paper mostly deals with transforming a given enumeration algorithm into another one having better complexity guarantees, it is convenient to call an algorithm as an oracle to execute it step by step. Therefore, we use two other instructions in our pseudocode: load and move. The instruction $\mathsf{load}(I, x)$ takes two parameters: the first one is the code of a RAM and the second one is its input. It returns a data structure M that can later be used with the move instruction: $\mathsf{move}(M)$ simulates the next step of the computation of machine I on input x. We assume that $\mathsf{move}(M)$ returns false if the computation is finished. We also assume that we have access to the following functions on M:

- $\mathsf{sol}(M)$ returns the solution that M has just output. If the last simulated step of M was not an output instruction, it returns undef. We abuse notation by writing **if**$(\mathsf{sol}(M))$ **then** ... to express the fact that we explore the **then** branch if and only if $\mathsf{sol}(M)$ is not undef.
- $\mathsf{steps}(M)$ returns the number of steps of M that have been simulated.

If we have an upper bound $u(|x|)$ on the memory used by a machine of code I on input x, and if u is computable in time $t(|x|)$, we can implement load and move on a RAM with respective complexity $O(t(|x|))$ and $O(1)$ and space $O(u(|x|))$. Indeed, it is sufficient to reserve $u(|x|)$ contiguous registers in memory and shift all registers used by I so that it uses the reserved memory.

It is also possible to implement these instructions without knowing in advance the memory used by I but one has to use data structures able to dynamically adjust the memory used. In this case, move can be executed either in $O(1)$ with a small space overhead or in $O(\log(\log(|x|)))$ with no space overhead. We leave this improvement for a long version of this article (see [13]) and state the main results when a polynomial time computable upper bound $u(|x|)$ on the memory is known.

Complexity measures and classes. Complexity measures and the relevant complexity classes for enumeration have been formally introduced by Johnson, Yanakakis and Papadimitriou in [23] first. The *total time*, that is $T_M(x, \sharp A(x))$, is similar to what is used for the complexity of decision problems. Since the number of solutions can be exponential in the *input* size, it is more relevant to give the total time as a function of the combined size of the *input and output*. However, this notion does not capture the dynamic nature of an enumeration algorithm. When generating all solutions already takes too long, we want to be able to generate at least some solutions. Hence, we should measure (and bound) the total time used to produce a given number of solutions. We give here the notion of linear incremental time, central to the paper, while the more general notion of polynomial incremental time is given and studied in Section 3.3.

▶ **Definition 1** (Linear incremental time). *A problem* $\Pi_A \in$ EnumP *is in* IncP_1 *if there is a polynomial* d *and a machine* M *which solves* Π_A*, such that for all* x *and for all* $1 < i \leq \sharp A(x)$*,* $T(x, i) < i \cdot d(|x|)$ *and* $T(x, 1) < d(|x|)$*. Such a machine* M *is called an* IncP_1*-enumerator with incremental delay* $d(n)$*.*

▶ **Definition 2** (Polynomial delay). *A problem* $\Pi_A \in$ EnumP *is in* DelayP *if there is a polynomial* d *and a machine* M *which solves* Π_A*, such that for all* x *and for all* $1 < i \leq \sharp A(x)$*,* $T(x, i) - T(x, i-1) \leq d(|x|)$ *and* $T(x, 1) \leq d(|x|)$*. Such a machine* M *is called a* DelayP*-enumerator of delay* $d(n)$*.*

Observe that if M is a DelayP-enumerator then for all i we have $T(x, i) - T(x, 1) \leq \sum_{1<j\leq i} d(|x|) = (i-1)d(|x|)$. Hence DelayP $\subseteq \mathrm{IncP}_1$. Polynomial delay is the most common notion of tractability in enumeration, because it guarantees both regularity and linear total time and also because it is relatively easy to prove that an algorithm has a polynomial delay. Indeed, most methods used to design enumeration algorithms such as backtrack search with a polynomial time extension problem [29], or efficient traversal of a supergraph of solutions [27, 3, 16], yield polynomial delay algorithms on many enumeration problems.

To better capture the notion of tractability in enumeration, it is important to use polynomial space algorithms. We let $\mathrm{DelayP}^{\mathrm{poly}}$ be the class of problems solvable by a polynomial space DelayP-enumerator. We define $\mathrm{IncP}_1^{\mathrm{poly}}$, as the class of problems which can be solved by a polynomial space IncP_1-enumerator.

3 From IncP_1 to DelayP

3.1 Geometric Amortization

The folklore method (e.g., [23, 34, 14]) used to transform an IncP_1-enumerator into a DelayP-enumerator that was sketched in the introduction uses a queue to delay the output of solutions. This queue may however become of size exponential in the input size. To overcome this issue, we introduce a technique that we call *geometric amortization*, illustrated by Algorithm 1 which regularizes the delay of an IncP_1-enumerator with a space overhead of $O(\log(\sharp I(x)))$, which is polynomially bounded since I is in EnumP. To achieve this, we however have to compromise a bit on the delay which becomes $O((\log(\sharp I(x)) \cdot p(|x|))$. Moreover, with geometric amortization, the solutions are not output in the same order as the order they are output by I. Algorithm 1 relies on the knowledge of an upper bound K of $\sharp I(x)$, but this assumption is relaxed in Section 3.2. We now proceed to prove the correctness and complexity of Algorithm 1 that is summarized in the theorem below.

Algorithm 1 Using geometric amortization for regularizing the delay of an IncP_1-enumerator I having incremental delay $p(n)$ only using polynomial space. In the code, $Z_0 = [0; p(n)]$ and $Z_j = [2^{j-1}p(n) + 1; 2^j p(n)]$ for $j > 0$.

```
Input  : x ∈ Σ* of size n and K such that K ≥ ♯I(x)
Output : Enumerate I(x) with delay O(p(n) · log(K))
1  begin
2      N ← ⌈log(K)⌉;
3      for i = 0 to N do M[i] ← load(I, x) ;
4      j ← N;
5      while j ≥ 0 do
6          for b ← 2p(n) to 0 do
7              move(M[j]);
8              if sol(M[j]) and steps(M[j]) ∈ Z_j then
9                  output(sol(M[j]));
10                 j ← N;
11                 break;
12         if b = 0 then j ← j − 1;
```

▶ **Theorem 3.** *Given an* IncP_1*-enumerator* I *with incremental delay* $p(n)$ *and space complexity* $s(n)$ *and given* $K \geq \sharp I(x)$*, one can construct a* DelayP*-enumerator* I' *which enumerates* $I(x)$ *on any input* $x \in \Sigma^*$ *with delay* $O(\log(K)p(n))$ *and space complexity* $O(s(n)\log(K))$*.*

Proof. The pseudo-code for I', accessing an oracle to I as a blackbox, is presented in Algorithm 1. Its correctness and complexity are proven in the rest of this section. ◀

Since $\mathrm{IncP}_1 \subseteq \mathrm{EnumP}$, we know that there is a polynomial $q(n)$ such that every element of $I(x)$ is of size at most $q(|x|)$ and by choosing $K = |\Sigma|^{q(n)}$, we have that $\log(K)$ is polynomially bounded and the following is a direct corollary of Theorem 3:

▶ **Corollary 4.** $\mathrm{IncP}_1^{\mathrm{poly}} = \mathrm{DelayP}^{\mathrm{poly}}$.

The construction of I' from I in Theorem 3 is presented in Algorithm 1, which uses a technique that we call *geometric amortization*. The idea of geometric amortization is to simulate several copies of I on input x at different speeds. Each process is responsible for enumerating solutions in different intervals of time to avoid repetitions in the enumeration. The name comes from the fact that the size of the intervals we use follows a geometric progression (the size of the $(i+1)^{\text{th}}$ interval is twice the size of the i^{th} one).

Explanation of Algorithm 1. Algorithm 1 maintains $N+1$ simulations $M[0], \ldots, M[N]$ of I on input x where $N = \lceil \log(K) \rceil$. When simulation $M[i]$ finds a solution, it outputs it if and only if the number of steps of $M[i]$ is in Z_i, where $Z_i := [1 + 2^{i-1}p(n), 2^i p(n)]$ for $i > 0$ and $Z_0 = [1, p(n)]$. These intervals are clearly disjoint and cover every possible step of the simulation since the total time of I is at most $\sharp I(x)p(n) \leq Kp(n) \leq 2^N p(n)$ (by convention, we assumed enumerators to stop on their last solution, see Section 2). Thus, every solution is enumerated as long as $M[i]$ has reached the end of Z_i when the algorithm stops.

Algorithm 1 starts by moving $M[N]$. It is given a budget of $2p(n)$ steps. If these $2p(n)$ steps are executed without finding a solution in Z_N, $M[N-1]$ is then moved similarly with a budget of $2p(n)$ steps. It continues until one machine $M[j]$ finds a solution in its zone Z_j. In this case, the solution is output and the algorithm proceeds back with $M[N]$. The algorithm stops when $M[0]$ has left Z_0, that is when $p(n)+1$ steps of $M[0]$ have been simulated[2].

Bounding the delay. From the above description of Algorithm 1, between two outputs, the variable j takes at most $N+1$ values (from N to 0) and at most $2p(n)$ move instructions are executed for each machine $M[j]$. A move instruction can be executed in $O(1)$ (see Appendix A). Moreover, the size of b being $O(\log(n))$, we can increment it in $O(1)$ in the RAM model we consider. Finally, we have to compare $\mathsf{steps}(M[i])$ with integers of values in $O(\log(K)p(n))$. Manipulating such integers in the RAM model would normally cost $O(\log(K)/\log(n))$. However, we give in Appendix A.2 a method using Gray Code encodings which allows us to increment $\mathsf{steps}(M[i])$ and to detect when it enters and exits Z_i in $O(1)$. Thus, the overall delay of Algorithm 1 is $O(\log(K)p(n))$.

Space complexity. We have seen in Section 2 that a RAM can be simulated without using more space than the original machine (see Appendix A for more details). Since Algorithm 1 uses $O(\log(K)$ simulations of I, its space complexity is $O(s(n)\log(K))$.

Correctness of Algorithm 1. It remains to show that Algorithm 1 correctly outputs $I(x)$ on input x. Recall that a solution of $I(x)$ is enumerated by $M[i]$ if it is produced by I at step $c \in Z_i = [1+2^{i-1}p(n), 2^i p(n)]$. Since, by definition, the total time of I on input x is at most $\sharp I(x)p(n)$, it is clear that $Z_0 \uplus \cdots \uplus Z_N \supseteq [1, Kp(n)] \supseteq [1, \sharp(I)p(n)]$ covers every solution and that each solution is produced at most once. Thus, it remains to show that when the algorithm stops, $M[i]$ has moved by at least $2^i p(n)$ steps, that is, it has reached the end of Z_i and output all solutions in this zone.

We study an execution of Algorithm 1 on input x. For the purpose of the proof, we only need to look at the values of $\mathsf{steps}(M[0]), \ldots, \mathsf{steps}(M[N])$ during the execution of the algorithm. We thus say that the algorithm is in state $c = (c_0, \ldots, c_N)$ if $\mathsf{steps}(M[i]) = c_i$ for all $0 \leq i \leq N$. We denote by S_i^c the set of solutions that have been output by $M[0], \ldots, M[i]$ when state c is reached; that is, a solution is in S_i^c if and only if it is produced by I at step $k \in Z_j$ for $j \leq i$ and $k \leq c_j = \mathsf{steps}(M[j])$. We claim the following invariant:

▶ **Lemma 5.** *For every state c and $i < N$, we have $c_{i+1} \geq 2p(n)|S_i^c|$.*

Proof. The proof is by induction on c. For the state c just after initializing the variables, we have that for every $i \leq N$, $|S_i^c| = 0$ and $c_i = 0$. Hence, for $i < N$, $c_{i+1} \geq 0 = 2p(n)|S_i^c|$.

Now assume the statement holds at state c' and let c be the next state. Let $i < N$. If $|S_i^c| = |S_i^{c'}|$, then the inequality still holds since $c_{i+1} \geq c'_{i+1}$ and $c'_{i+1} \geq 2p(n)|S_i^{c'}| = 2p(n)|S_i^c|$ by induction. Otherwise, we have $|S_i^c| = |S_i^{c'}| + 1$, that is, some simulation $M[k]$ with $k \leq i$ has just output a solution. In particular, the variable j has value $k \leq i < N$. Let c'' be the first state before c' such that variable j has value $i+1$ and b has value $2p(n)$, that is, c'' is the state just before Algorithm 1 starts the for loop to move $M[i+1]$ by $2p(n)$ steps. No solution has been output between state c'' and c' since otherwise j would have been

[2] An illustration of Algorithm 1 can be found at `http://florent.capelli.me/coussinet/` where one can see the run of a machine represented as a list and the different simulations moving in this list and discovering solutions.

reset to N. Thus, $|S_i^{c''}| = |S_i^{c'}|$. Moreover, $c_{i+1} \geq c'_{i+1} \geq c''_{i+1} + 2p(n)$ since $M[i+1]$ has moved by $2p(n)$ steps in the for loop without finding a solution. By induction, we have $c''_{i+1} \geq 2p(n)|S_i^{c''}| = 2p(n)|S_i^{c'}|$. Thus $c_{i+1} \geq c''_{i+1} + 2p(n) = 2p(n)(|S_i^{c'}| + 1) = 2p(n)|S_i^{c}|$ which concludes the induction. ◀

▶ **Corollary 6.** *Let $c = (c_0, \ldots, c_N)$ be the state reached when Algorithm 1 stops. We have for every $i \leq N$, $c_i \geq 2^i p(n)$.*

Proof. The proof is by induction on i. If i is 0, then we necessarily have $c_0 \geq p(n)$ since Algorithm 1 stops only when $M[0]$ has moved outside Z_0, that is when it has been moved by at least $p(n)$ steps.

Now assume $c_j \geq 2^j p(n)$ for every $j < i$. This means that for every $j < i$, $M[j]$ has been moved at least to the end of Z_j. Thus, $M[j]$ has found every solution in Z_j. Since it holds for every $j < i$, it means that $M[0], \ldots, M[i-1]$ have found every solution in the interval $K = [1, 2^{i-1}p(n)]$. Since I is an IncP_1-enumerator with delay $p(n)$ and since $2^{i-1} \leq \sharp I(x)$ by definition of N, K contains at least 2^{i-1} solutions, that is, $|S_{i-1}^c| \geq 2^{i-1}$. Applying Lemma 5 gives that $c_i \geq 2^{i-1} \cdot 2p(n) = 2^i p(n)$. ◀

The correctness of Algorithm 1 directly follows from Corollary 6. Indeed, it means that for every $i \leq N$, every solution of $Z_i = [1 + 2^{i-1}p(n), 2^i p(n)]$ have been output, that is, every solution of $[1, 2^N p(n)]$ and $2^N p(n)$ is an upper bound on the total run time of I on input x.

Observe that Algorithm 1 does not preserve the order of the solutions since it interleaves solutions later seen in the enumeration process in order to amortize large delay between two solutions. One possible workaround is to output pairs of the form (s, r) where s is a solution and r its rank in the original enumeration order. To do so, one just has to keep a counter of the number of solutions seen so far by each process and output it along a solution. It does not restore the order, but allows recovering it afterward. When we are interested in finding the first K solutions only (a likely scenario for solving top-k problems), the previous workaround is not useful. However, our algorithm can output them only with a $\log(K)$ overhead by just running $\log(K)$ processes and ignoring solutions having a rank higher than K; the solutions will however not be output in order.

3.2 Improving Algorithm 1

One drawback of Algorithm 1 is that it needs to know in advance an upper bound K on $\sharp I(x)$ since it uses it to determine how many simulations of I it has to maintain. In theory, such an upper bound exists because I is assumed to be in EnumP and it is often known, *e.g.*, $|\Sigma|^N$ where N is an upper bound on the size of the output. In practice, however, it might be cumbersome to compute it or it may hurt efficiency if the upper bound is overestimated. It turns out that one can remove this hypothesis by slightly modifying Algorithm 1. The key observation is that during the execution of the algorithm, if $M[i]$ has not entered Z_i, it is simulated in the same way as $M[i+1], \ldots, M[N]$. Indeed, it is not hard to see that $M[j]$ is always ahead of $M[i]$ for $j > i$ and that if $M[i]$ is not in Z_i, it will not output any solution in the loop at Line 6, hence this iteration of the loop will move $M[i]$ by $2p(n)$ steps, just like $M[j]$ for $j > i$. Hence, Algorithm 1 can be improved in the following way: we start with only two simulations $M[0], M[1]$ of I. Whenever $M[1]$ is about to enter Z_1, we start $M[2]$ as an independent copy of $M[1]$. During the execution of the algorithm, we hence maintain a list M of simulations of I and each time the last simulation $M[N]$ is about to enter Z_N, we copy it into a new simulation $M[N+1]$. The hardest part of implementing this idea is to

Algorithm 2 Improvement of Algorithm 1 which works without upper bounds on the number of solutions and has a better total time. In the code, $a_0 = 0$ and $a_j = 2^{j-1}p(n) + 1$.

```
   Input  : x ∈ Σ* of size n
   Output : Enumerate I(x) with delay O(p(n) · log(♯I(x)))
 1 begin
 2     M ← list(∅);
 3     insert(M, load(I, x));
 4     j ← length(M) − 1;
 5     while j ≥ 0 do
 6         for b ← 2p(n) to 0 do
 7             move(M[j]);
 8             if j = length(M) − 1 and steps(M[j]) = a_j then
 9                 insert(M, copy(M[j]));
10                 j ← length(M) − 1;
11                 break;
12             if sol(M[j]) and steps(M[j]) ∈ [a_j; a_{j+1} − 1] then
13                 output(sol(M[j]));
14                 j ← length(M) − 1;
15                 break;
16         if b = 0 then j ← j − 1;
```

show that one can copy simulation $M[N]$ without affecting the overall delay of the algorithm. That can be achieved by lazily copying parts of $M[N]$ whenever we move $M[N+1]$. The details are given in Appendix A.4.

By implementing this idea, one does not need to know an upper bound on $\sharp I(x)$ anymore: new simulations will be created as long as it is necessary to discover new solutions ahead. The fact that one has found every solution is still witnessed by the fact that $M[0]$ reaches the end of Z_0. This improvement has yet another advantage compared to Algorithm 1: it has roughly the same total time as the original algorithm. Hence, if one is interested in generating every solution with a polynomial delay from an IncP_1-enumerator, our method may make the maximal delay worse but does not change much the time needed to generate all solutions.

Correctness of Algorithm 2. Correctness of Algorithm 2 can be proven in a similar way as for Algorithm 1. Lemma 5 still holds for every state, where N in the statement has to be replaced by $\mathsf{length}(M) - 1$. The proof is exactly the same but we have to verify that when a new simulation is inserted into M, the property still holds. Indeed, let c be a state that follows the insertion of a new simulation (Line 8). We have now $\mathsf{length}(M) - 1 = N + 1$ (thus the last index of M is $N + 1$). Moreover, we claim that $S^c_{N+1} = S^c_N$. Indeed, at this point, the simulation $M[N+1]$ has not output any solution. Moreover, by construction, $c_N = \mathsf{steps}(M[N]) = \mathsf{steps}(M[N+1]) = c_{N+1}$. Since $c_N \geq 2p(n)|S^c_N|$ by induction, we have that $c_{N+1} \geq 2p(n)|S^c_{N+1}|$. Moreover, the following adaptation of Corollary 6 holds for Algorithm 2.

▶ **Lemma 7.** *Let c be the state reached when Algorithm 1 stops. Then $N := \mathsf{length}(M) - 1 = 1 + \log(\sharp I(x))$ and for every $i \leq N$, $c_i \geq 2^i p(n)$.*

Proof. The lower bound $c_i \geq 2^i p(n)$ for $i \leq N$ is proven by induction exactly as in the proof of Lemma 5. The induction holds as long as $2^{i-1} \leq \sharp I(x)$, because we need this assumption to prove that there are at least 2^{i-1} solutions in the interval $[1, 2^{i-1}p(n)]$. Now, one can easily see that if $i \leq 1 + \log(\sharp I(x))$ and $c_i \geq 2^i p(n)$ then the simulation $M[i]$ has reached $2^{i-1}p(n)$ at some point and thus, has created a new simulation $M[i+1]$. Thus, by induction, the algorithms creates at least $1 + \log(\sharp I(x)) = N$ new simulations. Thus $\mathsf{length}(M) \geq N + 1$ (as M starts with one element).

Finally, observe that $M[N]$ outputs solutions in the zone $Z_N = [2^{N-1}p(n) + 1, 2^N p(n)]$ and that $2^{N-1}p(n) = \sharp I(x)p(n)$ which is an upper bound on the total time of I on input x. Thus, the simulation $M[N]$ will end without creating a new simulation. In other words, $\mathsf{length}(M) - 1 = N$. ◀

Delay of Algorithm 2. While establishing the correctness of Algorithm 2 is similar to the one of Algorithm 1, proving a bound on the delay of Algorithm 2 is not as straightforward. By Lemma 7, the size of M remains bounded by $2 + \log(\sharp I(x))$ through the algorithm, so there are at most $2p(n)(2 + \log(\sharp I(x)))$ executions of move between two solutions, for the same reasons as before. However, we also have to account for the execution of copy. When implemented naively, this operation requires a time $O(s(n))$ to copy the entire configuration of the simulation in some fresh part of the memory. It would add $O(s(n))$ to the delay of Algorithm 2 compared to Algorithm 1. However, one can amortize this copy operation by lazily copying the memory while running the original simulation and by adapting the sizes of the zones so that we can still guarantee a delay of $O(\log(\sharp I(x))p(n))$ in Algorithm 2. The method is formally described in Appendix A.4.

Total time of Algorithm 2. A minor modification of Algorithm 2 improves its efficiency in terms of total time. By definition, when simulation $M[i]$ exits Z_j, it does not output solutions anymore. Thus, it can be removed from the list of simulations. It does not change anything concerning the correctness of the algorithm. One just has to be careful to adapt the bounds in Algorithm 2. Indeed, $2^j p(n)$ is not the right bound anymore as removing elements from M may shift the positions of the others. It can be easily circumvented by also maintaining a list Z such that $Z[i]$ always contains the zone that $M[i]$ has to enumerate.

By doing it, it can be seen that each step of I having a position in Z_i will only be visited by two simulations: the one responsible for enumerating Z_i and the one responsible for enumerating Z_{i+1}. Indeed, the other simulations would either be removed before entering Z_i or will be created after the last element of M has entered Z_{i+1}. Thus, the move operation is executed at most $2T(|x|)$ times where $T(|x|)$ is the total time taken by I on input x and the total time of this modification of Algorithm 2 is $O(T(n))$ where $T(n)$ is the total time of I.

All previous comment on Algorithm 2 allows us to state the following improvement of Theorem 3, where no upper bound on $\sharp I(x)$ is necessary but $s(n)$ and $p(n)$ are known.

▶ **Theorem 8.** *Given an* IncP_1*-enumerator* I *with incremental delay* $p(n)$*, space complexity* $s(n)$ *and total time* $T(n)$*, one can construct a* DelayP*-enumerator* I' *which enumerates* $I(x)$ *on any input* $x \in \Sigma^*$ *with space complexity* $O(s(n)\log(\sharp I(x)))$*, delay* $O(\log(\sharp I(x))p(n))$ *and total time* $O(T(n))$*.*

We observe that Algorithm 2 can be modified so that it can work with IncP_1-enumerators having a preprocessing. Indeed one only needs, as a preprocessing step of Algorithm 2, to run the first simulation created by the algorithm until it outputs its first solution to be in the same state as the case where there is no preprocessing.

We sill need to know two parameters (or an upper bound on them) to run Algorithm 2: the space of the amortized algorithm and its incremental delay. By using dynamic data structures, one could adapt our algorithm when the space used by the enumerator is not known for a very small overhead. Moreover, it is possible to give a lower bound showing that one cannot get a $O(p(n))$ polynomial delay when the incremental delay $p(n)$ is unknown (if I is a blackbox). We leave this improvement for a long version of this article (see [13]).

3.3 Geometric Amortization for IncP_i with $i > 1$

The dynamic version of the total time is called *incremental time*: Given an enumeration problem A, we say that a machine M solves Π_A in incremental time $f(i)g(n)$ if on every input x, and for all $i \leq \sharp A(x)$, $T_M(x,i) \leq f(i)g(|x|)$. The linear incremental time corresponds to the case $f(i) = i$. We generalize IncP_1, by polynomially bounding the incremental time.

▶ **Definition 9** (Polynomial incremental time). *A problem $\Pi_A \in \mathrm{EnumP}$ is in IncP_a if there is a constant b and a machine M which solves it with incremental time $O(i^a n^b)$. Such a machine is called an IncP_a-enumerator. Moreover, we define $\mathrm{IncP} = \bigcup_{a \geq 1} \mathrm{IncP}_a$.*

Allowing arbitrary polynomial preprocessing does not modify the class IncP_a since this preprocessing can be interpreted as the polynomial time before outputting the first solution. The class IncP is believed to be strictly included in OutputP, the class of problems solvable in total polynomial time, since this is equivalent to TFNP $\neq$ FP [11]. Moreover, the classes IncP_a form a strict hierarchy assuming the exponential time hypothesis [11].

▶ **Definition 10** (Usual definition of incremental time.). *A problem $\Pi_A \in \mathrm{EnumP}$ is in $\mathrm{UsualIncP}_a$ if there are b and c integers and a machine M which solves Π_A such that for all x and for all $0 < t \leq \sharp A(x)$, $T(x,t) - T(x,t-1) < ct^a|x|^b$.*

Our definition of IncP captures the fact that we can generate t solutions in time polynomial in t and in the size of the input, which seems more general than bounding the delay because the time between two solutions is not necessarily regular. Using geometric amortization, we can show that both definitions are equivalent even when the space is required to be polynomial.

For $a \geq 0$, we denote by $\mathrm{IncP}_a^{\mathrm{poly}}$ (respectively $\mathrm{UsualIncP}_a^{\mathrm{poly}}$), the class of problems that can be solved by an IncP_a (respectively $\mathrm{UsualIncP}_a$) algorithm and polynomial space. The following generalises Corollary 4 since $\mathrm{DelayP} = \mathrm{UsualIncP}_0$.

▶ **Theorem 11.** *For all $a \geq 0$, $\mathrm{IncP}_{a+1}^{\mathrm{poly}} = \mathrm{UsualIncP}_a^{\mathrm{poly}}$.*

Proof. The inclusion $\mathrm{UsualIncP}_a^{\mathrm{poly}} \subseteq \mathrm{IncP}_{a+1}^{\mathrm{poly}}$ is straightforward and follows by a simple computation of the time to generate i solutions, see [11].

The inclusion $\mathrm{IncP}_{a+1}^{\mathrm{poly}} \subseteq \mathrm{UsualIncP}_a^{\mathrm{poly}}$ is done by geometric amortization, by adapting Algorithm 1. Let I be an algorithm solving a problem in IncP_{a+1}. We assume we know $t^{a+1}p(n)$, a bound on its incremental time.

Then, the only modification we do in Algorithm 1 is to maintain a counter S of the number of output solutions and modify the initialization of b in the for loop at line 6 to $S^a(a+1)2p(n)$. By construction of the amortization algorithm, the delay between two solutions before the algorithm ends is bounded by $S^a(a+1)2p(n)\log(s)$, where S is the number of solutions output up to this point of the algorithm and s the total number of solutions. Thus, the algorithm is in $\mathrm{UsualIncP}_a$.

We still have to prove that all solutions are enumerated by the algorithm. Assume that the first $i+1$ machines $M[0], \dots, M[i]$ have output all the solutions in their zones, then we prove as in Corollary 6, that the the machine $M[i+1]$ has also output all its solutions. The number of solutions output by $M[0], \dots, M[i]$ is the number of solutions output by I up to time step $2^i p(n)$. Let s_i be this number, then $s_i^{a+1} p(n) \geq 2^i p(n)$ since I is in incremental time $t^{a+1}p(n)$. Hence, $s_i \geq 2^{i/(a+1)}$.

When a solution is output by a machine $M[j]$ with $j \leq i$, then j is set to N and all machines $M[k]$ with $k > i$ move by at least $S^a(a+1)2p(n)$ steps where S is the current number of output solutions before $M[i]$ moves again. Hence, we can lower bound the number of moves of the machine $M[i+1]$ by $\sum_{S=0}^{s_i} S^a(a+1)2p(n) \geq \sum_{S=0}^{2^{i/(a+1)}} S^a(a+1)2p(n)$. Since $\sum_{S=0}^{n} S^a \geq \int_0^n S^a \, dS \geq n^{a+1}/(a+1)$, the number of moves of $M[i+1]$ is larger than $2^{i+1}p(n)$ which is the upper bound of its zone. ◀

4 Other Applications of Geometric Amortization

4.1 Amortizing Self-Reducible Problems

Given an enumeration problem Π_A, we assume from now on, to lighten the exposition, that the solutions in $A(x)$ are sets over some universe $U(x)$. From A, we define the predicate $\tilde{A}$ which contains the pairs $((x, a, b), y)$ such that $y \in A(x)$ and $a \subseteq y \subseteq b$. From this predicate, we define a self-reducible[3] variant of Π_A and the extension problem ExtSol·A defined as the set of triples (x, a, b) such that there is a y in $\tilde{A}(x, a, b)$.

Solving Π_A on input x is equivalent to solving $\Pi_{\tilde{A}}$ on $(x, \emptyset, U(x))$. Let us now formalize a recursive method to solve $\Pi_{\tilde{A}}$, sometimes called *binary partition*, because it partitions the solutions to enumerate in two disjoint sets. Alternatively, it is called *flashlight search*, because we peek at subproblems to solve them only if they yield solutions. To our knowledge, all uses of flashlight search in the literature can be captured by this formalization, except for the partition of the set of solutions which can be in more than two subsets. We only present the binary partition for the sake of clarity, but our analysis can be adapted to finer partitions.

Given an instance (x, a, b) of $\Pi_{\tilde{A}}$ and some global auxiliary data D, a flashlight search consists in the following (subroutines are not specified, and yield different flashlight searches):

- if $a = b$, a is ouput and the algorithm stops
- choose $u \in b \setminus a$;
- if $(x, a \cup \{u\}, b) \in$ ExtSol·A, compute some auxiliary data D_1 from D and make a recursive call on $(x, a \cup \{u\}, b)$;
- if $(x, a, b \setminus \{u\}) \in$ ExtSol·A, compute some auxiliary data D_2 from D_1 and make a recursive call on $(x, a, b \setminus \{u\})$, then compute D from D_2.

Flashlight search can be seen as a depth-first traversal of a *partial solutions tree*. A node of this tree is a pair (a, b) such that $(x, a, b) \in$ ExtSol·A. Node (a, b) has children $(a \cup \{u\}, b)$ and $(a, b \setminus \{u\})$ if they are nodes. A leaf is a pair (a, a) and the root is $(\emptyset, U(x))$. The *cost* of a node (a, b) is the time to execute the flashlight search on (x, a, b) *except the time spent in recursive calls*. Usually, the cost of a node comes from deciding ExtSol·A and modifying the global data structure D used to solve ExtSol·A faster.

[3] For a classical definition of self-reducible problems, see e.g. [25, 5].

The cost of a path in a partial solution tree is the sum of the costs of the nodes in the path. We define the *path time* of a flashlight search algorithm as the maximum over the cost of all paths from the root. Twice the path time bounds the delay since, between two output solutions, a flashlight search traverses at most two paths in the tree of partial solutions. To our knowledge, all bounds on the delay of flashlight search are proved by bounding the path time. The path time is bounded by $\sharp U(x)$ times the complexity of solving ExtSol·A. Auxiliary data can be used to amortize the cost of evaluating ExtSol·A repeatedly, generally to prove that the path time is equal to the complexity of solving ExtSol·A once, e.g., when generating minimal models of monotone CNF [31].

Using flashlight search, we obtain that $\Pi_A \in$ DelayP if ExtSol·$A \in$ P and indeed many enumeration problems are in DelayP because their extension problem are in P, see e.g., [38, 29]. However, there are NP-hard extension problems whose enumeration problem is in DelayP, e.g., the extension of a maximal clique, whose hardness can be derived from the fact that finding the largest maximal clique in lexicographic order is NP-hard [23].

The *average delay* (also amortized delay or amortized time) of a machine M solving Π_A on input x is $T(x, \sharp A(x))/\sharp A(x)$. The average delay of an enumerator is bounded by its delay but it can be much smaller. This happens in flashlight search when the internal nodes of the tree of partial solutions are guaranteed to have many leaves. Uno describes the pushout method [38] harnessing this property to obtain constant average delay algorithms for many problems such as generating spanning trees.

To make sense of very low complexity enumeration algorithms, we may separate the preprocessing $T(x, 1)$ from the rest of the computation. We say that a machine with preprocessing has incremental delay $d(n)$ if, for all x and i, $T(x, i) - T(x, 1) \leq i \cdot d(|x|)$. The preprocessing is not taken into account in the incremental delay. When the preprocessing time is not zero, it is explicitly specified and we use preprocessing only in this section. We now prove, using Theorem 3, that the average delay of a flashlight search can be turned into a delay up to a small multiplicative factor. It relies on a small queue for amortization, so that its incremental delay is equal to its average delay, and on geometric amortization to turn the incremental delay into a delay.

▶ **Theorem 12.** *Let Π_A be an enumeration problem solved by a flashlight search algorithm, with space $s(n)$, path time $p(n)$ and average delay $d(n)$. Let $b(n)$ be the size of a single solution. There is an algorithm solving Π_A on any input x, with preprocessing $O(p(n)b(n))$, delay $O(d(n)\log(\sharp I(x)))$ and space $O(s(n)\log(\sharp I(x)) + p(n)b(n))$.*

Proof. Let I be the flashlight search algorithm solving Π_A. Let us first describe an algorithm I' in incremental linear time, which produces the same solutions as I on any input x of size n. The preprocessing of I' is to run I for $p(n)$ steps and to store each solution output in a queue. It takes a time at most $O(p(n)b(n))$ since there are at most $p(n)$ solutions of size $b(n)$ to store in the queue. The queue requires an additional space of $O(p(n)b(n))$. After the preprocessing, we first output all solutions in the queue and then I is simulated for the rest of its run and the solutions output by I are output by I' right away.

Checking the queue for emptiness and outputting a solution can be done in constant time. Hence, we can guarantee that there is a constant C, such that after C computation steps of I', one step of I is executed. Let us evaluate the number of solutions output when I' has run for a time Ct after the preprocessing. If at this time the queue is not empty, then a solution has been output at each time step, hence there are at least t output solutions.

If the queue is empty, the number of solutions output by I' is the same as the number of solutions output by I after running for a time $p(n) + t$. At this point in time, the flashlight search is considering some node (a, b) of the partial solutions tree and we denote

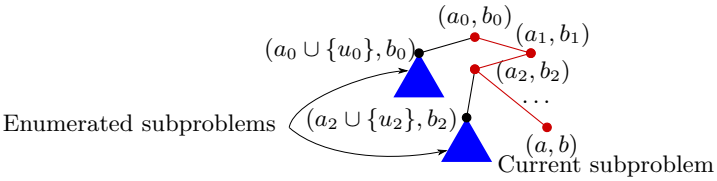


Figure 1 A traversal of the tree of partial solutions by the flashlight search. The subproblems completely solved recursively in blue, the path to the current solution in red.

by $(\emptyset, U(x)) = (a_0, b_0), \ldots, (a_i, b_i) = (a, b)$ the path from the root to (a, b). The time spent on the nodes of this path is bounded by $p(n)$, the path time of I. Hence, I spends at least a time t in the subtrees whose root is a child of some (a_i, b_i).

Also, observe that a subtree rooted at a child (c, d) of (a_i, b_i) with $(c, d) \neq (a_{i+1}, b_{i+1})$ has been either completely explored by the flashlight search or not at all, as shown in Figure 1. Since I is a flashlight search, it works recursively on subtrees, corresponding to subproblems. If a subtree rooted at (c, d) has been completely explored, then the flashlight search has recursively solved the problem $\tilde{A}(x, c, d)$. By definition of the average delay, the solutions in $\tilde{A}(x, c, d)$ have been produced by flashlight search in total time less than $d(n)\sharp\tilde{A}(x, c, d)$. Hence, the subproblems entirely solved by I contribute at least $t/d(n)$ solutions. Therefore, in time Ct, I' outputs at least $t/d(n)$ solutions.

Therefore, we have proven that I' is in incremental delay $O(d(n))$, space $O(s(n)+p(n)b(n))$ and preprocessing $O(p(n)b(n))$. Applying Theorem 3 to I' yields an algorithm with the stated complexity. ◀

4.2 Enumeration of the Models of DNF Formulas

In this section, we explore consequences of Theorem 12 on the problem of generating models of a DNF formula, which has been extensively studied in [12]. Let us denote by n the number of variables of a DNF formula, by m its number of terms and by Π_{DNF} the problem of generating the models of a DNF formula. The size of a DNF formula is at least m and at most $O(mn)$ (depending on the representation and the size of the terms), which can be exponential in n. Hence, we want to understand whether Π_{DNF} can be solved with a delay polynomial in n only, that is depending on the size of a model of the DNF formula but not on the size of the formula itself. A problem that admits an algorithm with a delay polynomial in the size of a single solution is said to be *strongly polynomial* and is in the class SDelayP. One typical obstacle to being in SDelayP is dealing with large non-disjoint unions of solutions. The problem Π_{DNF} is an example of such difficulty: its models are the union of the models of its terms, which are easy to generate with constant delay.

The paper [12] defines the *strong DNF enumeration conjecture* as follows: there is no algorithm solving Π_{DNF} in delay $o(m)$. It also describes an algorithm solving Π_{DNF} in *average* sublinear delay. It is based on flashlight search, with appropriate data structures and choice of variables to branch on (Theorem 10 in [12]). Thanks to Theorem 12, we can trade the average delay for a guaranteed delay and falsify the strong DNF enumeration conjecture.

▶ **Corollary 13.** *There is an algorithm solving* Π_{DNF} *with linear preprocessing, delay* $O(n^2 m^{1-\log_3(2)})$ *and space* $O(n^2 m)$.

Proof. The algorithm of [12] enumerates all models with average delay $O(nm^{1-\log_3(2)})$ and the space used is the representation of the DNF formula by a trie, that is $O(mn)$. We apply Theorem 12 to this algorithm. We have a bound on the incremental delay, the space used and the number of solutions, hence we can use Theorem 3 to do the geometric amortization without overhead in the method of Theorem 12. The auxiliary queue used in Theorem 12 is of size n^2m, since the path time is nm. The number of models is bounded by 2^n, hence the delay obtained by amortization is $O(n^2m^{1-\log_3(2)})$ and the space $O(n^2m)$, which proves the theorem. ◀

For monotone DNF formulas, Theorem 14 of [12] gives a flashlight search with an average delay of $O(\log(mn))$. Hence, we obtain an algorithm with delay $O(n\log(mn))$ listing the models of monotone DNF formulas with strong polynomial delay by Theorem 12. It gives an algorithm having a better delay, preprocessing and space usage than the algorithm given by Theorem 12 of [12].

▶ **Corollary 14.** *There is an algorithm solving Π_{DNF} on monotone formulas with polynomial space, linear preprocessing and strong polynomial delay.*

We have not proven that $\Pi_{DNF} \in$ SDelayP, and the DNF Enumeration Conjecture, which states that $\Pi_{DNF} \notin$ SDelayP still seems credible. Theorem 3 shows that this conjecture can be restated in terms of incremental delay, suggesting that the conjectured hardness should rely on the incremental delay and not on the delay.

▶ **Conjecture 15.** *There is no polynomial p such that Π_{DNF} can be solved with polynomial space and incremental delay $p(n)$.*

References

1 Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

2 Ricardo Andrade, Martin Wannagat, Cecilia C Klein, Vicente Acuña, Alberto Marchetti-Spaccamela, Paulo V Milreu, Leen Stougie, and Marie-France Sagot. Enumeration of minimal stoichiometric precursor sets in metabolic networks. *Algorithms for Molecular Biology*, 11(1):25, 2016.

3 David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.

4 Guillaume Bagan. *Algorithms and complexity of enumeration problems for the evaluation of logical queries*. PhD thesis, Université de Caen, France, 2009.

5 JoséL Balcázar. Self-reducibility. *Journal of Computer and System Sciences*, 41(3):367–388, 1990.

6 Dominique Barth, Olivier David, Franck Quessette, Vincent Reinhard, Yann Strozecki, and Sandrine Vial. Efficient generation of stable planar cages for chemistry. In *International Symposium on Experimental Algorithms*, pages 235–246. Springer, 2015.

7 James R Bitner, Gideon Ehrlich, and Edward M Reingold. Efficient generation of the binary reflected gray code and its applications. *Communications of the ACM*, 19(9):517–521, 1976.

8 Thomas Bläsius, Tobias Friedrich, Julius Lischeid, Kitty Meeks, and Martin Schirneck. Efficiently enumerating hitting sets of hypergraphs arising in data profiling. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 130–143. SIAM, 2019.

9 Kateřina Böhmová, Luca Häfliger, Matúš Mihalák, Tobias Pröger, Gustavo Sacomoto, and Marie-France Sagot. Computing and listing st-paths in public transportation networks. *Theory of Computing Systems*, 62(3):600–621, 2018.

10 Caroline Brosse, Vincent Limouzy, and Arnaud Mary. Polynomial delay algorithm for minimal chordal completions. 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 33:1–33:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.33.

11 Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discrete Applied Mathematics*, 268:179–190, 2019.

12 Florent Capelli and Yann Strozecki. Enumerating models of DNF faster: Breaking the dependency on the formula size. *Discrete Applied Mathematics*, 303:203–215, 2021.

13 Florent Capelli and Yann Strozecki. Geometric amortization of enumeration algorithms. *arXiv preprint arXiv:2108.10208*, 2021.

14 Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 134–148, 2019.

15 Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *Journal of Computer and System Sciences*, 74(7):1147–1159, 2008.

16 Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Listing maximal subgraphs satisfying strongly accessible properties. *SIAM Journal on Discrete Mathematics*, 33(2):587–613, 2019.

17 Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1179–1190, 2019.

18 Stephen A Cook and Robert A Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7(4):354–375, 1973.

19 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4):21, 2007.

20 Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM Journal on Computing*, 32(2):514–537, 2003.

21 Michael Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.

22 Leslie Ann Goldberg. *Efficient algorithms for listing combinatorial structures*. PhD thesis, University of Edinburgh, UK, 1991. URL: http://hdl.handle.net/1842/10917.

23 David S Johnson, Mihalis Yannakakis, and Christos H Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.

24 Leonid Khachiyan, Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On the complexity of some enumeration problems for matroids. *SIAM Journal on Discrete Mathematics*, 19(4):966–984, 2005.

25 Samir Khuller and Vijay V Vazirani. Planar graph coloring is not self-reducible, assuming P≠ NP. *Theoretical Computer Science*, 88(1):183–189, 1991.

26 Donald E Knuth. Combinatorial algorithms, part 1, volume 4a of the art of computer programming, 2011.

27 Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

28 Édouard Lucas. *Récréations mathématiques: Les traversées. Les ponts. Les labyrinthes. Les reines. Le solitaire. La numération. Le baguenaudier. Le taquin*, volume 1. Gauthier-Villars et fils, 1882.

29 Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. *Discrete Mathematics & Theoretical Computer Science*, 21(3), 2019.

30 Kurt Mehlhorn. *Data structures and algorithms 1: Sorting and searching*, volume 1. Springer Science & Business Media, 2013.

31 Keisuke Murakami and Takeaki Uno. Efficient algorithms for dualizing large-scale hypergraphs. *Discrete Applied Mathematics*, 170:83–94, 2014.

32 Ronald C Read and Robert E Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

33 Frank Ruskey. Combinatorial generation. *Preliminary working draft. University of Victoria, Victoria, BC, Canada*, 11:20, 2003.

34 Yann Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université Paris Diderot - Paris 7, 2010.

35 Yann Strozecki. Enumeration complexity. *Bulletin of EATCS*, 1(129), 2019.

36 James C Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Communications of the ACM*, 13(12):722–726, 1970.

37 Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. Any-k algorithms for enumerating ranked answers to conjunctive queries, 2022. doi:10.48550/arXiv.2205.05649.

38 Takeaki Uno. Constant time enumeration by amortization. In *Workshop on Algorithms and Data Structures*, pages 593–605. Springer, 2015.

39 Kunihiro Wasa and Kazuhiro Kurita. Enumeration of enumeration algorithms and its complexity. https://kunihirowasa.github.io/enum/problem_list. Accessed: 2021-10-31.

A Oracles to RAM

In this Appendix, the size of the input of the algorithm is denoted by n. We assume in this section that the polynomial $p(n)$ is the *known* delay of I, the simulated RAM. The complexity of any operation in the RAM model, say $a + b$ is $(\log(a) + \log(b))/\log(n)$. If a and b are bounded by some polynomial in n, then $(\log(a) + \log(b))/\log(n) < C$ for some constant C. All integers used in this section are bounded by a polynomial in n and can thus be manipulated in constant time and stored using constant space. We assume an infinite supply of *zero-initialized memory*, that is all registers of the machines we use are first initialized to zero. It is not a restrictive assumption, since we can relax it, by using a lazy initialization method (see [30] 2, Section III.8.1) for all registers, for only a constant time and space overhead for all memory accesses.

A.1 Pointers and Memory

To implement extensible data structures, we need to use pointers. A pointer is an integer, stored in some register, which denotes the index of the register from which is stored an element. In this article, the value of a pointer is always bounded by a polynomial in n, thus it requires constant memory to be stored. Using pointers, it is easy to implement linked lists, each element contains a pointer to its value and a pointer to the next element of the list. Following a pointer in a list can be done in constant time. Adding an element at the end of a list can be done in constant time if we maintain a pointer to the last element. We also use arrays, which are a set of consecutive registers of known size.

In our algorithms, we may need memory to extend a data structure or to create a new one, but we never need to free the memory. Such a memory allocator is trivial to implement: we maintain a register containing the value F, such that no register of index larger than F is used. When we need k consecutive free registers to extend a data structure, we use the registers from F to $F + k - 1$ and we update F to $F + k$.

A.2 Counters

All algorithms presented in this paper rely, sometimes implicitly, on our ability to efficiently maintain counters, for example, to keep track of the number of steps of a RAM that have been simulated so far. Implementing them naively by simply incrementing a register would result in efficiency loss since these registers may end up containing values as large as $2^{\text{poly}(n)}$ and we could not assume that this register can be incremented, compared, or multiplied in constant time in the uniform cost model that we use in this paper.

To circumvent this difficulty, we introduce in this section a data structure that allows us to work in constant time with counters representing large values. Of course, we will not be able to perform any arithmetic operations on these counters. However, we show that our counter data structure enjoys the following operations in constant time: $\mathsf{inc}(c)$ increases the counter by 1 and $\mathsf{mbit}(c)$ returns the index of the most significant bit of the value encoded by c. In other words, if $k = \mathsf{mbit}(c)$ then we know that $\mathsf{inc}(c)$ has been executed at least 2^k times and at most 2^{k+1} times since the initialization of the counter.

The data structure is based on Gray code encoding of numbers. A Gray code is an encoding enjoying two important properties: the Hamming distance of two consecutive elements in the Gray enumeration order is one and one can produce the next element in the order in constant time. The method we present in this section is inspired by Algorithm G presented in [26] which itself is inspired by [7] for the complexity. The only difference with Algorithm G is that we maintain a stack containing the positions of the 1-bits of the code in increasing order so that we can retrieve the next bit to switch in constant time which is not obvious in Algorithm G. Our approach is closer to the one presented in Algorithm L of [26] but for technical reasons, we could not use it straightforwardly.

We assume in the following that we have a data structure for a stack supporting initialization, push and pop operations in constant time and using $O(s)$ registers in memory where s is the size of the stack (it can be implemented by a linked list).

Counters with a known upper bound on the maximal value. We start by presenting the data structure when an upper bound on the number of bits needed to encode the maximal value to be stored in the counter is known. For now on, we assume that the counter will be incremented at most $2^k - 1$ times, that is, we can encode the maximal value of the counter using k bits.

To initialize the data structure, we simply allocate k consecutive registers $R_0, \dots, R_{k-1}$ initialized to 0, which can be done in constant time since the memory is assumed to be initialized to 0, and we initialize an empty stack S. Moreover, we have two other registers A and M initialized to 0.

We will implement mbit and inc to ensure the following invariants: the bits of the Gray Code encoding the value of the counter are stored in $R_0, \dots, R_{k-1}$. A contains the parity of the number of 1 in $R_0, \dots, R_{k-1}$. M contains an integer smaller than k that is the position of the most significant bit in the Gray Code (the biggest $j \leq k-1$ such that R_j contains 1). Finally, S contains all positions j such that R_j is set to 1 in decreasing order (that is if $j < j'$ are both in S, j will be poped before j').

To implement mbit, we simply return the value of M. It is well-known and can be easily shown that the most significant bit of the Gray Code is the same as the most significant bit of the value it represents in binary so if the invariant is maintained, M indeed contains a value j such that the number of times $\mathsf{inc}(c)$ has been executed is between 2^j and $2^{j+1} - 1$.

To implement inc, we simply follow Algorithm G from [26]. If A is 0 then we swap the value of R_0. Otherwise, we swap the value of R_{j+1} where j is the smallest position such that $R_j = 1$ (if j is $k-1$ then we have reached the maximal value of the code which we

have assumed to be impossible, see below to handle unbounded counters). One can find j in constant time by just popping the first value in S, which works if the invariant is maintained. Now, one has to update the auxiliary memory: A is replaced by $1 - A$ so that it still represents the parity of the number of 1 in the Gray Code. To update S, we proceed as follows: if A is 0 then either R_0 has gone from 0 to 1, in which case we have to push 0 in S or R_0 has gone from 1 to 0, in which case we have to pop one value in S, which will be 0 since S respects the invariant. It can be readily proven that this transformation preserves the invariant on S. Now, if A is 1, then either the value of R_{j+1} has gone from 0 to 1 which means that we have to push $j+1$ and j on the stack (j is still the first bit containing 1 so it has to be pushed back on the top of the stack and $j+1$ is the next bit set to 1 so it has to be just after j in S). Or the value of R_{j+1} has gone from 1 to 0. In this case, it means that after having popped j from S, $j+1$ sits at the top of S. Since R_{j+1} is not 0, we have to pop $j+1$ from S and push back j. Again, it is easy to see that these transformations preserve the invariant on S. Moreover, we never do more than 2 operations on the stack so this can be done in constant time. Finally, if R_{j+1} becomes 1 and $j+1 > M$, we set M to $j+1$.

Observe that we are using $2k+2$ registers for this data structure since the stack will never hold more than k values.

Unbounded counters. To handle unbounded counters, we start by initializing a bounded counter c_0 with k bits (k can be chosen arbitrarily, $k = 1$ works). When c_0 reaches its maximal value, we just initialize a new counter c_1 with $k+1$ bits and modify it so it contains the Gray Code of c_0 (with one extra bit) and copy its stack S and the values of A and M.

This can be done in constant time thanks to the following property of Gray code: the Gray code encoding of $2^k - 1$ contains exactly one bit set to 1 at position $k-1$. Thus, to copy the value of c_0, we only have to swap one bit in c_1 (which has been initialized to 0 in constant time). Moreover, the stack of c_0 containing only positions of bit set to 1, it contains at this point only the value $k-1$ that we can push into the stack of c_1. Copying registers A and M is obviously in constant time.

To summarize, we have proven the following:

▶ **Theorem 16.** *There is a data structure* Counter *that can be initialized in constant time and for which operations* inc *and* mbit *can be implemented in constant time with the following semantic:* mbit(c) *returns an integer j such that v is between 2^j and $2^{j+1} - 1$ where v is the number of time* inc(c) *has been executed since the initialization of c. Moreover, the data structure uses $O(\log(v)^2)$ register.*

One could make the data structure more efficient in memory by lazily freeing the memory used by the previous counters so that it is $O(\log(v))$. However, such an optimization is not necessary for our purpose.

A.3 Instructions load, move and steps for Known Parameters

In this section, we explain formally how one can simulate a given RAM as an oracle with good time and memory guarantees. More precisely, we explain how one can implement the instructions load, move and steps that we are using in our algorithms so that their complexity is $O(1)$ and their memory usage is $O(s(n))$ where $s(n)$ is the memory used by I the simulated RAM on an input of size n. We do the following assumptions: we know an upper bound for both values $s(n)$ and $\lceil \log(\sharp I(x)) \rceil$. We also assume that $s(n)$ is bounded by a polynomial. Note that $\lceil \log(\sharp I(x)) \rceil$ is polynomial in n, since we consider only machines solving problems in EnumP.

Configuration. Instruction load(I, x) returns a structure M which stores the **configuration** of I when its runs on input x. A configuration of I is the content of the registers up to the last one which has been accessed and the state of the machine, i.e. the index of the next instruction to be executed by I. Moreover, the number of executed move(M) instructions is also part of the configuration to support the steps instruction.

Remark that we make explicit that machine I is simulated, by giving it as argument of load. However, the amortization algorithms we design all use load only on the machine I. They must be understood as a method to build an amortized algorithm for each I. Therefore, we do not need a universal machine to simulate I when executing a move(M) instruction.

To simulate I in constant time, the crucial part is to be able to read and write the ith register of I as stored in M in constant time. If we know a bound $s(n)$ on the space used by I, and a bound on the number of solutions $\sharp I(x)$ as in Algorithm 1, the structure M is very simple. For a structure M, we reserve $s(n)$ registers which are mapped one to one to the registers R_1 up to $R_{s(n)}$ of I. We also require 1 register to store the index of the current instruction to be executed by I. We also initialize a counter c to 0 as explained in Section A.2 for steps(M) to keep track of the number of steps that have been simulated so far. This counter will use up to $O(\log(\sharp I(x))^2)$ registers. To really account for steps(M), one should increment c each time an instruction move is executed. However, in Algorithm 1 and Algorithm 2, one need to compare steps(M) with another value. We explain below how one can adapt this counter so that this comparison is constant time for both algorithms.

Let $m = s(n) + 2\lceil\log(\sharp I(x))\rceil + 2$, then for all j from 0 to $\lceil\log(\sharp I(x))\rceil + 1$, the structure $M[j]$ uses the registers from jm to $(j+1)m - 1$. Hence, if $M[j]$ must simulate the access of I to register R_i, it accesses the register R_{jm+i}. This operation is in constant time, since it requires to compute $jm + i$, where i, m and j are polynomial in n.

At Line 8 of Algorithm 1, one has to determine whether the number of steps simulated is in $[2^{j-1}p(n) + 1, 2^j p(n)]$. To check this inequality in constant time, we simply initialize a counter c_j as in Section A.2. Instead of incrementing it each time move$(M[j])$ is called, we increment it every $p(n)$ calls to move. This can easily be done by keeping another register R which is incremented each time move is called and whenever it reaches value $p(n)$, it is reset to 0 and c_j is incremented. Now to decide whether $M[j]$ enters its zone, it is sufficient to test whether mbit$(c_j) = j - 1$. The first time it happens, then exactly $2^{j-1}p(n)$ steps of $M[j]$ have been executed, so it will enter its zone in the next move, so we can remember it to start the enumeration. When mbit(c_j) becomes j, it means that $2^j p(n)$ steps of $M[j]$ have been executed, that is, $M[j]$ leaves its zone. Thus, we can perform the check of Line 8 in constant time.

A.4 Instruction copy

Algorithm 2, which does not require to know $\#I(x)$, relies on an instruction copy. This instruction takes as a parameter a data structure M storing the configuration of a RAM and returns a new data structure M' of the same machine starting in the same configuration (an exact copy of the memory). A straightforward way of implementing copy would be to copy every register used by the data structure M in a fresh part of the memory. However, this approach may be too expensive since we need to copy the whole memory used by M. Since we are guaranteed to have one output solution between each copy instruction, the delay of Algorithm 1 becomes $O(\log(\#I(x))(p(n) + s(n)))$.

In this section, we explain how one can lazily implement this functionality so that the memory of M is copied only when needed. This method ensures that copy runs in $O(1)$, however, there is an overhead to the cost of the instruction move. We show it still runs in $O(1)$ if the memory usage of I is well behaved, otherwise the overhead is small and exists only when $\log(\#I(x)) \leq \log(n)^2$.

Let us explain how the data structure M is lazily copied. The data structure contains a register for the index of the current instruction, a counter of the number of steps and an array to represent the registers of I the simulated machine. The counter in M' is stored as in Theorem 16. It is initialized so that it represents the value $2^{j-1}p(n)$ and it counts up to $2^{j+1}p(n)$. This value is represented by a regular counter of value 0 and the Gray code counter contains the $2^{j-1}p(n)$th integer in Gray code order for integers of size $j+1$. This number is equal to $2^{j-1}+2^{j-2}$, which has only two one bits (the second and the third), hence it can be set up in constant time. The auxiliary structure is the list of ones of the integer, which is here of size two and can thus be set up in constant time.

We explain how we lazily copy an array. Assume we want to create an exact copy of the array A of size m. We create both A' and U of size m initialized to zero. The value of $U[r]$ is 0 if $A'[r]$ has not been copied from $A[r]$ and 1 otherwise. Each time $\mathsf{move}(M)$ is executed and it modifies the value $A[r]$, if $U[r]=0$, it first set $A'[r]=A[r]$ and $U[r]=1$. Each time $\mathsf{move}(M')$ is executed and reads the value $A'[r]$, if $U[r]=0$, it first set $A'[r]=A[r]$ and $U[r]=1$. This guarantees that the value of A' is always the same as if we had completely copied it from A when the instruction $\mathsf{copy}(M)$ is executed. The additional checks and updates of U add a constant time overhead to move. Moreover, we maintain a simple counter c, and each time a $\mathsf{move}(M')$ operation is executed, if $U[c]=0$, we set $A'[c]=A[c]$ and $U[c]=1$. When $c=m$, the copy is finished and we can use A and A' as before, without checking U.

The described implementation of the copy operation is in constant time. The move instruction, modified as described, has a constant overhead for each lazy copy mechanism in action. To evaluate the complexity of Algorithm 2, we must evaluate the number of active copies. We prove, that when $s(n)$ is known, a variant of Algorithm 2 has only a single active copy mechanism at any point in time.

▶ **Theorem 17.** *Given an* IncP_1*-enumerator* I*, its incremental delay* $p(n)$ *and its space complexity* $s(n)$*, one can construct a* DelayP*-enumerator* I' *which enumerates* $I(x)$ *on input* $x \in \Sigma^*$ *with delay*

$$O(\log(\#I(x))p(n))$$

and space complexity $O(s(n)\log(\#I(x)))$.

Proof. We use a hybrid version of Algorithm 1 and Algorithm 2. First, in the preprocessing step, I is run for $s(n)$ steps. If the computation terminates before $s(n)$ steps, then we store all solutions during the preprocessing and enumerate them afterward.

Otherwise, let i be the integer such that $2^{i-1}p(n) \leq s(n) < 2^i p(n)$. We run Algorithm 1 with $\log(s(n)/p(n))$ as a bound on the number of solutions. It means that $M[0]$ up to $M[i]$ are loaded in the preprocessing. Since the number of solutions can be larger than $\log(s(n)/p(n))$, we need machines $M[j]$ for $j > i$. These machines are created dynamically as in Algorithm 2. When a machine $M[j]$ is created, it is lazily copied from $M[j-1]$ using copy. There are at least $2^{j-1}p(n) \geq 2^i p(n) \geq s(n)$ instructions executed before the next copy instruction. Therefore, a single lazy copy is active at any point of the algorithm, which proves that the delay is $O(\log(\#I(x))p(n))$. ◀

One Drop of Non-Determinism in a Random Deterministic Automaton

Arnaud Carayol
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Philippe Duchon
Univ. Bordeaux, CNRS UMR 5800, LaBRI, F-33400 Talence, France

Florent Koechlin
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Cyril Nicaud
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Abstract

Every language recognized by a non-deterministic finite automaton can be recognized by a deterministic automaton, at the cost of a potential increase of the number of states, which in the worst case can go from n states to 2^n states. In this article, we investigate this classical result in a probabilistic setting where we take a deterministic automaton with n states uniformly at random and add just one random transition. These automata are almost deterministic in the sense that only one state has a non-deterministic choice when reading an input letter. In our model each state has a fixed probability to be final. We prove that for any $d \geq 1$, with non-negligible probability the minimal (deterministic) automaton of the language recognized by such an automaton has more than n^d states; as a byproduct, the expected size of its minimal automaton grows faster than any polynomial. Our result also holds when each state is final with some probability that depends on n, as long as it is not too close to 0 and 1, at distance at least $\Omega(\frac{1}{\sqrt{n}})$ to be precise, therefore allowing models with a sublinear number of final states in expectation.

2012 ACM Subject Classification Theory of computation → Regular languages; Mathematics of computing → Combinatorics; Mathematics of computing → Probability and statistics

Keywords and phrases non-deterministic automaton, powerset construction, probabilistic analysis

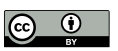
Digital Object Identifier 10.4230/LIPIcs.STACS.2023.19

Acknowledgements The authors would like to thank the reviewers for their helpful comments.

1 Introduction

A fundamental result in automata theory is that deterministic complete finite state automata recognize the same languages as non-deterministic finite state automata. This result can be established using the classical (accessible) subset construction [17, 14]: starting with a non-deterministic automaton with n states, one can build a deterministic automaton with at most 2^n states that recognizes the same language. This upper bound is tight; there are regular languages recognized by an n-state non-deterministic automaton whose minimal automaton (the smallest deterministic and complete automaton that recognizes the language) has 2^n states. The number of states of the minimal automaton of a regular language is called its *state complexity*. Figure 1 shows two n-state non-deterministic automata with somewhat similar shape, and whose languages have very different state complexities. In both automata, there is only one non-deterministic choice, when reading the letter a at the initial state.

In this article, we address the following (informal) question: if we take a random n-state deterministic automaton and add just one random transition, what can be said about the state complexity of the resulting recognized language? Does it hugely increase as for $\mathcal{L}_\ell$, or does it remain small as for $\mathcal{L}_r$?


© Arnaud Carayol, Philippe Duchon, Florent Koechlin, and Cyril Nicaud;
licensed under Creative Commons License CC-BY 4.0
40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 19; pp. 19:1–19:14

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

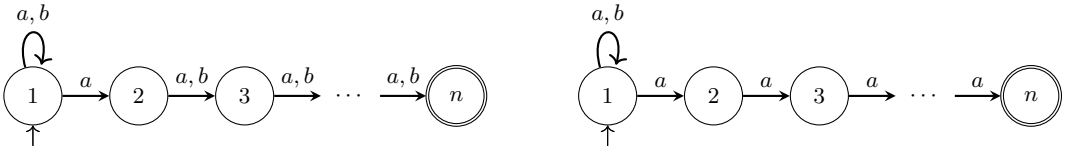


Figure 1 On the left, a non-deterministic automaton with n states recognizing the language $\mathcal{L}_\ell = \Sigma^* a \Sigma^{n-2}$. On the right, a non-deterministic automaton with n states recognizing the language $\mathcal{L}_r = \Sigma^* a^{n-1}$. The minimal automaton of $\mathcal{L}_\ell$ has 2^{n-1} states, whereas the one of $\mathcal{L}_r$ has n states.

From [3], we know that with high probability, the state complexity of the language recognized by a size-n deterministic automaton taken uniformly at random is linear. This is important as it implies that the corresponding distribution on regular languages is not degenerated: this contrasts with the case of random regular expressions where the expected state complexity of the described regular languages is constant [15] which means that the induced distribution on regular languages is concentrated on a finite number of languages.

To be more precise, our formal setting in this article is the following. Let $\Sigma = \{a, b, \ldots\}$ be a finite alphabet with $k \geq 2$ letters. For any $n \geq 1$, we consider the uniform distribution on deterministic and complete automata on Σ, with $\{1, \ldots, n\}$ as their set of states and with no final states (for now); the initial state is picked uniformly at random, and the action of the letters on the stateset are k uniform and independent random mappings. We also pick uniformly at random and independently two states p and q, and add a transition $p \xrightarrow{a} q$, if it is not already there. Finally each state is final with a given fixed probability $f \in (0, 1)$, independently. Hence in this model an almost deterministic automaton has an expected number of final states of $f\,n$. Our results still hold if we allow the probability f of being final to depend on the size n of the automaton, provided that f_n has a distance to 0 and 1 in $\Omega(\frac{1}{\sqrt{n}})$. This allows us to consider a probabilistic model in which random automata have an expected number of final states that is as low as $\Theta(\sqrt{n})$.

Our main result is that for any $d \geq 1$ there exists a constant $c_d > 0$ such that the state complexity of the language of such a random almost deterministic automaton is greater than n^d with probability at least c_d, for n sufficiently large. That is, for any polynomial P, there is a non-negligible probability that the state complexity of the language of a random automaton is greater than $P(n)$: we will say that the state complexity is *super-polynomial* with *visible probability*. As a direct consequence, the expected state complexity is super-polynomial.

It should be noted that with the same random models for deterministic automata, one cannot hope to replace visible probability in our results with a probability that converges to 1 (high probability). Indeed random automata have, with high probability, a constant fraction of states that are not accessible from the initial state; if the source of the added transition is not accessible from the initial state, the added transition does not impact the recognized language, whose state complexity is therefore at most equal to n. Thus, we make no effort in the present paper to optimize our probabilistic lower bounds. See the conclusion for a more advanced discussion on this topic.

Related work. The study of random deterministic automata can be traced back to the work of Grusho on the size of the accessible part [13]: he established that, with high probability, a constant proportion of the states are accessible from the initial state. He also shows that with high probability there is a unique terminal strongly connected component of size approximately $\nu_k n$, for some $\nu_k > \frac{1}{2}$ that only depends on the size k of the alphabet.

More structural results on the underlying graph of a random deterministic automaton were established in the work of Carayol and Nicaud [6], with a local limit law for the size of the accessible part and an application to random generation of accessible deterministic automata, and more recently in the work of Cai and Devroye [5], with, in particular, a fine grained analysis of what is happening outside the large strongly connected component. In [1], Addario-Berry, Balle and Perarnau gave a precise analysis of the diameter of a random deterministic automaton, showing in particular that it is logarithmic. We will use some of these results in this paper, namely one on the size of the largest terminal strongly connected component. We will deal with the restriction to states accessible from the initial state in the powerset construction using the result of [5] that with high probability the cycles outside the accessible part are small: for any $\varepsilon > 0$, with probability at least $1 - \varepsilon$ all the non-accessible cycles have length smaller than some constant C_ε. In particular, for any $\omega(n) \to \infty$, all the cycles outside the accessible part have length at most $\omega(n)$ with high probability.

All these results on random automata focus on the underlying graph of the transition structures, without saying much about the recognized languages, and on the average complexity of textbook algorithms on automata. Some results were established in this direction: the probability that a random accessible automaton is minimal was studied by Bassino, David and Sportiello [3], the analysis of minimization algorithms by Bassino, David and Nicaud [2, 8], etc. More recently, several papers studied the synchronization of random automata [4, 19], until the very recent work of Chapuy and Perarnau [7], establishing that most deterministic automata are synchronizing, with a word of length $O(\sqrt{n}\log n)$. We refer the interested reader to the survey of Nicaud [18] for an overview on random deterministic automata.

To our knowledge, there is no well-established random model for non-deterministic automata (e.g., for the uniform distribution they recognize almost all words with high probability). Applying the powerset construction to the mirror of a random deterministic automaton was studied by De Felice and Nicaud [10, 11], in order to analyze the average case complexity of Brzozowski's state minimization algorithm. As in the present article, they studied the determinization procedure of random automata, but for a model that is very different from ours: they consider the mirror of a uniform random deterministic automaton. In particular, with high probability, there is a linear number of states having a non-deterministic choice in their setting. Another natural model would be to use a critical Erdős-Rényi [9] digraph for each letter, which would also result in a linear number of states having a non-deterministic choice. In this article, we choose a random model with the minimum amount of non-determinism by adding just one transition to a uniform deterministic automaton, and establish that we likely have a combinatorial explosion already in this case.

2 Definitions and notations

For any $n \geq 1$, let $[n] = \{1, \ldots, n\}$. If $x, y \in \mathbb{R}$ with $x \leq y$, let $[\![x, y]\!] = [x, y] \cap \mathbb{Z}$ be the set of integers that are between x and y. Let $\mathcal{E}$ be a set equipped with a size function s from $\mathcal{E}$ to $\mathbb{Z}_{\geq 0}$, and let $\mathcal{E}_n$ denote the elements of $\mathcal{E}$ having size n. A property X on $\mathcal{E}$ (that is, a subset of $\mathcal{E}$ viewed as the set of elements for which the property holds) holds with *visible probability* if there exists some constant $c > 0$ such that, for n sufficiently large, $\mathcal{E}_n$ is non-empty and $\mathbb{P}(X) \geq c$ for the uniform distribution on $\mathcal{E}_n$. By a slight abuse of notation, if X is a random variable $\mathcal{E} \to \mathbb{Z}_{\geq 0}$ we say that for the uniform distribution on $\mathcal{E}$, X is *super-polynomial with visible probability* when for any $d \geq 1$, there exists a constant $c_d > 0$, such that for n sufficiently large, $\mathcal{E}_n \neq \emptyset$ and $\mathbb{P}(X \geq n^d) \geq c_d$ for the uniform distribution on $\mathcal{E}_n$.

Recall that if u and v are two words on an ordered alphabet Σ, u is *smaller than v for the length-lexicographic order* if $|u| < |v|$ or they have same length and $u <_{\text{lex}} v$ for the lexicographic order.

Throughout the article, the stateset of an automaton with n states will always be $[n]$, with the exception of the powerset construction recalled just below. The alphabet will always be $\Sigma = \{a, b\}$, except in the statement of our main theorem, where we allow larger alphabets as it is trivially generalized to this case. Hence, in our setting, a *deterministic (and complete) automaton* is just a tuple (n, δ, F), where $F \subseteq [n]$ is the *set of final states* and δ is the *transition function*, a mapping from $[n] \times \Sigma$ to $[n]$. We will often write $\delta_\alpha(s) = t$ or $s \xrightarrow{\alpha} t$ instead of $\delta(s, \alpha) = t$, for $s, t \in [n]$ and $\alpha \in \Sigma$, and call this an α-transition or a transition. The transition function is classically extended to sets of states by setting $\delta(X, \alpha) = \{\delta(s, \alpha) : s \in X\}$, for $X \subseteq [n]$, and to words by setting inductively $\delta(s, w) = s$ if w is the empty word ε and $\delta(s, w\alpha) = \delta(\delta(s, w), \alpha)$. We will not need to specify the *initial state* until the end of the proof; when we finally do, it will be generated uniformly at random and independently in $[n]$. Final states are only used in the last part of our proof, so to ease the presentation, we define a *deterministic (and complete) transition structure* as being an automaton with neither initial nor final states: they are given by a pair (n, δ) where n is the number of states and δ is the transition function.

An *almost deterministic automaton* $(n, \delta, F, p \xrightarrow{a} q)$ is a deterministic automaton (n, δ, F) in which we add the additional a-transition $p \xrightarrow{a} q$. Similarly, an *almost deterministic transition structure* $(n, \delta, p \xrightarrow{a} q)$ is a deterministic transition structure (n, δ) in which we add the additional a-transition $p \xrightarrow{a} q$. For any $\alpha \in \Sigma$ and any $r \in [n]$, the transition function γ of an almost deterministic automaton $(n, \delta, F, s \xrightarrow{a} t)$ (or almost deterministic transition structure) is therefore defined by $\gamma(r, \alpha) = \{\delta(r, \alpha)\}$ if $(r, \alpha) \neq (p, a)$ and $\gamma(p, a) = \{\delta(p, a), q\}$. These automata or transition structures can be deterministic, when we already have $\delta(p, a) = q$.

The classical *powerset automaton* $\mathcal{B}$ of a possibly non-deterministic automaton $\mathcal{A} = (n, \delta, F, p \xrightarrow{a} q)$, with a transition function γ, is a deterministic automaton $\mathcal{B}$ with states in $2^{[n]}$ and transition function γ extended to sets, as defined above. If we add an initial state i_0 to $\mathcal{A}$, the initial state of $\mathcal{B}$ is $\{i_0\}$ and it recognizes the same language as $\mathcal{A}$ when a state X of $\mathcal{B}$ is final if and only if at least one of its element is final in $\mathcal{A}$, i.e. $X \cap F \neq \emptyset$. We can restrict this construction to the accessible part of $\mathcal{B}$ only (from its initial state $\{i_0\}$, where i_0 is the initial state of $\mathcal{A}$) while still recognizing the same language; we call this automaton the *accessible powerset automaton* of $\mathcal{A}$.

Recall that two states r and s in a deterministic automaton $\mathcal{A}$ are *equivalent* if the languages recognized by moving the initial state to r or to s are equal. The *minimal automaton* of a regular language $\mathcal{L}$ is the deterministic complete automaton with the smallest number of states that recognizes $\mathcal{L}$. The number of states of the minimal automaton of $\mathcal{L}$ is called the *state complexity* of $\mathcal{L}$. We will use the following classical property [14]:

▶ **Proposition 1.** *If there is a set of accessible states X in a deterministic automaton $\mathcal{A}$ such that the states of X are pairwise non-equivalent, then $\mathcal{A}$ has state complexity at least $|X|$.*

The following remark allows us to focus on the case of a two-letter alphabet:

▶ Remark 2. Let $\Gamma \subseteq \Sigma$ be two non-empty alphabets. If $\mathcal{L}$ is a regular language on Σ, the state complexity of $\mathcal{L}$ is at least the state complexity of $\mathcal{L} \cap \Gamma^*$.

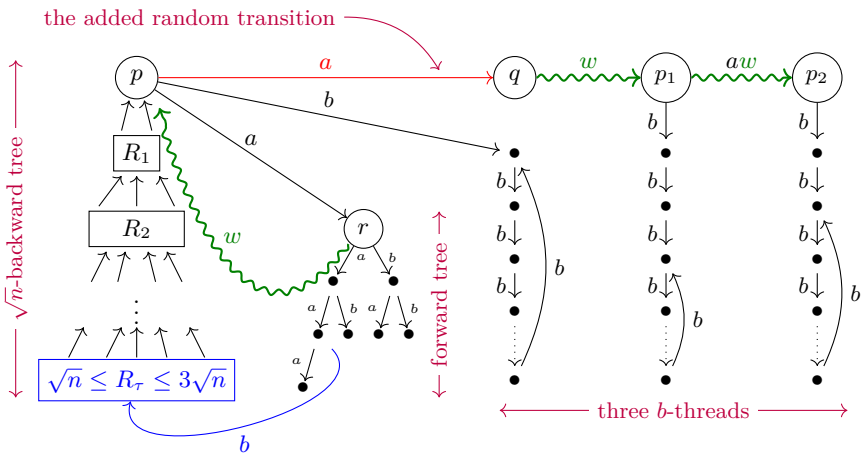


Figure 2 Illustration of the proof sketch. On the left, the backward tree from p that is detailed in Section 4.1, it has size $O(\sqrt{n})$ and contains between $\sqrt{n}$ and $3\sqrt{n}$ extremal leaves (i.e. leaves in its last level τ) to be valid. On its right, the forward tree from r, described in Section 4.2; it is a breadth-first traversal that is valid if it hits an extremal leaf of the backward tree before $O(\sqrt{n})$ states are examined. On the right the b-threads introduced in Section 4.3, obtained by reading b's from the p_i's; they are valid if they are made of previously unseen states and do not intersect.

3 Main statement and proof outline

Our main result is that the state complexity of the language recognized by a random almost deterministic automaton is super-polynomial with visible probability, when for each n, each state is final, independently, with some probability f_n that is not too close to either 0 or 1, as precised in the statement:

▶ **Theorem 3.** *Let Σ be an alphabet with at least two letters. Let f_n be a map from $\mathbb{Z}_{\geq 1}$ to $(0,1)$ such that there exists a constant $\alpha > 0$ such that $f_n \geq \frac{\alpha}{\sqrt{n}}$ and $1 - f_n \geq \frac{\alpha}{\sqrt{n}}$ for n sufficiently large. Consider an almost deterministic n-state transition structure $\mathcal{A}$ on Σ taken uniformly at random. Each state of $\mathcal{A}$ is then taken to be final with probability f_n, independently of everything else. Then, with visible probability, the language recognized by $\mathcal{A}$ has super-polynomial state complexity.*

▶ **Corollary 4.** *Under the conditions of Theorem 3, the expected state complexity of the language recognized by $\mathcal{A}$ growths faster than any polynomial in n.*

The proof of Theorem 3 consists in identifying a structure and several constraints (see Figure 2) that guarantee that when performing the accessible powerset construction and adding a random set of final states, we have sufficiently many pairwise non-equivalent states. At each step, we add a new constraint on top of those we already have, and we have to ensure that these constraints are still satisfied by sufficiently many almost deterministic transition structures. A convenient way to sketch the proof is to consider that we start with n states and no transitions, and add random transitions when needed, on the fly. More precisely, our proofs can be seen as the description of an algorithm that tries to expose the required structure by performing two types of queries on the set of still unknown transitions: either we ask what the destination of a given transition is, or we ask for all the transitions that have a given state as their destination. Thus, at any point in the algorithm, conditioned on the results of all previous queries, the destinations of all still unexposed transitions are

independent and uniform among the set of states for which we have not performed the second type of query. We use this to prove that our algorithm has a non-negligible probability of success. We also have two random states p and q and will add the transition $p \xrightarrow{a} q$ at some point. We fix $d \geq 1$ and describe the main steps of the proof below. In this high level description, recall that δ refers to the transition function of the deterministic base of the almost deterministic automaton being generated, whereas the γ refers to its transition function when adding the transition $p \xrightarrow{a} q$ during step 2. Except during this step, all transitions are added to both δ and γ simultaneously.

1. Generate $r = \delta_a(p)$, the target of the a-transition starting from p in the deterministic transition structure. With visible probability, $r \neq q$ and there is a word w of length $\Theta(\log n)$ such that $\delta_w(r) = p$, which can be found by generating $O(\sqrt{n})$ random transitions. We also assume that the b-transition starting at p is still unset. This step is the most technical, we explore backward from p and forward from r until we reach a common state.
2. Assuming such a w is found, we add the transition $p \xrightarrow{a} q$ to γ, which makes the automaton non-deterministic. We then iteratively generate the transitions starting from q and following the word $w(aw)^{d-1}$, and ask that the target of each such transition be a state that was not previously seen in the whole process. This happens with visible probability.
3. Let $p_0 = p$ and $p_i = \delta_{w(aw)^{i-1}}(q)$ for $i \in [d]$. If the two previous steps are successful, then $\gamma_{(aw)^d}(\{p\}) = \{p_0, p_1, \dots, p_d\}$, and the outgoing b-transition of each p_i is still unset. Then, for each p_i, we iteratively generate the b-transitions $\delta_b(p_i)$, $\delta_{bb}(p_i)$, ... until we cycle after λ_i steps. This process is considered successful if we do not use an already set b-transition and if the $d+1$ cycles are pairwise disjoint. We furthermore ask that the λ_i are all in $\Theta(\sqrt{n})$. All these properties happen with visible probability.
4. At this stage, we have $\gamma_{(aw)^d}(\{p\}) = \{p_0, \dots, p_d\}$; this set is composed of $d+1$ different states, and reading b's from each p_i eventually ends in a b-cycle of length ℓ_i. Given the λ_i's, each ℓ_i is a uniform element of $[\lambda_i]$, and they are independent. We now ask that the ℓ_i's are pairwise coprime, and that each of them is in $\Omega(\sqrt{n})$. This also happens with visible probability [20]. Our precise requirements ensure that once met, the ℓ_i are uniform and independent elements of $[\![\frac{1}{2}\sqrt{n}, \sqrt{n}]\!]$.
5. If everything worked so far, in the powerset construction applied to the almost deterministic transition structure there is a b-cycle of length $\prod_{i=0}^{d} \ell_i = \Omega(n^{\frac{d+1}{2}})$. We now randomly determine which states are final. If we consider a b-cycle alone in the automaton, of length $\Omega(\sqrt{n})$, its states are pairwise non-equivalent with visible probability as soon as the probability f_n that a state is final is not too close to either 0 or 1, which we assumed in our model. This property happens to be preserved when building the product automaton for the union of two one-letter cycles, provided their lengths are coprime. Consequently, the large b-cycles in the powerset construction are made of pairwise non-equivalent states with visible probability.
6. It just remains to guarantee that $\{p\}$ is accessible in the subset construction. We use the fact that with high probability, all cycles with length in $\Omega(\log(n))$ are accessible in a random deterministic automaton [5]. By construction the cycle around p labelled aw built at step 1 has length $\Theta(\log n)$, hence p is accessible with high probability.

The first steps of the proof sketch are depicted in Figure 2, with more details and notations that will be introduced in the next section.

4 Random almost deterministic transition structures

As indicated in the presentation of the proof in Section 3, a convenient way to see a uniform random transition structure is to start with no known transition at all, and generate them on the fly, when needed: we use the fact that the targets of the $2n$ transitions in a size-n uniform transition structure are independent uniform random elements of $[n]$.

Consider for instance that we take a random state s and iteratively follow b-transitions starting from s: we generate the path $s \xrightarrow{b} \delta(s,b) \xrightarrow{b} \delta(s,bb) \xrightarrow{b} \dots$ until we cycle back on a previously seen state. In this process, we keep picking uniformly at random and independently integers in $[n]$ until we have a collision: this is exactly the setting of the classical Birthday Problem (see for instance [12, p.114]). Straightforward computations show that the expected length ℓ_s of this b-path $\mathcal{P}_s$ is in $\Theta(\sqrt{n})$, and that it is between $\sqrt{n}$ and $2\sqrt{n}$ with visible probability.

Now suppose that we want to add the condition that the target of every a-transition outgoing from a state of $\mathcal{P}_s$ is not in $\mathcal{P}_s$. We can proceed as follows: for a given fixed path $\mathcal{P}_s$ of length ℓ_s, the Birthday Problem analysis tells us that with visible probability the outgoing a-transitions do not reach $\mathcal{P}_s$. As long as $\sqrt{n} \leq \ell_s \leq 2\sqrt{n}$, we can lower bound this probability by a constant that does not depend on ℓ_s. Moreover, a given transition structure can have only one b-path from s, so we can partition the set of size-n transition structures according to their b-path, for a given s. Hence a simple computation using the law of total probabilities (or direct counting) shows that we can combine the two "with visible probability" and that, with visible probability there is a b-path $\mathcal{P}_s$ from s of length between $\sqrt{n}$ and $2\sqrt{n}$ such that every outgoing a-transition ends outside $\mathcal{P}_s$.

We detailed this reasoning because it is the main technique we will use in the sequel to build on the previous results and add new constraints, until we exhibit a shape that ensures that applying the accessible powerset construction will produce a large (super-polynomial) number of states. Also, we will rely much on properties derived from the Birthday Problem, such as:

- If we generate $O(\sqrt{n})$ elements of $[n]$, there is no collision with visible probability, even if there is a set of forbidden states of size $O(\sqrt{n})$ which make the process fail.
- If we generate $\Omega(\sqrt{n})$ elements of $[n]$, there is a collision with visible probability, even if there is a set of forbidden states of size $O(\sqrt{n})$ which make the process fail.
- If we generate random elements of $[n]$, with visible probability we hit a fixed set of states of size $\Omega(\sqrt{n})$ before a collision occurs.

4.1 Backward tree

We first look at the shape of a typical backward tree[1] from a state p in a random transition structure $\mathcal{T} = (n, \delta)$. We define $d(x,y)$ as the smallest length of a word w such that $\delta_w(x) = y$ (and ∞ if y is not accessible from x). For a given state p, we consider the backward exploration of $\mathcal{T}$ starting from p: we iteratively build the sets of states $R_i(p) = \{x : d(x,p) = i\}$. For $\tau \geq 1$, the nodes of the *backward tree* of depth τ from p are $B_\tau(p) = \cup_{i=0}^{\tau} R_i(p)$ and the edges are the transitions $x \xrightarrow{\alpha} y$ that go from a state $x \in R_i(p)$ to a state $y \in R_{i-1}(p)$, for $i \in [\tau]$.

We keep building the backward tree until the first time τ where $|R_\tau(p)| \geq \sqrt{n}$. If such a τ exists, the tree is called the $\sqrt{n}$-backward tree. If the transition structure is taken uniformly at random, there is a visible probability that $R_\tau(p)$ exists and has size at most $3\sqrt{n}$, that $\tau = \Omega(\log n)$ and that the whole $\sqrt{n}$-backward tree contains at most $O(\sqrt{n})$ nodes.

[1] The backward tree is not a tree in the graph theoretical sense as a node at depth ℓ can have two out-going edges to two different nodes at depth $\ell - 1$.

To see that, first consider $R_1(p)$. Each state $x \neq p$ can be in $R_1(p)$, if there is a transition $x \xrightarrow{a} p$ or $x \xrightarrow{b} p$ (or both) in $\mathcal{T}$. This happens with probability $\pi_n^{(1)} = \frac{2}{n} - \frac{1}{n^2} \approx \frac{2}{n}$. The cardinality of $R_1(p)$ thus follows a binomial law of parameters $n-1$ and $\pi_n^{(1)}$. In particular, in expectation it contains around 2 states.

Assume now that we know all the $R_j(p)$ for $j \leq i$ and want to compute $R_{i+1}(p)$; we suppose that $R_i(p) \neq \emptyset$. Recall that $B_i(p) = \cup_{j=0}^{i} R_j(p)$ and let $k_i = |B_i(p)|$. By definition of d, none of the states of $B_i(p)$ can be in $R_{i+1}(p)$. On the other hand, any state x of $[n] \setminus B_i(p)$ can be in $R_{i+1}(p)$, and the condition that a state is not in $B_i(p)$ is exactly that its outgoing transitions are not in $B_{i-1}(p)$. All other target states are equally likely under this conditioning, for both transitions. Hence there are $n - k_{i-1}$ possible targets for $\delta(x,a)$ and $\delta(x,b)$: the probability that at least one of them is in $R_i(p)$ is $\pi_n^{(i)} = \frac{2|R_i(p)|}{(n-k_{i-1})} - \frac{|R_i(p)|^2}{(n-k_{i-1})^2} \approx \frac{2|R_i(p)|}{n}$ if $|R_i(p)|$ and k_{i-1} are both $o(n)$. Hence the number of elements in $R_{i+1}(p)$ follows a binomial law of parameters $n - k_i$ and $\pi_n^{(i)}$. In particular, in expectation, $R_{i+1}(p)$ is roughly twice as large as $R_i(p)$, as long as they are not too big. Since binomial laws are concentrated around their means, the presentation above can be turned into a formal proof, establishing the following result.

▶ **Lemma 5.** *Let p be a random state of a random n-state deterministic transition structure. With visible probability, the $\sqrt{n}$-backward tree from p exists, has depth $\tau \in \Theta(\log n)$, contains between $\sqrt{n}$ and $3\sqrt{n}$ extremal leaves, i.e. states in $R_\tau(p)$, and has a total number of nodes in $\Theta(\sqrt{n})$.*

In [5], Cai and Devroye also consider backward trees, with a precise analysis for fixed depth (that does not depend on n) conditionally on p being in the large strongly connected component; they use approximation by a Galton-Watson branching process. This allows them to give a more precise analysis on the existence of the circuit we are building in this paper: they prove that conditioned on the fact that p is accessible, there is such a circuit with high probability. However we cannot reuse their result directly, since we need to quantify the amount of randomness used to discover the circuit: we need unset transitions to continue our construction. It is not obvious to describe the distribution of the transitions if we condition on the existence of the circuit (in particular, there can be several such circuits).

In our setting, we have a direct access to the distribution of most unseen transitions. Indeed, if we fix the $\sqrt{n}$-backward tree T_p from p and consider a state x that is not in the tree, its outgoing transitions can end either in $[n] \setminus T_p$ or at an *extremal leaf*, a leaf of maximal depth, of T_p (otherwise x would be in T_p); and every possible state has the same probability. It is a bit more complicated for transitions outgoing from a state of T_p that are not already part of the tree, but we will not use them in our construction; except for p itself, but if we condition on having T_p, its outgoing transitions ends in uniform elements of $[n]$. So as long as we do not consider a transition outgoing from a node of T_p, except p, we can easily perform our probabilistic computations given the $\sqrt{n}$-backward tree of p being T_p. Since the $\sqrt{n}$-backward tree of p of a transition structure is unique if it exists, we can use the law of total probabilities at the end to complete the proof.

Also observe that we cannot hope for a result with high probability in our setting: the probability that p has no incoming transition is $(1 - \frac{1}{n})^{2(n-1)} \approx e^{-2}$ and is therefore visible.

4.2 Forward tree and existence of a small circuit

We fix the $\sqrt{n}$-backward tree T_p of p that satisfies the conditions of Lemma 5. Then, we generate the a-transition $p \xrightarrow{a} r$ outgoing from p: as explained in the previous section, this is a uniform random element of $[n]$. We then begin a process consisting in doing a breadth-first

traversal of the transition structure starting from $r_0 := r$. We discover the states $r_0 = \delta(r, \varepsilon)$, $r_1 = \delta(r, a)$, $r_2 = \delta(r, b)$, $r_3 = \delta(r, aa)$, $r_4 = \delta(r, ab)$, ..., where the words are taken in length-lexicographic order. We continue this process until we reach either some r_i that belongs to T_p, or an already seen r_i ($r_i = r_j$ for some $j < i$). The process is successful if we halt because we hit an extremal leaf of T_p after at most $\sqrt{n}$ steps, otherwise it fails.

Let L_p be the set of extremal leaves of T_p. As mentioned above, since we only discover new states before the last step of the process, the transition considered at time $i \geq 1$ ends in a uniform random state of $[n] \setminus (T_p \setminus L_p)$: the fact that T_p is the $\sqrt{n}$-backward tree from p prevents transitions from ending at a node of $T_p \setminus L_p$ (the case of time 0 is easily handled separately). Hence we are in a variant of the Birthday Problem: we have a target set L_p of size $\Theta(\sqrt{n})$ and we iteratively draw random numbers of $[n] \setminus (T_p \setminus L_p)$ until we hit L_p (success) or we see an element twice (failure). All the computations are classical even if we ask that the process halts before $\sqrt{n}$ steps. In particular $|[n] \setminus (T_p \setminus L_p)| = n - O(\sqrt{n})$ so we do not differ much from the standard case with parameter n. This yields:

▶ **Lemma 6.** *For the uniform distribution on size-n transition structures having T_p as $\sqrt{n}$-backward tree from p, with visible probability the breadth-first traversal starting at $r := \delta_a(p)$ hits an extremal leaf of T_p before it discovers the same state twice, and it does this in at most $\sqrt{n}$ steps.*

If the conclusions of Lemma 6 hold then there is a word w of length $\Theta(\log n)$ such that $\delta_w(r) = p$, and aw labels a circuit around p: starting from p, we read a to reach r, then we follow the path that hits an extremal leaf of T_p, discovered during the breadth-first traversal; then finally go back to p using the transitions of T_p. Observe that there can be several paths that work in the last part: it is possible that both transitions outgoing from a state at distance $i+1$ from p end in states at distance i. To uniquely determine w, we choose, in this last part, the smallest for the lexicographic order. Doing this still preserves uniqueness in the following sense: for a given transition structure, there is at most one triplet (T_p, r, F_r) such that T_p is the $\sqrt{n}$-backward tree from p, $r = \delta_a(p)$, and F_r is the forward tree from r, and all the properties of Lemma 5 and Lemma 6 are satisfied. The choice of w is then fixed by (T_p, r, F_r), and the uniqueness of the triplet, which exists when all the requirement are fulfilled, allows the use of the law of total probabilities.

Let $p \in [n]$. An n-state transition structure is *p-compatible* if its $\sqrt{n}$-backward tree from p exists and satisfies the conclusions of Lemma 5, and if the breadth-first traversal from r discovers different states that are not in T_p for all labels smaller than z, and $\delta(r, z) \in L_p$, with $|z| \leq \frac{1}{2}\log_2 n$. When the transition structure $\mathcal{T}$ is p-compatible, we define its *p-substructure* as being the incomplete automaton whose states are the states in T_p together with r and all the other states discovered during the breadth-first traversal until label z. Its transitions are the transitions of T_p, and all the transitions of the breadth-first search until label z (included). We have:

▶ **Proposition 7.** *With visible probability, an n-state transition structure taken uniformly at random is p-compatible, where p is also taken uniformly at random and independently in $[n]$. In this case, the p-substructure is uniquely determined, has $O(\sqrt{n})$ states, and contains a circuit around p labelled aw, where w is uniquely determined using the transitions of the p-structure only and we have $|w| \in \Theta(\log n)$.*

4.3 Discovering the b-threads

Fix a p-substructure X_p and consider the uniform distribution over n-state transition structures that are p-compatible with X_p. For this distribution, if we take a state $s \notin X_p$, its outgoing transitions end in an element of $[n] \setminus (T_p \setminus L_p)$, uniformly at random and independently from the others transitions. Otherwise, the state s would be in the $\sqrt{n}$-backward-tree of p.

We now add a random a-transition $p \xrightarrow{a} q$ to form a random almost deterministic transition structure that has X_p as p-substructure, by picking uniformly at random $q \in [n]$. Since $|X_p| \in O(\sqrt{n})$, with high probability $q \notin X_p$. We fix some $d \geq 1$ from now on, and read, letter by letter, the word $w(aw)^{d-1}$ starting from q, where aw labels the circuit around p in X_p given in Proposition 7. Since w has length $\Theta(\log n)$, the word $w(aw)^{d-1}$ has logarithmic length, and, using the Birthday Problem once again, with high probability we only discover new states that are not in X_p while reading the whole word. In this case, we name $p_0 = p$ and $p_i = \delta(q, w(aw)^{i-1})$ for $i \in [d]$. Observe that in the whole process, we never considered b-transitions starting from one of the p_i, with $0 \leq i \leq d$. Since $p_0 = p$ is the root of X_p, there are no constraints on its out-going transitions, thus $\delta(p_0, b)$ is a uniform random element of $[n]$. Moreover, for $i \geq 1$ each $p_i \notin X_p$ and, under our conditioning, its outgoing transitions are uniform random elements of $[n] \setminus (T_p \setminus L_p)$.

Let us define the *b-thread* of p_i as the set of all states reached from p_i using words of the form b^j. Discovering state by state such a b-thread consists in iteratively generating the outgoing b-transition of the previous state, which is done by taking a uniform element of $[n] \setminus (T_p \setminus L_p)$. Let us start with the b-thread of p_0. By the Birthday Problem again, with visible probability it cycles back after discovering between $\sqrt{n}$ and $2\sqrt{n}$ states while never discovering a state of X_p, since $|X_p| \in O(\sqrt{n})$. If this happens, we consider the b-thread from p_1. With visible probability, it also cycles back after discovering between $\sqrt{n}$ and $2\sqrt{n}$ states while never discovering a state of X_p or of the b-thread from p_0, as they both have size in $O(\sqrt{n})$. Since d is fixed, doing this for the b-thread starting at each p_i we obtain:

▶ **Lemma 8.** *Let $d \geq 1$. Let X_p be a p-substructure of size-n transition structures. For the uniform distribution on size-n transition structures that are p-compatible and that have X_p as p-substructure, if we add a random transition $p \xrightarrow{a} q$ by choosing q uniformly at random and independently in $[n]$, then with visible probability (i) the states discovered while following the path labeled by $w(aw)^{d-1}$ are all different and do not belong to X_p (ii) the b-threads starting at the p_i's, where $p_0 = p$ and $p_i = \delta(q, w(aw)^{i-1})$, have length between $\sqrt{n}$ and $2\sqrt{n}$, are pairwise disjoint and do not intersect X_p.*

4.4 Cycle lengths and accessibility

An almost deterministic transition structure that satisfies the conditions of Lemma 8 is called *(p, b)-compatible*, and we say that it has b-thread lengths $\vec{\lambda} = (\lambda_0, \ldots, \lambda_d)$ if the b-thread from each p_i has length λ_i. We also define its *(p, b)-substructure* as its p-substructure where we add the states along the path labeled by $w(aw)^{d-1}$ from q and the b-threads from each p_i.

Consider an almost deterministic transition structure $\mathcal{T}$ of given (p, b)-substructure $X_{p,b}$ with b-thread lengths $\vec{\lambda} = (\lambda_0, \ldots, \lambda_d)$ and cycle lengths $\vec{\ell} = (\ell_0, \ldots, \ell_d)$. If $\vec{\ell'} = (\ell'_0, \ldots, \ell'_d)$ is another vector where each $\ell'_i \in [\lambda_i]$, we can re-target the last b-transition of each b-thread so that the cycle lengths are now $\vec{\ell'}$. Thus, conditioned on $\vec{\lambda}$, each cycle length ℓ_i is a uniform random element of $[\lambda_i]$. Since $\sqrt{n} \leq \lambda_i \leq 2\sqrt{n}$, and since each $\ell_i \in [\![\frac{1}{2}\sqrt{n}, \sqrt{n}]\!]$, with visible probability the ℓ_i's are uniform and independent random elements of $[\![\frac{1}{2}\sqrt{n}, \sqrt{n}]\!]$.

To conclude this part, we generate the initial state i_0 uniformly at random. All our constraints so far hold with visible probability, and one of them implies the existence of a circuit of length $\Omega(\log n)$ around p. Cai and Devroye [5] established that with high probability such a cycle is accessible; the conjunction of a high-probability event with a visible event is still visible. This yields:

▶ **Theorem 9.** *Let $d \geq 1$. There exists a set $\mathfrak{T}_n$ of almost deterministic transition structures with n states and one initial state such that with visible probability for the uniform distribution over size-n almost deterministic transition structure with an initial state, the state p (source*

of the additional a-transition) is accessible from the initial state and there exists a word w of length $\Theta(\log n)$ such that $\gamma(p, w(aw)^{d-1}) = \{p_0, \dots, p_d\}$ is a set of $d+1$ states, and the b-threads starting from the p_i's have lengths λ_i in $[\![\sqrt{n}, 2\sqrt{n}\,]\!]$ and their cycle length is in $[\![\frac{1}{2}\sqrt{n}, \sqrt{n}\,]\!]$. Moreover, this set $\mathfrak{T}_n$ can be built so that for the uniform distribution on $\mathfrak{T}_n$, the cycle lengths are uniform and independent random elements of $[\![\frac{1}{2}\sqrt{n}, \sqrt{n}\,]\!]$.

If $\mathcal{T}$ is in the set $\mathfrak{T}_n$ and we read b's from $P = \{p_0, \dots, p_d\}$, we eventually reach the b-cycle of P in the accessible powerset transition structure of $\mathcal{T}$, and its length is $\mathrm{lcm}(\ell_0, \dots, \ell_d)$. As the ℓ_i's are uniform and independent random elements of $[\![\frac{1}{2}\sqrt{n}, \sqrt{n}]\!]$, their lcm is $\Omega(n^{\frac{d+1}{2}})$ with visible probability [11], yielding our first main consequence (before adding final states):

▶ **Corollary 10.** *For the uniform distribution on size-n almost deterministic transition structures, the accessible powerset transition structure has a super-polynomial number of states with visible probability.*

5 Adding final states

We are now ready to randomly select which states are final. In our model, for every n, each state is final with fixed probability f_n, which may depend on n as long as it is not too close to either 0 or 1: we require that a set of $\Theta(\sqrt{n})$ states contains both final and non-final states with visible probability. This holds under our condition that f_n and $1 - f_n$ are in $\Omega(\frac{1}{\sqrt{n}})$, as a variant of the Birthday Problem again.

Previously, we exhibited the existence with visible probability of $d+1$ occurrences of b-cycles in a random almost deterministic transition structure, yielding a large b-cycle when applying the powerset construction. We will focus on b-cycles in the sequel, as it turns out to be sufficient to prove our main result. It relies on the notion of primitive words, which we now recall.

Let Γ be a nonempty finite alphabet. If $w \in \Gamma^\ell$ is a word of length ℓ, we write $w = w_0 \cdots w_{\ell-1}$ and use the convention that all indices are taken modulo ℓ: for instance w_ℓ is the letter w_0. A nonempty word w is *primitive* if it is not a non-trivial power of another word: it cannot be written $w = z^k$ for some word z and some $k \geq 2$. If w is primitive, it is easily seen that every circular permutation of w is also primitive. See [16] for a more detailed account on primitive words.

Primitive words appear in our proof with the following observation. If $\mathcal{C} = (c_0, \dots, c_{\ell-1})$ is a b-cycle of states starting at c_0, its *associated word* is the size-ℓ word $v = v_0 \dots v_{\ell-1}$ of $\{0,1\}^\ell$ where $v_i = 1$ if and only if c_i is a final state. Recall that if we start the same cycle elsewhere, at c_i, the associated word $v' = v_i \cdots v_\ell v_0 \cdots v_{i-1}$ is primitive if and only if v is primitive: reading the associated word from any starting state preserves primitivity. A b-cycle is said to be *primitive* if one (equivalently, all) of its associated words is (are) primitive. Our study is based on the following statement.

▶ **Lemma 11.** *Let $\mathcal{A}$ be a deterministic automaton on Σ and $\alpha \in \Sigma$. If $\mathcal{C}$ is a primitive α-cycle of $\mathcal{A}$, then the states of $\mathcal{C}$ are pairwise non-equivalent: the state complexity of the language recognized by $\mathcal{A}$ is at least $|\mathcal{C}|$.*

So we reduced our problem to studying the primitivity of the b-cycles we built in Section 4, and to how it exports to the associated b-cycle in the powerset construction.

5.1 Some properties of primitive words

If $w^{(1)}$ and $w^{(2)}$ are two non-empty words of respective lengths ℓ_1 and ℓ_2 on the binary alphabet $\{0,1\}$, we denote by $w^{(1)} \odot w^{(2)}$ the word w of length $\ell = \mathrm{lcm}(\ell_1, \ell_2)$ given by $w_i = 1$ if and only if $w_i^{(1)} = 1$ or $w_i^{(2)} = 1$ (recall that the indices are taken modulo the

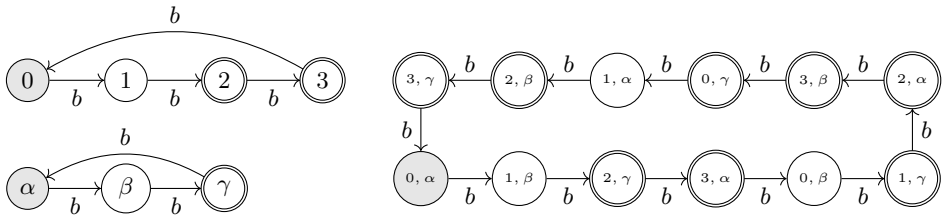


Figure 3 On the left, two primitive b-cycles (accepting states are denoted by double circles) whose associated words are 0011 (top) and 001 (bottom), starting at 0 and α, respectively. On the right, the b-cycle of $\{0, \alpha\}$ of associated word $0011 \odot 001 = 001101111011$, which is primitive by Lemma 12.

length of the word). We will see in the sequel that this operation naturally happens when extending the notion of state equivalence from each b-cycle to the corresponding b-cycle in the powerset construction.

▶ **Lemma 12.** *Let $w^{(1)}$ and $w^{(2)}$ be two primitive words on $\{0,1\}$ of lengths at least 2 that are coprime. Then, the word $w^{(1)} \odot w^{(2)}$ is primitive.*

▶ Remark 13. Lemma 12 does not hold if the lengths are not coprime. For instance, if $w^{(1)} = 011111$ and $w^{(2)} = 1011$, then $w^{(1)} \odot w^{(2)} = \underbrace{1 \dots 1}_{12 \text{ times}}$, which is not primitive.

From a probabilistic point of view, it is well known [16] that a uniform random word is primitive with very high probability. We rely on the following finer result.

▶ **Lemma 14** (De Felice, Nicaud [11])**.** *Let μ be a probability measure on $\{0,1\}^n$ such that $\mu(0^n) = \mu(1^n) = 0$ and such that two words with the same number of 0's have same probability. Then, the probability that a word is not primitive under μ is at most $\frac{2}{n}$.*

We adapt it to our needs as follows:

▶ **Corollary 15.** *Let f_n be a sequence of real numbers in $(0,1)$ such that $f_n = \Omega(\frac{1}{\sqrt{n}})$ and $1 - f_n = \Omega(\frac{1}{\sqrt{n}})$. Let ℓ be an integer greater than $\alpha\sqrt{n}$, for a fixed α, and let w be a random binary word of length ℓ whose letters are 1's with probability f_n and 0 with probability $1 - f_n$, independently. Then, w is primitive with visible probability.*

5.2 Finalizing the proof of Theorem 3

By Lemma 12, primitivity is preserved by the product $\odot$ when the lengths are coprime, so we restrict the cycle lengths built in Section 4 so that they are pairwise coprime. By Theorem 9, these lengths are uniform random elements of $[\![\frac{1}{2}\sqrt{n}, \sqrt{n}]\!]$, we therefore adapt a known result of probabilistic number theory to prove that it still happens with visible probability.

More precisely, Tóth established [20] that the probability that $d+1$ integer taken uniformly at random and independently in $[n]$ are pairwise coprime tends to some positive constant A_{d+1}, generalizing the folklore result that two independent random numbers in $[n]$ are coprime with probability that tends to $\frac{6}{\pi^2}$. This can be used to derive the following variant:

▶ **Corollary 16.** *Let $\ell_0, \ell_1, \dots, \ell_d$ be $d+1$ integers taken uniformly at random and independently in $[\![\frac{1}{2}\sqrt{n}, \sqrt{n}]\!]$. With visible probability, the ℓ_i's are pairwise coprime.*

Combining Corollary 15 and Corollary 16, we can extend Theorem 9 to also require that the b-cycles are primitive and their lengths are pairwise coprime. And this still happens with visible probability.

We can then conclude as follows: if all these requirements are met, the state p is accessible and there is a word z such that $\delta(p, z) = \{p_0, \dots, p_d\}$, the b-threads of the p_i's are pairwise disjoint and eventually form cycles of respective pairwise coprime lengths ℓ_i, and each such cycle is primitive. Moreover, all the ℓ_i are in $\Theta(\sqrt{n})$. By a direct induction on Lemma 12, this yields that the b-cycle of $\{p_0, \dots, p_d\}$ in the powerset automaton is primitive and has length $\Theta(\sqrt{n}^{d+1})$. By Lemma 11, the language recognized by this almost deterministic automaton has state complexity at least $\Theta(n^{\frac{d+1}{2}})$. This concludes the proof, as it holds for every fixed d.

6 Conclusion and discussion

Our main theorem states that the state complexity of a random almost deterministic automaton is greater than n^d with probability at least $c_d > 0$ for n sufficiently large. One can wonder how small the constant c_d is and for which sizes the lower-bound holds. As we said in the introduction, we did not try to estimate c_d nor did we try to optimize its value in this article. Since the powerset construction quickly generates very large automata which would need to be minimized, a proper experimental study does not seem feasible. However, we did generate 1000 almost deterministic transition structures with $n = 100$ states and apply the accessible powerset construction: in 78.6% of the 1000 cases the output had more than n^3 states. This would lead us to guess that even if the constant c_3 that can be derived from our proof is very small, combinatorial explosion does occur frequently in practice.

Also, as noticed above, in our setting it is certain that the property does not hold with high probability, as there is an asymptotically constant probability that the source of the added transition is not accessible. However, this probability is roughly 20.4%, not too far from what we obtained in our experiment on size-100 structures: it is very possible that if we condition the source of the added transition to be accessible, then our result holds with high probability. However, our proof techniques, based on an intensive use of the Birthday Problem cannot prove this: completely new ideas are necessary to establish such a result.

Another natural direction is to consider the case when there are *few* final states, as $\Theta(\sqrt{n})$ final states may be considered too large for a random deterministic automaton. The extreme case is to allow exactly one final state by choosing it uniformly at random. If we do so, our analysis using primitive words fails: with high probability the b-cycles we built have no final state at all, and neither has the associated b-cycle $\mathcal{C}$ in the powerset construction. However, we are confident that our techniques can be used to capture this distribution: by studying the paths ending in this final state, we should be able to find for each b-cycle $\mathcal{C}_i$ a word w_i that maps exactly one state to the final state, and such that the w_i are all different. This would be enough to establish that the states of $\mathcal{C}$ are pairwise non-equivalent and prove the conjecture. Completely formalizing and proving this idea is an ongoing work.

References

1 Louigi Addario-Berry, Borja Balle, and Guillem Perarnau Llobet. Diameter and stationary distribution of random r-out digraphs. *Electronic journal of combinatorics*, 27(P3. 28):1–41, 2020.

2 Frédérique Bassino, Julien David, and Cyril Nicaud. Average case analysis of Moore's state minimization algorithm. *Algorithmica*, 63(1-2):509–531, 2012. doi:10.1007/s00453-011-9557-7.

3 Frédérique Bassino, Julien David, and Andrea Sportiello. Asymptotic enumeration of minimal automata. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 88–99. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.88.

4 Mikhail V. Berlinkov. On the probability of being synchronizable. In Sathish Govindarajan and Anil Maheshwari, editors, *Algorithms and Discrete Applied Mathematics – Second International Conference, CALDAM 2016, Thiruvananthapuram, India, February 18-20, 2016, Proceedings*, volume 9602 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2016. doi:10.1007/978-3-319-29221-2_7.

5 Xing Shi Cai and Luc Devroye. The graph structure of a deterministic automaton chosen at random. *Random Structures & Algorithms*, 51(3):428–458, 2017.

6 Arnaud Carayol and Cyril Nicaud. Distribution of the number of accessible states in a random deterministic automaton. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th – March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 194–205. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.194.

7 Guillaume Chapuy and Guillem Perarnau. Short synchronizing words for random automata. *CoRR*, abs/2207.14108, 2022. doi:10.48550/arXiv.2207.14108.

8 Julien David. Average complexity of Moore's and Hopcroft's algorithms. *Theor. Comput. Sci.*, 417:50–65, 2012. doi:10.1016/j.tcs.2011.10.011.

9 Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

10 Sven De Felice and Cyril Nicaud. Brzozowski algorithm is generically super-polynomial for deterministic automata. In Marie-Pierre Béal and Olivier Carton, editors, *Developments in Language Theory – 17th International Conference, DLT 2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings*, volume 7907 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2013. doi:10.1007/978-3-642-38771-5_17.

11 Sven De Felice and Cyril Nicaud. Average case analysis of Brzozowski's algorithm. *Int. J. Found. Comput. Sci.*, 27(2):109–126, 2016. doi:10.1142/S0129054116400025.

12 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

13 Aleksandr Aleksandrovich Grusho. Limit distributions of certain characteristics of random automaton graphs. *Mathematical Notes of the Academy of Sciences of the USSR*, 14(1):633–637, 1973.

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

15 Florent Koechlin, Cyril Nicaud, and Pablo Rotondo. Simplifications of uniform expressions specified by systems. *Int. J. Found. Comput. Sci.*, 32(6):733–760, 2021.

16 Lothaire. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, 2 edition, 1997. doi:10.1017/CBO9780511566097.

17 Albert R Meyer and Michael J Fischer. Economy of description by automata, grammars, and formal systems. In *12th Annual Symposium on Switching and Automata Theory (SWAT 1971)*, pages 188–191. IEEE Computer Society, 1971.

18 Cyril Nicaud. Random deterministic automata. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 5–23. Springer, 2014. doi:10.1007/978-3-662-44522-8_2.

19 Cyril Nicaud. The Černý conjecture holds with high probability. *J. Autom. Lang. Comb.*, 24(2-4):343–365, 2019. doi:10.25596/jalc-2019-343.

20 László Tóth. The probability that k positive integers are pairwise relatively prime. *Fibonacci Quart*, 40:13–18, 2002.

Improved NP-Hardness of Approximation for Orthogonality Dimension and Minrank

Dror Chawin
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Ishay Haviv
School of Computer Science, The Academic College of Tel Aviv-Yaffo, Tel Aviv, Israel

Abstract

The orthogonality dimension of a graph G over $\mathbb{R}$ is the smallest integer k for which one can assign a nonzero k-dimensional real vector to each vertex of G, such that every two adjacent vertices receive orthogonal vectors. We prove that for every sufficiently large integer k, it is NP-hard to decide whether the orthogonality dimension of a given graph over $\mathbb{R}$ is at most k or at least $2^{(1-o(1))\cdot k/2}$. We further prove such hardness results for the orthogonality dimension over finite fields as well as for the closely related minrank parameter, which is motivated by the index coding problem in information theory. This in particular implies that it is NP-hard to approximate these graph quantities to within any constant factor. Previously, the hardness of approximation was known to hold either assuming certain variants of the Unique Games Conjecture or for approximation factors smaller than 3/2. The proofs involve the concept of line digraphs and bounds on their orthogonality dimension and on the minrank of their complement.

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

Keywords and phrases hardness of approximation, graph coloring, orthogonality dimension, minrank, index coding

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.20

Related Version *Full Version*: http://arxiv.org/abs/2301.00732

Funding *Dror Chawin*: 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 helpful comments.

1 Introduction

A graph G is said to be k-colorable if its vertices can be colored by k colors such that every two adjacent vertices receive distinct colors. The chromatic number of G, denoted by $\chi(G)$, is the smallest integer k for which G is k-colorable. As a fundamental and popular graph quantity, the chromatic number has received a considerable amount of attention in the literature from a computational perspective, as described below.

The problem of deciding whether a graph G satisfies $\chi(G) \leq 3$ is one of the classical twenty-one NP-complete problems presented by Karp [22] in 1972. Khanna, Linial, and Safra [24] proved that it is NP-hard to distinguish between graphs G that satisfy $\chi(G) \leq 3$ from those satisfying $\chi(G) \geq 5$. This result, combined with the approach of Garey and Johnson [11] and with a result of Stahl [33], implies that for every $k \geq 6$, it is NP-hard to decide whether a graph G satisfies $\chi(G) \leq k$ or $\chi(G) \geq 2k-2$. Brakensiek and Guruswami [5] proved that for every $k \geq 3$, it is NP-hard to distinguish between the cases $\chi(G) \leq k$ and $\chi(G) \geq 2k-1$, and the $2k-1$ bound was further improved to $2k$ by Barto, Bulín, Krokhin, and Opršal [3]. For large values of k, it was shown by Khot [25] that it is NP-hard to decide whether a graph G satisfies $\chi(G) \leq k$ or $\chi(G) \geq k^{\Omega(\log k)}$, and the latter condition


© Dror Chawin and Ishay Haviv;
licensed under Creative Commons License CC-BY 4.0


was strengthened to $\chi(G) \geq 2^{k^{1/3}}$ by Huang [20]. A substantial improvement was recently obtained by Wrochna and Živný [34], who proved that for every $k \geq 4$, it is NP-hard to decide whether a given graph G satisfies $\chi(G) \leq k$ or $\chi(G) \geq \binom{k}{\lfloor k/2 \rfloor}$. The proof of this result combined the hardness result of [20] with the construction of line digraphs [16] and with a result of Poljak and Rödl [31]. Note that under certain variants of the Unique Games Conjecture, stronger hardness results are known to hold, namely, hardness of deciding whether a given graph G satisfies $\chi(G) \leq k_1$ or $\chi(G) \geq k_2$ for all integers $k_2 > k_1 \geq 3$ [9] (see also [10]).

The present paper studies the computational complexity of algebraic variants of the chromatic number of graphs. A k-dimensional orthogonal representation of a graph $G = (V, E)$ over a field $\mathbb{F}$ is an assignment of a vector $u_v \in \mathbb{F}^k$ with $\langle u_v, u_v \rangle \neq 0$ to each vertex $v \in V$, such that for every two adjacent vertices v and v' it holds that $\langle u_v, u_{v'} \rangle = 0$. Here, for two vectors $x, y \in \mathbb{F}^k$, we consider the standard inner product defined by $\langle x, y \rangle = \sum_{i=1}^{k} x_i y_i$ with operations over $\mathbb{F}$. The orthogonality dimension of G over $\mathbb{F}$, denoted by $\overline{\xi}_{\mathbb{F}}(G)$, is the smallest integer k for which G admits a k-dimensional orthogonal representation over $\mathbb{F}$ (see Remark 4). It can be easily seen that for every graph G and for every field $\mathbb{F}$, it holds that $\overline{\xi}_{\mathbb{F}}(G) \leq \chi(G)$. In addition, if $\mathbb{F}$ is a fixed finite field or the real field $\mathbb{R}$, it further holds that $\overline{\xi}_{\mathbb{F}}(G) \geq \Omega(\log \chi(G))$. Both bounds are known to be tight in the worst case (see Claim 8 and [28, Chapter 10]). The study of orthogonal representations and orthogonality dimension was initiated in the seminal work of Lovász [27] on the ϑ-function and has found applications in various areas, e.g., information theory [27], graph theory [29], and quantum communication complexity [8, Chapter 8.5].

The interest in the hardness of determining the orthogonality dimension of graphs dates back to a paper of Lovász, Saks, and Schrijver [29], where it was noted that the problem seems difficult. The aforementioned relations between the chromatic number and the orthogonality dimension yield that hardness of deciding whether a graph G satisfies $\chi(G) \leq k_1$ or $\chi(G) \geq k_2$ implies the hardness of deciding whether it satisfies $\overline{\xi}_{\mathbb{F}}(G) \leq k_1$ or $\overline{\xi}_{\mathbb{F}}(G) \geq \Omega(\log k_2)$, provided that $\mathbb{F}$ is a finite field or $\mathbb{R}$. It therefore follows from [9] that assuming certain variants of the Unique Games Conjecture, it is hard to decide whether a graph G satisfies $\overline{\xi}_{\mathbb{F}}(G) \leq k_1$ or $\overline{\xi}_{\mathbb{F}}(G) \geq k_2$ for all integers $k_2 > k_1 \geq 3$. This reasoning, however, does not yield NP-hardness results for the orthogonality dimension (without additional complexity assumptions), even using the strongest known NP-hardness results of the chromatic number. Yet, a result of Peeters [30] implies that for every field $\mathbb{F}$, it is NP-hard to decide if a given graph G satisfies $\overline{\xi}_{\mathbb{F}}(G) \leq 3$, hence it is NP-hard to approximate the orthogonality dimension of a graph over $\mathbb{F}$ to within any factor smaller than $4/3$. Over the reals, the hardness of approximation for the orthogonality dimension was recently extended in [12] to any factor smaller than $3/2$.

Another algebraic quantity of graphs is the minrank parameter that was introduced in 1981 by Haemers [15] in the study of the Shannon capacity of graphs. The minrank parameter was used in [14, 15] to answer questions of Lovász [27] and was later applied by Alon [1], with a different formulation, to disprove a conjecture of Shannon [32]. The minrank of a graph G over a field $\mathbb{F}$, denoted by $\mathrm{minrk}_{\mathbb{F}}(G)$, is closely related to the orthogonality dimension of the complement graph $\overline{G}$ over $\mathbb{F}$ and satisfies $\mathrm{minrk}_{\mathbb{F}}(G) \leq \overline{\xi}_{\mathbb{F}}(\overline{G})$. The difference between the two quantities comes, roughly speaking, from the fact that the definition of minrank involves the notion of orthogonal bi-representations rather than orthogonal representations (for the precise definitions, see Section 2.1). The study of the minrank parameter is motivated by various applications in information theory and in theoretical computer science. A prominent one is the well-studied index coding problem, for which the minrank parameter perfectly characterizes the optimal length of its linear solutions, as was shown by Bar-Yossef, Birk, Jayram, and Kol [2] (see Section 2.2).

Similarly to the situation of the orthogonality dimension, it was proved in [30] that for every field $\mathbb{F}$, it is NP-hard to decide if a given graph G satisfies $\text{minrk}_{\mathbb{F}}(G) \leq 3$. It was further shown by Dau, Skachek, and Chee [7] that it is NP-hard to decide whether a given digraph G satisfies $\text{minrk}_{\mathbb{F}_2}(G) \leq 2$. Note that for (undirected) graphs, the minrank over any field is at most 2 if and only if the complement graph is bipartite, a property that can be checked in polynomial time. Motivated by the computational aspects of the index coding problem, Langberg and Sprintson [26] related the minrank of a graph to the chromatic number of its complement and derived from [9] that assuming certain variants of the Unique Games Conjecture, it is hard to decide whether a given graph G satisfies $\text{minrk}_{\mathbb{F}}(G) \leq k_1$ or $\text{minrk}_{\mathbb{F}}(G) \geq k_2$, provided that $k_2 > k_1 \geq 3$ and that $\mathbb{F}$ is a finite field. Similar hardness results were obtained in [26] for additional settings of the index coding problem, including the general (non-linear) index coding problem over a constant-size alphabet.

1.1 Our Contribution

This paper provides improved NP-hardness of approximation results for the orthogonality dimension and for the minrank parameter over various fields. We start with the following result, which is concerned with the orthogonality dimension over the reals.

▶ **Theorem 1.** *There exists a function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(k) = 2^{(1-o(1))\cdot k/2}$ such that for every sufficiently large integer k, it is NP-hard to decide whether a given graph G satisfies $\overline{\xi}_{\mathbb{R}}(G) \leq k$ or $\overline{\xi}_{\mathbb{R}}(G) \geq f(k)$.*

Theorem 1 implies that it is NP-hard to approximate the orthogonality dimension of a graph over the reals to within any constant factor. Previously, such NP-hardness result was known to hold only for approximation factors smaller than 3/2 [12].

We proceed with the following result, which is concerned with the orthogonality dimension and the minrank parameter over finite fields.

▶ **Theorem 2.** *For every finite field $\mathbb{F}$, there exists a function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(k) = 2^{(1-o(1))\cdot k/2}$ such that for every sufficiently large integer k, the following holds.*

1. *It is NP-hard to decide whether a graph G satisfies $\overline{\xi}_{\mathbb{F}}(G) \leq k$ or $\overline{\xi}_{\mathbb{F}}(G) \geq f(k)$.*
2. *It is NP-hard to decide whether a graph G satisfies $\text{minrk}_{\mathbb{F}}(G) \leq k$ or $\text{minrk}_{\mathbb{F}}(G) \geq f(k)$.*

Theorem 2 implies that over any finite field, it is NP-hard to approximate the orthogonality dimension and the minrank of a graph to within any constant factor. Let us stress that this hardness result relies solely on the assumption $\mathsf{P} \neq \mathsf{NP}$ rather than on stronger complexity assumptions and thus settles a question raised in [26]. Prior to this work, it was known that it is NP-hard to approximate the minrank of graphs to within any factor smaller than 4/3 [30] and the minrank of digraphs over $\mathbb{F}_2$ to within any factor smaller than 3/2 [7].

A central component of the proofs of Theorems 1 and 2 is the notion of line digraphs, introduced in [16], that was first used in the context of hardness of approximation by Wrochna and Živný [34] (see also [13]). It was shown in [17, 31] that the chromatic number of any graph is exponential in the chromatic number of its line digraph. This result was iteratively applied by the authors of [34] to improve the NP-hardness of the chromatic number from the k vs. $2^{k^{1/3}}$ gap of [20] to their k vs. $\binom{k}{\lfloor k/2 \rfloor}$ gap. The main technical contribution of the present work lies in analyzing the orthogonality dimension of line digraphs and the minrank parameter of their complement. We actually show that on line digraphs, these graph parameters are quadratically related to the chromatic number (see Theorems 15, 17, and 20). This allows us to derive our hardness results from the hardness of the chromatic number given in [34], where the obtained gaps are only quadratically weaker. We further discuss some limitations of our approach, involving an analogue of Sperner's theorem for subspaces due to Kalai [21].

We finally show that our approach might be useful for proving hardness results for the general (non-linear) index coding problem over a constant-size alphabet, for which no NP-hardness result is currently known. It was shown by Langberg and Sprintson [26] that for an instance of the index coding problem represented by a graph G, the length of an optimal solution is at most $\chi(\overline{G})$ and at least $\Omega(\log\log\chi(\overline{G}))$. It thus follows that an NP-hardness result for the chromatic number with a double-exponential gap would imply an NP-hardness result for the general index coding problem. However, no such NP-hardness result is currently known for the chromatic number without relying on further complexity assumptions. To tackle this issue, we study the index coding problem on instances which are complement of line digraphs (see Theorem 22). As a consequence of our results, we obtain that the NP-hardness of the general index coding problem can be derived from an NP-hardness result of the chromatic number with only a single-exponential gap, not that far from the best known gap given in [34]. For a precise statement, see Theorem 29.

1.2 Related Work

We gather here several related results from the literature.

- A result of Zuckerman [35] asserts that for any $\varepsilon > 0$, it is NP-hard to approximate the chromatic number of a graph on n vertices to within a factor of $n^{1-\varepsilon}$. It would be interesting to figure out if such hardness result holds for the orthogonality dimension and for the minrank parameter. The present paper, however, focuses on the hardness of gap problems with constant thresholds, independent of the number of vertices.
- As mentioned earlier, Peeters [30] proved that for every field $\mathbb{F}$, it is NP-hard to decide if the minrank (or the orthogonality dimension) of a given graph is at most 3. We note that for finite fields, this can also be derived from a result of Hell and Nešetřil [19].
- For the chromatic number of hypergraphs, the gaps for which NP-hardness is known to hold are much stronger than for graphs. For example, it was shown in [4] that for some $\delta > 0$, it is NP-hard to decide if a given 4-uniform hypergraph G on n vertices satisfies $\chi(G) \leq 2$ or $\chi(G) \geq \log^{\delta} n$. An analogue result for the orthogonality dimension of hypergraphs over $\mathbb{R}$ was proved in [18].
- On the algorithmic side, a long line of work has explored the number of colors that an efficient algorithm needs for properly coloring a given k-colorable graph, where $k \geq 3$ is a fixed constant. For example, there exists a polynomial-time algorithm that on a given 3-colorable graph with n vertices uses $O(n^{0.19996})$ colors [23]. Algorithms of this nature exist for the graph parameters studied in this work as well. Indeed, there exists a polynomial-time algorithm that given a graph G on n vertices with $\overline{\xi}_{\mathbb{R}}(G) \leq 3$ finds a proper coloring of G with $O(n^{0.2413})$ colors [18]. Further, there exists a polynomial-time algorithm that given a graph G on n vertices with $\mathrm{minrk}_{\mathbb{F}_2}(\overline{G}) \leq 3$ finds a proper coloring of G with $O(n^{0.2574})$ colors [6]. Note that the colorings obtained by these two algorithms provide, respectively, orthogonal and bi-orthogonal representations for the input graph G (see Claim 8).

1.3 Outline

The rest of the paper is organized as follows. In Section 2, we collect several definitions and results that will be used throughout this paper. In Section 3, we study the underlying graphs of line digraphs and their behavior with respect to the orthogonality dimension, the minrank parameter, and the index coding problem. We also discuss some limitations of our approach. Finally, in Section 4, we prove our hardness results and complete the proofs of Theorems 1 and 2. Some of the proofs are omitted and can be found in the full version of the paper.

2 Preliminaries

Throughout the paper, undirected graphs are referred to as graphs, and directed graphs are referred to as digraphs. All the considered graphs and digraphs are simple, and all the logarithms are in base 2 unless otherwise specified. For an integer n, we use the notation $[n] = \{1, 2, \ldots, n\}$.

2.1 Orthogonality Dimension and Minrank

The orthogonality dimension of a graph is defined as follows (see, e.g., [28, Chapter 11]).

▶ **Definition 3** (Orthogonality Dimension). *A* k-dimensional orthogonal representation *of a graph* $G = (V, E)$ *over a field* $\mathbb{F}$ *is an assignment of a vector* $u_v \in \mathbb{F}^k$ *with* $\langle u_v, u_v \rangle \neq 0$ *to each vertex* $v \in V$*, such that* $\langle u_v, u_{v'} \rangle = 0$ *whenever* v *and* v' *are adjacent vertices in* G*. Here, for two vectors* $x, y \in \mathbb{F}^k$*, we let* $\langle x, y \rangle = \sum_{i=1}^{k} x_i y_i$ *denote the standard inner product of* x *and* y *over* $\mathbb{F}$*. The* orthogonality dimension *of a graph* G *over a field* $\mathbb{F}$*, denoted by* $\overline{\xi}_{\mathbb{F}}(G)$*, is the smallest integer* k *for which there exists a* k*-dimensional orthogonal representation of* G *over* $\mathbb{F}$*.*

▶ Remark 4. We note that orthogonal representations are sometimes defined in the literature such that the vectors associated with *non-adjacent* vertices are required to be orthogonal, that is, as orthogonal representations of the complement graph. While we find it more convenient to use the other definition in this paper, one can view the notation $\overline{\xi}_{\mathbb{F}}(G)$ as standing for $\xi_{\mathbb{F}}(\overline{G})$, i.e., the orthogonality dimension of the complement graph. The same holds for the notion of orthogonal bi-representations, given in Definition 6.

The minrank parameter, introduced in [15], is defined as follows.

▶ **Definition 5** (Minrank). *Let* $G = (V, E)$ *be a digraph on the vertex set* $V = [n]$*, and let* $\mathbb{F}$ *be a field. We say that a matrix* $M \in \mathbb{F}^{n \times n}$ represents G *if* $M_{i,i} \neq 0$ *for every* $i \in V$*, and* $M_{i,j} = 0$ *for every distinct vertices* $i, j \in V$ *such that* $(i, j) \notin E$*. The* minrank *of* G *over* $\mathbb{F}$ *is defined as* $\mathrm{minrk}_{\mathbb{F}}(G) = \min\{\mathrm{rank}_{\mathbb{F}}(M) \mid M \text{ represents } G \text{ over } \mathbb{F}\}$*. The definition is naturally extended to graphs by replacing every edge with two oppositely directed edges.*

We next describe an alternative definition due to Peeters [30] for the minrank of graphs. This requires the following extension of orthogonal representations, called orthogonal bi-representations.

▶ **Definition 6.** *A* k-dimensional orthogonal bi-representation *of a graph* $G = (V, E)$ *over a field* $\mathbb{F}$ *is an assignment of a pair of vectors* $(u_v, w_v) \in \mathbb{F}^k \times \mathbb{F}^k$ *with* $\langle u_v, w_v \rangle \neq 0$ *to each vertex* $v \in V$*, such that* $\langle u_v, w_{v'} \rangle = \langle u_{v'}, w_v \rangle = 0$ *whenever* v *and* v' *are adjacent in* G*.*

The following proposition follows directly from Definitions 5 and 6 combined with the fact that for every matrix $M \in \mathbb{F}^{n \times n}$, $\mathrm{rank}_{\mathbb{F}}(M)$ is the smallest integer k for which M can be written as $M = M_1^T \cdot M_2$ for two matrices $M_1, M_2 \in \mathbb{F}^{k \times n}$.

▶ **Proposition 7** ([30]). *For every field* $\mathbb{F}$ *and for every graph* G*,* $\mathrm{minrk}_{\mathbb{F}}(\overline{G})$ *is the smallest integer* k *for which there exists a* k*-dimensional orthogonal bi-representation of* G *over* $\mathbb{F}$*.*

The following claim summarizes some known relations between the studied graph parameters.

▷ Claim 8. For every field $\mathbb{F}$ and for every graph G, it holds that $\mathrm{minrk}_{\mathbb{F}}(\overline{G}) \leq \overline{\xi}_{\mathbb{F}}(G) \leq \chi(G)$. In addition, if $\mathbb{F}$ is finite, then $\mathrm{minrk}_{\mathbb{F}}(\overline{G}) \geq \log_{|\mathbb{F}|} \chi(G)$.

We finally recall that a homomorphism from a graph $G_1 = (V_1, E_1)$ to a graph $G_2 = (V_2, E_2)$ is a function $g : V_1 \to V_2$ such that for every two vertices $x, y \in V_1$ with $\{x, y\} \in E_1$, it holds that $\{g(x), g(y)\} \in E_2$. Observe that if there exists a homomorphism from G_1 to G_2 then we have $\chi(G_1) \leq \chi(G_2)$, and for every field $\mathbb{F}$, we have $\overline{\xi}_{\mathbb{F}}(G_1) \leq \overline{\xi}_{\mathbb{F}}(G_2)$ and $\text{minrk}_{\mathbb{F}}(\overline{G_1}) \leq \text{minrk}_{\mathbb{F}}(\overline{G_2})$.

2.2 Index Coding

The index coding problem, introduced in [2], is concerned with economical strategies for broadcasting information to n receivers in a way that enables each of them to retrieve its own message, a symbol from some given alphabet Σ. For this purpose, each receiver is allowed to use some prior side information that consists of a subset of the messages required by the other receivers. The side information map is naturally represented by a digraph on $[n]$, which includes an edge (i, j) if the ith receiver knows the message required by the jth receiver. The objective is to minimize the length of the transmitted information. For simplicity, we consider here the case of symmetric side information maps, represented by graphs rather than by digraphs. The formal definition follows.

▶ **Definition 9** (Index Coding). *Let G be a graph on the vertex set $[n]$, and let Σ be an alphabet. An* index code *for G over Σ of length k is an encoding function $E : \Sigma^n \to \Sigma^k$ such that for every $i \in [n]$, there exists a decoding function $g_i : \Sigma^{k+|N_G(i)|} \to \Sigma$, such that for every $x \in \Sigma^n$, it holds that $g_i(E(x), x|_{N_G(i)}) = x_i$. Here, $N_G(i)$ stands for the set of vertices in G adjacent to the vertex i, and $x|_{N_G(i)}$ stands for the restriction of x to the indices of $N_G(i)$. If Σ is a field $\mathbb{F}$ and the encoding function E is linear over $\mathbb{F}$, then we say that the index code is* linear over $\mathbb{F}$.

Bar-Yossef et al. [2] showed that the minrank parameter characterizes the length of optimal solutions to the index coding problem in the linear setting.

▶ **Proposition 10** ([2]). *For every field $\mathbb{F}$ and for every graph G, the minimal length of a linear index code for G over $\mathbb{F}$ is* $\text{minrk}_{\mathbb{F}}(G)$.

3 Line Digraphs

In 1960, Harary and Norman [16] introduced the concept of line digraphs, defined as follows.

▶ **Definition 11** (Line Digraph). *For a digraph $G = (V, E)$, the* line digraph of G, *denoted by δG, is the digraph on the vertex set E that includes a directed edge from a vertex (x, y) to a vertex (z, w) whenever $y = z$.*

Definition 11 is naturally extended to graphs G by replacing every edge of G with two oppositely directed edges. Note that in this case, the number of vertices in δG is twice the number of edges in G. We will frequently consider the underlying graph of the digraph δG, i.e., the graph obtained from δG by ignoring the directions of the edges.

The following result of Poljak and Rödl [31], which strengthens a previous result of Harner and Entringer [17], shows that the chromatic number of a graph G precisely determines the chromatic number of the underlying graph of δG. The statement of the result uses the function $b : \mathbb{N} \to \mathbb{N}$ defined by $b(n) = \binom{n}{\lfloor n/2 \rfloor}$.

▶ **Theorem 12** ([17, 31]). *Let G be a graph, and let H be the underlying graph of the digraph δG. Then, $\chi(H) = \min\{n \mid \chi(G) \leq b(n)\}$.*

Using the fact that $b(n) \sim \frac{2^n}{\sqrt{\pi n/2}}$, Theorem 12 implies that the chromatic number of G is exponential in the chromatic number of H. Our goal in this section is to relate the chromatic number of G to other graph parameters of H, namely, the orthogonality dimension, the minrank of the complement, and the optimal length of an index code for the complement.

3.1 Orthogonality Dimension

For a field $\mathbb{F}$, an integer n, and a subspace U of $\mathbb{F}^n$, we denote by $U^\perp$ the subspace of $\mathbb{F}^n$ that consists of the vectors that are orthogonal to U over $\mathbb{F}$, i.e.,

$$U^\perp = \{w \in \mathbb{F}^n \mid \langle w, u\rangle = 0 \text{ for every } u \in U\}.$$

Consider the following family of graphs.

▶ **Definition 13.** *For a field $\mathbb{F}$ and an integer n, let $S_1(\mathbb{F}, n)$ denote the graph whose vertices are all the subspaces of $\mathbb{F}^n$, where two distinct subspaces U_1 and U_2 are adjacent if there exists a vector $w \in \mathbb{F}^n$ with $\langle w, w\rangle \neq 0$ that satisfies $w \in U_1 \cap U_2^\perp$ and, in addition, there exists a vector $w' \in \mathbb{F}^n$ with $\langle w', w'\rangle \neq 0$ that satisfies $w' \in U_2 \cap U_1^\perp$.*

In words, two subspaces of $\mathbb{F}^n$ are adjacent in the graph $S_1(\mathbb{F}, n)$ if each of them includes a non-self-orthogonal vector that is orthogonal to the entire other subspace. Note that for an infinite field $\mathbb{F}$ and for $n \geq 2$, the vertex set of $S_1(\mathbb{F}, n)$ is infinite.

We argue that the chromatic number of a graph G can be used to estimate the orthogonality dimension of the underlying graph H of its line digraph δG. First, recall that by Theorem 12, the chromatic number of H is logarithmic in $\chi(G)$. This implies, using Claim 8, that the orthogonality dimension of H over any field is at most logarithmic in $\chi(G)$. For a lower bound on the orthogonality dimension of H, we need the following lemma that involves the chromatic numbers of the graphs $S_1(\mathbb{F}, n)$.

▶ **Lemma 14.** *Let $\mathbb{F}$ be a field, let G be a graph, let H be the underlying graph of the digraph δG, and put $n = \overline{\xi}_{\mathbb{F}}(H)$. Then, $\chi(G) \leq \chi(S_1(\mathbb{F}, n))$.*

Proof. Put $G = (V_G, E_G)$ and $H = (V_H, E_H)$. The assumption $n = \overline{\xi}_{\mathbb{F}}(H)$ implies that there exists an n-dimensional orthogonal representation of H over $\mathbb{F}$, that is, an assignment of a vector $u_v \in \mathbb{F}^n$ with $\langle u_v, u_v\rangle \neq 0$ to each vertex $v \in V_H$, such that $\langle u_v, u_{v'}\rangle = 0$ whenever v and v' are adjacent in H. Recall that the vertices of H, just as the vertices of δG, are the ordered pairs (x, y) of adjacent vertices x, y in G.

For every vertex $y \in V_G$, let U_y denote the subspace spanned by the vectors of the given orthogonal representation that are associated with the vertices of H whose tail is y, namely,

$$U_y = \text{span}(\{u_v \mid v = (x, y) \text{ for some } x \in V_G\}).$$

Note that U_y is a subspace of $\mathbb{F}^n$, and thus a vertex of $S_1(\mathbb{F}, n)$.

Consider the function that maps every vertex $y \in V_G$ of G to the vertex U_y of $S_1(\mathbb{F}, n)$. We claim that this function forms a homomorphism from G to $S_1(\mathbb{F}, n)$. To see this, let $x, y \in V_G$ be adjacent vertices in G, and consider the vector $w = u_{(x,y)}$ assigned by the given orthogonal representation to the vertex (x, y) of H. By the definition of an orthogonal representation, it holds that $\langle w, w\rangle \neq 0$. Since (x, y) is a vertex of H whose tail is y, it follows that $w \in U_y$. Further, every vertex of H of the form (x', x) for some $x' \in V_G$ is adjacent in H to (x, y), hence it holds that $\langle u_{(x',x)}, w\rangle = 0$. Since the subspace U_x is spanned by those vectors $u_{(x',x)}$, we obtain that w is orthogonal to the entire subspace U_x. It thus follows

that the vector w satisfies $\langle w, w\rangle \neq 0$ and $w \in U_y \cap U_x^{\perp}$. By symmetry, there also exists a vector $w' \in \mathbb{F}^n$ satisfying $\langle w', w'\rangle \neq 0$ and $w' \in U_x \cap U_y^{\perp}$, hence the subspaces U_x and U_y are adjacent vertices in $S_1(\mathbb{F}, n)$. We conclude that the above function is a homomorphism from G to $S_1(\mathbb{F}, n)$, hence the chromatic numbers of these graphs satisfy $\chi(G) \leq \chi(S_1(\mathbb{F}, n))$, as required. ◀

In order to derive useful bounds from Lemma 14, we need upper bounds on the chromatic numbers of the graphs $S_1(\mathbb{F}, n)$. Every vertex of $S_1(\mathbb{F}, n)$ is a subspace of $\mathbb{F}^n$ and thus can be represented by a basis that generates it. For a finite field $\mathbb{F}$ of size q, the number of possible bases does not exceed q^{n^2}, which obviously yields that $\chi(S_1(\mathbb{F}, n)) \leq q^{n^2}$. While this simple bound suffices for proving our hardness results for the orthogonality dimension over finite fields, we note that the number of vertices in $S_1(\mathbb{F}, n)$ is in fact $q^{(1+o(1))\cdot n^2/4}$, where the $o(1)$ term tends to 0 when n tends to infinity.[1]

We conclude this discussion with the following theorem.

▶ **Theorem 15.** *Let $\mathbb{F}$ be a finite field of size q, let G be a graph, and let H be the underlying graph of the digraph δG. Then, it holds that*

$$\overline{\xi}_{\mathbb{F}}(H) \geq \sqrt{\log_q \chi(G)}.$$

Proof. Put $n = \overline{\xi}_{\mathbb{F}}(H)$, and apply Lemma 14 to obtain that $\chi(G) \leq \chi(S_1(\mathbb{F}, n)) \leq q^{n^2}$. By rearranging, the proof is completed. ◀

3.1.1 The Chromatic Number of $S_1(\mathbb{R}, n)$

For the real field $\mathbb{R}$ and for $n \geq 2$, the vertex set of the graph $S_1(\mathbb{R}, n)$ is infinite, and yet, its chromatic number is finite. To see this, let us firstly observe a simple upper bound of 2^{3^n}. To each vertex of $S_1(\mathbb{R}, n)$, i.e., a subspace U of $\mathbb{R}^n$, assign the subset of $\{0, \pm 1\}^n$ that consists of all the sign vectors of the vectors of U. This assignment forms a proper coloring of the graph, because for adjacent vertices U and V there exists a nonzero vector $w \in U$ that is orthogonal to V, hence the sign vector of w belongs to the set of sign vectors of U but does not belong to the one of V (because the inner product of two vectors with the same nonzero sign vector is positive). Since the number of subsets of $\{0, \pm 1\}^n$ is 2^{3^n}, it follows that $\chi(S_1(\mathbb{R}, n)) \leq 2^{3^n}$.

The above double-exponential bound is not sufficient for deriving NP-hardness of approximation results for the orthogonality dimension over $\mathbb{R}$ from the currently known NP-hardness results of the chromatic number. We therefore need the following lemma that provides an exponentially better bound which is suitable for our purposes. For a vector $w \in \mathbb{R}^n$, we use here the notation $\|w\| = \sqrt{\langle w, w\rangle}$ for the Euclidean norm of w.

▶ **Lemma 16.** *For every integer n, it holds that $\chi(S_1(\mathbb{R}, n)) \leq (2n+1)^{n^2}$.*

Proof. We define a coloring of the vertices of the graph $S_1(\mathbb{R}, n)$ as follows. For every vertex of $S_1(\mathbb{R}, n)$, i.e., a subspace U of $\mathbb{R}^n$, let $(u_1, \ldots, u_k)$ be an arbitrary orthonormal basis of U where $k \leq n$, and assign U to the color $c(U) = (u'_1, \ldots, u'_k)$ where u'_i is a vector obtained from u_i by rounding each of its values to a closest integer multiple of $\frac{1}{n}$. Note that for every $i \in [k]$, the vectors u_i and u'_i differ in every coordinate by no more than $\frac{1}{2n}$ in absolute value.

---

[1] To see this, observe that the number of k-dimensional subspaces of $\mathbb{F}^n$ is precisely $\prod_{i=0}^{k-1} \frac{q^n - q^i}{q^k - q^i}$ and that every term in this product lies in $[q^{n-k-1}, q^{n-k+1}]$. Hence, the total number of subspaces of $\mathbb{F}^n$ is at least $\sum_{k=0}^{n} q^{(n-k-1)k}$ and at most $\sum_{k=0}^{n} q^{(n-k+1)k}$. It follows that the number of subspaces of $\mathbb{F}^n$ is $q^{(1+o(1))\cdot n^2/4}$.

We claim that c is a proper coloring of $S_1(\mathbb{R}, n)$. To see this, let U and V be adjacent vertices in the graph. If $\dim(U) \neq \dim(V)$ then it clearly holds that $c(U) \neq c(V)$. So suppose that the dimensions of U and V are equal, and put $k = \dim(U) = \dim(V)$. Denote the orthonormal bases associated with U and V by $(u_1, \ldots, u_k)$ and $(v_1, \ldots, v_k)$ respectively, and let $c(U) = (u'_1, \ldots, u'_k)$ and $c(V) = (v'_1, \ldots, v'_k)$ be their colors. Our goal is to show that $c(U) \neq c(V)$.

Assume for the sake of contradiction that $c(U) = c(V)$, that is, $u'_i = v'_i$ for every $i \in [k]$. This implies that for every $i \in [k]$, the vectors u_i and v_i differ in each coordinate by no more than $\frac{1}{n}$ in absolute value, hence

$$\|u_i - v_i\| \leq \sqrt{n \cdot \frac{1}{n^2}} = \frac{1}{\sqrt{n}}. \tag{1}$$

Since U and V are adjacent in the graph $S_1(\mathbb{R}, n)$, by scaling, there exists a unit vector $u \in U \cap V^{\perp}$. Write $u = \sum_{i\in[k]} \alpha_i \cdot u_i$ for coefficients $\alpha_1, \ldots, \alpha_k \in \mathbb{R}$. Since the given basis of U is orthonormal, it follows that $\sum_{i\in[k]} \alpha_i^2 = \|u\|^2 = 1$. Now, consider the vector $v = \sum_{i\in[k]} \alpha_i \cdot v_i$, and observe that v is a unit vector that belongs to the subspace V. Observe further that

$$\begin{aligned} \|u - v\| &= \Big\| \sum_{i\in[k]} \alpha_i \cdot (u_i - v_i) \Big\| \leq \sum_{i\in[k]} |\alpha_i| \cdot \|u_i - v_i\| \\ &\leq \Big(\sum_{i\in[k]} \alpha_i^2 \Big)^{1/2} \cdot \Big(\sum_{i\in[k]} \|u_i - v_i\|^2 \Big)^{1/2} \leq 1, \end{aligned} \tag{2}$$

where the first inequality follows from the triangle inequality, the second from the Cauchy-Schwarz inequality, and the third from (1) using $k \leq n$. However, u and v are orthogonal unit vectors, and as such, the distance between them satisfies $\|u - v\| = \sqrt{2}$. This yields a contradiction to (2), hence $c(U) \neq c(V)$.

To complete the proof, we observe that the number of colors used by the proper coloring c does not exceed $(2n+1)^{n^2}$. Indeed, every color can be represented by an $n \times n$ matrix whose values are of the form $\frac{a}{n}$ for integers $-n \leq a \leq n$ (where the matrix associated with a subspace of dimension k consists of the rounded k column vectors concatenated with $n - k$ columns of zeros). Since the number of those matrices is bounded by $(2n+1)^{n^2}$, we are done. ◀

We derive the following theorem.

▶ **Theorem 17.** *There exists a constant $c > 0$, such that for every graph G with $\chi(G) \geq 3$, the underlying graph H of the digraph δG satisfies*

$$\overline{\xi}_{\mathbb{R}}(H) \geq c \cdot \sqrt{\frac{\log \chi(G)}{\log \log \chi(G)}}.$$

Proof. Put $n = \overline{\xi}_{\mathbb{R}}(H)$, and combine Lemma 14 with Lemma 16 to obtain that

$$\chi(G) \leq \chi(S_1(\mathbb{R}, n)) \leq (2n+1)^{n^2},$$

which yields the desired bound. ◀

A discussion on the clique numbers of the graphs $S_1(\mathbb{F}, n)$ can be found in the full version of the paper.

3.2 Minrank

As in the previous section, we start with a definition of a family of graphs.

▶ **Definition 18.** *For a field $\mathbb{F}$ and an integer n, let $S_2(\mathbb{F}, n)$ denote the graph whose vertices are all the pairs of subspaces of $\mathbb{F}^n$, where two distinct pairs (U_1, W_1) and (U_2, W_2) are adjacent if there exist two vectors $u, w \in \mathbb{F}^n$ with $\langle u, w\rangle \neq 0$ such that $u \in U_1 \cap W_2^{\perp}$ and $w \in W_1 \cap U_2^{\perp}$ and, in addition, there exist two vectors $u, w \in \mathbb{F}^n$ with $\langle u, w\rangle \neq 0$ such that $u \in U_2 \cap W_1^{\perp}$ and $w \in W_2 \cap U_1^{\perp}$.*

We next argue that the chromatic number of a graph G can be used to estimate the minrank of the complement of the underlying graph of its line digraph δG. This is established using the following lemma that involves the chromatic numbers of the graphs $S_2(\mathbb{F}, n)$. Its proof resembles that of Lemma 14 and can be found in the full version of the paper.

▶ **Lemma 19.** *Let $\mathbb{F}$ be a field, let G be a graph, let H be the underlying graph of the digraph δG, and put $n = \mathrm{minrk}_{\mathbb{F}}(\overline{H})$. Then, $\chi(G) \leq \chi(S_2(\mathbb{F}, n))$.*

We derive the following theorem.

▶ **Theorem 20.** *Let $\mathbb{F}$ be a finite field of size q, let G be a graph, and let H be the underlying graph of the digraph δG. Then, it holds that*

$$\mathrm{minrk}_{\mathbb{F}}(\overline{H}) \geq \sqrt{\tfrac{1}{2} \cdot \log_q \chi(G)}.$$

Proof. Put $n = \mathrm{minrk}_{\mathbb{F}}(\overline{H})$, and apply Lemma 19 to obtain that

$$\chi(G) \leq \chi(S_2(\mathbb{F}, n)) \leq q^{2n^2},$$

where the second inequality holds because the number of vertices in $S_2(\mathbb{F}, n)$ does not exceed q^{2n^2}. By rearranging, the proof is completed. ◀

3.2.1 The Chromatic Number of $S_2(\mathbb{R}, n)$

We next consider the problem of determining the chromatic numbers of the graphs $S_2(\mathbb{R}, n)$. The following theorem shows that these graphs cannot be properly colored using a finite number of colors, in contrast to the graphs $S_1(\mathbb{R}, n)$ addressed in Lemma 16. Its proof can be found in the full version of the paper.

▶ **Theorem 21.** *For every integer $n \geq 3$, it holds that $\chi(S_2(\mathbb{R}, n)) = \infty$.*

3.3 Index Coding

For the general (non-linear) index coding problem, we provide the following result (recall Definition 9). Its proof can be found in the full version of the paper.

▶ **Theorem 22.** *Let Σ be an alphabet of size at least 2, let G be a graph, and let H be the underlying graph of the digraph δG. If there exists an index code for $\overline{H}$ over Σ of length k, then $\chi(G) \leq 2^{|\Sigma|^k}$.*

4 Hardness Results

In this section, we prove our hardness results for the orthogonality dimension and for minrank. We also suggest a potential avenue for proving hardness results for the general index coding problem over a constant-size alphabet.

The starting point of our hardness proofs is the following theorem of Wrochna and Živný [34]. Recall that the function $b : \mathbb{N} \to \mathbb{N}$ is defined by $b(n) = \binom{n}{\lfloor n/2 \rfloor}$.

▶ **Theorem 23** ([34]). *For every integer $k \geq 4$, it is* NP*-hard to decide whether a given graph G satisfies $\chi(G) \leq k$ or $\chi(G) \geq b(k)$.*

Our hardness results for the orthogonality dimension and the minrank parameter over finite fields are given by the following theorem, which confirms Theorem 2.

▶ **Theorem 24.** *There exists a function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(k) = (1 - o(1)) \cdot \sqrt{b(k)}$ such that for every finite field $\mathbb{F}$ and for every sufficiently large integer k, the following holds.*

1. *It is* NP*-hard to decide whether a given graph G satisfies*

$$\overline{\xi}_{\mathbb{F}}(G) \leq k \quad \text{or} \quad \overline{\xi}_{\mathbb{F}}(G) \geq \frac{1}{\sqrt{\log |\mathbb{F}|}} \cdot f(k).$$

2. *It is* NP*-hard to decide whether a given graph G satisfies*

$$\mathrm{minrk}_{\mathbb{F}}(G) \leq k \quad \text{or} \quad \mathrm{minrk}_{\mathbb{F}}(G) \geq \frac{1}{\sqrt{2 \cdot \log |\mathbb{F}|}} \cdot f(k).$$

Proof. Fix a finite field $\mathbb{F}$ of size q. We start by proving the first item of the theorem. For an integer $k \geq 4$, consider the problem of deciding whether a given graph G satisfies

$$\chi(G) \leq b(k) \quad \text{or} \quad \chi(G) \geq b(b(k)),$$

whose NP-hardness follows from Theorem 23. To obtain our hardness result on the orthogonality dimension over $\mathbb{F}$, we reduce from this problem. Consider the reduction that given an input graph G produces and outputs the underlying graph H of the digraph δG. This reduction can clearly be implemented in polynomial time (in fact, in logarithmic space).

To prove the correctness of the reduction, we analyze the orthogonality dimension of H over $\mathbb{F}$. If G is a YES instance, that is, $\chi(G) \leq b(k)$, then by combining Claim 8 with Theorem 12, it follows that

$$\overline{\xi}_{\mathbb{F}}(H) \leq \chi(H) \leq k.$$

If G is a NO instance, that is, $\chi(G) \geq b(b(k))$, then by Theorem 15, it follows that

$$\overline{\xi}_{\mathbb{F}}(H) \geq \sqrt{\log_q \chi(G)} \geq \sqrt{\log_q b(b(k))} = \frac{1 - o(1)}{\sqrt{\log q}} \cdot \sqrt{b(k)},$$

where the $o(1)$ term tends to 0 when k tends to infinity. Note that we have used here the fact that $b(n) = \Theta(2^n/\sqrt{n})$. By letting k be any sufficiently large integer, the proof of the first item of the theorem is completed.

The proof of the second item of the theorem is similar. To avoid repetitions, we briefly mention the needed changes in the proof. First, to obtain a hardness result for the minrank parameter, the reduction has to output the complement $\overline{H}$ of the graph H rather than H itself. Second, in the analysis of the NO instances, one has to apply Theorem 20 instead of Theorem 15 to obtain that

$$\mathrm{minrk}_{\mathbb{F}}(\overline{H}) \geq \sqrt{\tfrac{1}{2} \cdot \log_q \chi(G)} \geq \sqrt{\tfrac{1}{2} \cdot \log_q b(b(k))} = \frac{1 - o(1)}{\sqrt{2 \cdot \log q}} \cdot \sqrt{b(k)}.$$

This completes the proof of the theorem. ◀

As an immediate corollary of Theorem 24, we obtain the following.

▶ **Corollary 25.** *For every finite field $\mathbb{F}$, the following holds.*
1. *It is* NP*-hard to approximate $\overline{\xi}_{\mathbb{F}}(G)$ for a given graph G to within any constant factor.*
2. *It is* NP*-hard to approximate* $\mathrm{minrk}_{\mathbb{F}}(G)$ *for a given graph G to within any constant factor.*

We next prove a hardness result for the orthogonality dimension over the reals, confirming Theorem 1.

▶ **Theorem 26.** *There exists a function $f : \mathbb{N} \to \mathbb{N}$ satisfying $f(k) = \Theta(\sqrt{b(k)/k})$ such that for every sufficiently large integer k, it is* NP*-hard to decide whether a given graph G satisfies*

$$\overline{\xi}_{\mathbb{R}}(G) \leq k \quad \text{or} \quad \overline{\xi}_{\mathbb{R}}(G) \geq f(k).$$

Proof. As in the proof of Theorem 24, for an integer $k \geq 4$, we reduce from the problem of deciding whether a given graph G satisfies

$$\chi(G) \leq b(k) \quad \text{or} \quad \chi(G) \geq b(b(k)),$$

whose NP-hardness follows from Theorem 23. Consider the polynomial-time reduction that given an input graph G produces and outputs the underlying graph H of the digraph δG.

To prove the correctness of the reduction, we analyze the orthogonality dimension of H over $\mathbb{R}$. If G is a YES instance, that is, $\chi(G) \leq b(k)$, then by combining Claim 8 with Theorem 12, it follows that

$$\overline{\xi}_{\mathbb{R}}(H) \leq \chi(H) \leq k.$$

If G is a NO instance, that is, $\chi(G) \geq b(b(k))$, then by Theorem 17 combined with the fact that $b(n) = \Theta(2^n/\sqrt{n})$, it follows that

$$\overline{\xi}_{\mathbb{R}}(H) \geq c \cdot \sqrt{\tfrac{\log b(b(k))}{\log\log b(b(k))}} = \Theta\Big(\sqrt{\tfrac{b(k)}{k}}\Big),$$

where c is an absolute positive constant. This completes the proof of the theorem. ◀

As an immediate corollary of Theorem 26, we obtain the following.

▶ **Corollary 27.** *It is* NP*-hard to approximate $\overline{\xi}_{\mathbb{R}}(G)$ for a given graph G to within any constant factor.*

We end this section with a statement that might be useful for proving NP-hardness results for the general index coding problem. Consider the following definition.

▶ **Definition 28.** *For an alphabet Σ and for two integers $k_1 < k_2$, let* Index-Coding$_{\Sigma}(k_1, k_2)$ *denote the problem of deciding whether the minimal length of an index code for a given graph G over Σ is at most k_1 or at least k_2.*

We prove the following result.

▶ **Theorem 29.** *Let Σ be an alphabet of size at least 2, and let k_1, k_2 be two integers. Then, there exists a polynomial-time reduction from the problem of deciding whether a given graph G satisfies $\chi(G) \leq b(k_1)$ or $\chi(G) \geq k_2$ to* Index-Coding$_{\Sigma}(k_1, \log_{|\Sigma|}\log k_2)$.

Proof. Consider the polynomial-time reduction that given an input graph G produces the underlying graph H of the digraph δG and outputs its complement $\overline{H}$. For correctness, suppose first that G is a YES instance, that is, $\chi(G) \leq b(k_1)$. Then, by combining Claim 8 with Theorem 12, it follows that $\text{minrk}_{\mathbb{F}_2}(\overline{H}) \leq \chi(H) \leq k_1$. By Proposition 10, it further follows that there exists a linear index code for $\overline{H}$ over $\mathbb{F}_2$ of length k_1. In particular, using $|\Sigma| \geq 2$, there exists an index code for $\overline{H}$ over the alphabet Σ of length k_1. Suppose next that G is a NO instance, that is, $\chi(G) \geq k_2$. By Theorem 22, it follows that the length of any index code for $\overline{H}$ over Σ is at least $\log_{|\Sigma|} \log k_2$, so we are done. ◀

Theorem 29 implies that in order to prove the NP-hardness of the general index coding problem over some finite alphabet Σ of size at least 2, it suffices to prove for some integer k that it is NP-hard to decide whether a given graph G satisfies $\chi(G) \leq b(k)$ or $\chi(G) > 2^{|\Sigma|^k}$.

References

1. Noga Alon. The Shannon capacity of a union. *Combinatorica*, 18(3):301–310, 1998.
2. 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 vesrion in FOCS'06.
3. Libor Barto, Jakub Bulín, Andrei A. Krokhin, and Jakub Opršal. Algebraic approach to promise constraint satisfaction. *J. ACM*, 68(4):28:1–28:66, 2021. Preliminary version in STOC'19.
4. Amey Bhangale. NP-hardness of coloring 2-colorable hypergraph with poly-logarithmically many colors. In *Proc. of the 45th International Colloquium on Automata, Languages, and Programming (ICALP'18)*, pages 15:1–15:11, 2018.
5. Joshua Brakensiek and Venkatesan Guruswami. New hardness results for graph and hypergraph colorings. In *Proc. of the 31st Conference on Computational Complexity (CCC'16)*, pages 14:1–14:27, 2016.
6. Eden Chlamtáč and Ishay Haviv. Linear index coding via semidefinite programming. *Combinatorics, Probability & Computing*, 23(2):223–247, 2014. Preliminary version in SODA'12.
7. Son Hoang Dau, Vitaly Skachek, and Yeow Meng Chee. Optimal index codes with near-extreme rates. *IEEE Trans. Inform. Theory*, 60(3):1515–1527, 2014. Preliminary version in ISIT'12.
8. Ronald de Wolf. Quantum Computing and Communication Complexity. PhD thesis, Universiteit van Amsterdam, 2001.
9. Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM J. Comput.*, 39(3):843–873, 2009. Preliminary version in STOC'06.
10. Irit Dinur and Igor Shinkar. On the conditional hardness of coloring a 4-colorable graph with super-constant number of colors. In *Proc. of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'10)*, pages 138–151, 2010.
11. Michael R. Garey and David S. Johnson. The complexity of near-optimal graph coloring. *J. ACM*, 23(1):43–49, 1976.
12. 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.
13. Venkatesan Guruswami and Sai Sandeep. d-To-1 hardness of coloring 3-colorable graphs with $O(1)$ colors. In *Proc. of the 47th International Colloquium on Automata, Languages, and Programming, (ICALP'20)*, pages 62:1–62:12, 2020.
14. Willem H. Haemers. On some problems of Lovász concerning the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(2):231–232, 1979.
15. Willem H. Haemers. An upper bound for the Shannon capacity of a graph. In László Lovász and Vera T. Sós, editors, *Algebraic Methods in Graph Theory*, volume 25/I of *Colloquia Mathematica Societatis János Bolyai*, pages 267–272. Bolyai Society and North-Holland, 1981.

16 Frank Harary and Robert Z. Norman. Some properties of line digraphs. *Rend. Circ. Mat. Palermo*, 9(2):161–168, 1960.

17 C. C. Harner and R. C. Entringer. On the arc-chromatic number of a digraph. *J. Comb. Theory, Ser. B*, 13(3):219–225, 1972.

18 Ishay Haviv. Approximating the orthogonality dimension of graphs and hypergraphs. In *Proc. of the 44th International Symposium on Mathematical Foundations of Computer Science (MFCS'19)*, pages 39:1–39:15, 2019.

19 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990.

20 Sangxia Huang. Improved hardness of approximating chromatic number. In *Proc. of the 16th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'13)*, pages 233–243, 2013.

21 Gil Kalai. Analogues for Sperner and Erdös-Ko-Rado theorems for subspaces of linear spaces. In Peter L. Hammer, editor, *Combinatorics 79*, volume 9 of *Annals of Discrete Math.*, page 135. Elsevier, 1980.

22 Richard M. Karp. Reducibility among combinatorial problems. In *Proc. of a Symposium on the Complexity of Computer Computations*, pages 85–103, 1972.

23 Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *J. ACM*, 64(1):4:1–4:23, 2017. Preliminary versions in FOCS'12 and STACS'14.

24 Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000. Preliminary version in ISTCS'93.

25 Subhash Khot. Improved inaproximability results for maxclique, chromatic number and approximate graph coloring. In *Proc. of the 42nd Symposium on Foundations of Computer Science (FOCS'01)*, pages 600–609, 2001.

26 Michael Langberg and Alexander Sprintson. On the hardness of approximating the network coding capacity. *IEEE Trans. Inform. Theory*, 57(2):1008–1014, 2011. Preliminary version in ISIT'08.

27 László Lovász. On the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(1):1–7, 1979.

28 László Lovász. *Graphs and Geometry*, volume 65. Colloquium Publications, 2019.

29 László Lovász, Michael Saks, and Alexander Schrijver. Orthogonal representations and connectivity of graphs. *Linear Algebra Appl.*, 114–115:439–454, 1989. Special Issue Dedicated to Alan J. Hoffman.

30 René Peeters. Orthogonal representations over finite fields and the chromatic number of graphs. *Combinatorica*, 16(3):417–431, 1996.

31 Svatopluk Poljak and Vojtech Rödl. On the arc-chromatic number of a digraph. *J. Comb. Theory, Ser. B*, 31(2):190–198, 1981.

32 Claude E. Shannon. The zero error capacity of a noisy channel. *Institute of Radio Engineers, Trans. Inform. Theory*, IT-2:8–19, 1956.

33 Saul Stahl. n-tuple colorings and associated graphs. *J. Comb. Theory, Ser. B*, 20(2):185–203, 1976.

34 Marcin Wrochna and Stanislav Živný. Improved hardness for H-colourings of G-colourable graphs. In *Proc. of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'20)*, pages 1426–1435, 2020.

35 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007. Preliminary version in STOC'06.

Extending Merge Resolution to a Family of QBF-Proof Systems

Sravanthi Chede
Indian Institute of Technology Ropar, Rupnagar, India

Anil Shukla
Indian Institute of Technology Ropar, Rupnagar, India

Abstract

Merge Resolution (MRes [4]) is a recently introduced proof system for false QBFs. Unlike other known QBF proof systems, it builds winning strategies for the universal player (countermodels) within the proofs as merge maps. Merge maps are deterministic branching programs in which isomorphism checking is efficient, as a result MRes is a polynomial time verifiable proof system.

In this paper, we introduce a family of proof systems MRes-$\mathcal{R}$ in which the information of countermodels are stored in any pre-fixed complete representation $\mathcal{R}$. Hence, corresponding to each possible complete representation $\mathcal{R}$, we have a sound and refutationally complete QBF-proof system in MRes-$\mathcal{R}$. To handle these arbitrary representations, we introduce consistency checking rules in MRes-$\mathcal{R}$ instead of the isomorphism checking in MRes. As a result these proof systems are not polynomial time verifiable (Non-P). Consequently, the paper shows that using merge maps is too restrictive and with a slight change in rules, it can be replaced with arbitrary representations leading to several interesting proof systems.

We relate these new systems with the implicit proof system from the algorithm in [8], which was designed to solve DQBFs (Dependency QBFs) using clause-strategy pairs like MRes. We use the OBDD (Ordered Binary Decision Diagrams) representation suggested in [8] and deduce that "Ordered" versions of the proof systems in MRes-$\mathcal{R}$ are indeed polynomial time verifiable.

On the lower bound side, we lift the lower bound result of regular MRes ([5]) by showing that the completion principle formulas (CR$_n$) from [17] which are shown to be hard for regular MRes in [5], are also hard for any regular proof system in MRes-$\mathcal{R}$. Thereby, the paper lifts the lower bound of regular MRes to an entire class of proof systems, which use various complete representations, including those undiscovered, instead of only merge maps. Thereby proving that the hardness of CR$_n$ formulas is intact even after changing the weak isomorphism checking in MRes to the stronger consistency checking in MRes-$\mathcal{R}$.

2012 ACM Subject Classification Theory of computation → Proof complexity

Keywords and phrases Proof complexity, QBFs, Merge Resolution, Simulation, Lower Bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.21

Acknowledgements We thank the anonymous reviewers who helped us substantially improve the motivation of this paper and also shorten the proofs. We also thank Gaurav Sood and Leroy Chew for important discussions, comments and suggestions regarding this paper.

1 Introduction

Proof complexity is a sub-branch of computational complexity in which the main focus is to understand the complexity of proving (refuting) theorems (contradictions) in various proof systems. Informally, a proof system is a polynomial time computable function which maps proofs to theorems. Several propositional proof systems like resolution [22], Cutting planes [12], and Frege [16] have been developed for proving (refuting) propositional formulas. The relative strength of these proof systems has been well studied [23]. Several proof systems which are not polynomial time verifiable (unless NP = co-NP) have also been well studied. For example, semantic resolution [18] and semantic cutting planes [15].


© Sravanthi Chede and Anil Shukla;
licensed under Creative Commons License CC-BY 4.0


Quantified Boolean formulas (QBFs) extend propositional logic by quantifying every variable by $\exists$ (there exists) or $\forall$ (for all). There are two major approaches for QBF-proof systems, namely, the CDCL-based (Conflict-Driven Clause Learning) and expansion-based systems. The basic systems in these approaches are Q-Res [19] and $\forall$Exp+Res [17] respectively. The Q-Res system was later extended to LD-Q-Res in [2] to allow a certain type of tautological clauses on universal variables in the proofs (which were always discarded in Q-Res) using merged literals. These merged literals have been shown to be interpreted as partial strategies rather than tautologies. These strategies were represented explicitly in [4] to form a new proof system called the MRes system. MRes (Merge Resolution) proof system [4] follows a different QBF solving approach. It builds partial strategies as "merge maps" at each line of the proof such that the strategy at the last line forms the countermodel for the input QBF. Before applying the refutation rules, MRes needs the strategies of the hypothesis to be isomorphic. As isomorphism checking is known to be efficient in merge maps, MRes is a polynomial time verifiable proof system.

In this paper, we extend MRes to a family of sound and refutationally complete QBF proof systems MRes-$\mathcal{R}$. We observe that the representation of strategies in the proofs as merge maps is not relevant for the soundness and completeness of the proof system. Strategies can be depicted by any complete representation (Section 2) and by slightly modifying the refutation rules to include arbitrary representations, the soundness and completeness of the proof system remains intact. To be precise, we change the isomorphism checking rule in MRes to "consistency" checking rule (Section 3.1) defined initially for Dependency Quantified Boolean Formulas (DQBFs, [21]) in [8]. This leads to the definition of a new proof system for each complete representation. All these new proof systems together form the family of proof systems denoted by MRes-$\mathcal{R}$. Since the consistency checking rules are computationally hard, the proof systems in MRes-$\mathcal{R}$ are not polynomial time verifiable (**Non-P proof systems**). In literature, many interesting Non-P QBF proof systems have been studied, for instance, semantic cutting planes for QBFs (SemCP+$\forall$red) [7] and QBF proof systems modulo NP [10].

This paper also studies in detail the strength and limitations of these new proof systems. In [4], the authors demonstrated how MRes allows a few forbidden resolution steps of LD-Q-Res. Similarly, we show that because of the introduction of such powerful consistency checking rules, proof systems in MRes-$\mathcal{R}$ also allow a few forbidden resolution steps of MRes (ref. Example 8). We also show a Lower bound on a restricted version of proof systems in MRes-$\mathcal{R}$ (regular MRes-$\mathcal{R}$). We explain our contributions in detail in the following section:

1.1 Our Contributions

1. **Introducing a new family of non-polynomial time verifiable proof systems MRes-$\mathcal{R}$ for QBFs:** As already stated, proof systems in MRes-$\mathcal{R}$ use consistency checking instead of isomorphism rules of MRes. Informally, an "isomorphism" check confirms whether two strategies are exactly the same or not. On the other hand, a "consistency" check confirms whether or not two strategies can give a non-contradicting output for every possible assignment of input variables. Precisely, an output of "$*$" (trivial/ don't care strategy) doesn't contradict with any output of the other strategy, while an output of "1" from one strategy and "0" from another is considered contradicting. MRes allows select operation (Section 2.1) on isomorphic strategies (i.e it only retains one of them). Whereas, proof systems in MRes-$\mathcal{R}$ allow union operation (Definition 2) on consistent strategies (i.e it retains both strategies and outputs the non-trivial assignment (if possible) from their outputs). For further explanations of the MRes and MRes-$\mathcal{R}$ proof systems, refer Sections 2.1 and 3 respectively.

For proving refutational completeness of the MRes-$\mathcal{R}$ family, we consider the MRes-$\mathcal{M}$ proof system $\in$ MRes-$\mathcal{R}$ which uses merge maps as the representation. We then prove that every valid rule of MRes is also valid in MRes-$\mathcal{M}$ (Theorem 6). We then show how to convert a MRes-$\mathcal{M}$ proof into a proof of any system $\mathcal{P} \in$ MRes-$\mathcal{R}$ (Claim 7). This conversion is non-efficient, but still guarantees the completeness of systems in MRes-$\mathcal{R}$.

The soundness proof of any $\mathcal{P}$ ($\in$ MRes-$\mathcal{R}$) proof system follows from proving that every line of the $\mathcal{P}$-refutation gives a partial falsifying strategy for the universal player. Hence at the last line, we prove that it gives a countermodel for the input QBF (Lemma 4).

In [4], the resolution steps in which strategies of universal variables left of the pivot are the same were shown to be forbidden in LD-Q-Res but allowed in MRes. We show in Example 8, resolution steps in which the strategy of universal variables left of the pivot are consistent but not isomorphic to be forbidden in MRes but allowed in MRes-$\mathcal{R}$.

2. **Relating MRes-$\mathcal{R}$ with the implicit proof system from [8]**: In [8], the authors introduce an algorithm to solve DQBFs, which works with clause-strategy pairs like the MRes system. They also give a representation called the OBDDs (Ordered Binary Decision Diagrams [8, Definition 3]) which can support the consistency check in polynomial time [13]. We observed that the implicit proof system from this algorithm is closely related to the MRes-$\mathcal{R}$ systems. More specifically, this algorithm outputs "ordered" MRes-$\mathcal{R}$ proofs using OBDDs as the representation (denoted by ordered MRes-$\mathcal{O}$). We also show in Proposition 12 that ordered MRes-$\mathcal{R}$ systems are polynomial time verifiable. For this, we first show how any MRes-$\mathcal{R}$ proof can be efficiently converted into a proof of MRes-$\mathcal{T}$ (the MRes-$\mathcal{R}$ system using a $\mathcal{T}$-representation defined in Section 4.1). Then, we give a method to convert efficiently an ordered MRes-$\mathcal{T}$ proof into an ordered MRes-$\mathcal{O}$ proof. Hence making the former system polynomial time verifiable. By transitivity from Theorem 9, it implies that all ordered systems in MRes-$\mathcal{R}$ are also polynomial time verifiable.
3. **Proving a lower bound for Regular MRes-$\mathcal{R}$:** A Lower bound for a proof system is a family of problems which are hard (exponential) to refute in that particular system. We establish one such lower bound for a restricted version of all proof systems in MRes-$\mathcal{R}$ (i.e regular MRes-$\mathcal{R}$ (Section 3.2)) with a family of QBFs called Completion Principle Formulas (CR_n). The CR_n formulas (Definition 13) were first introduced in [17], to show that level-ordered Q-Res cannot p-simulate the $\forall$Exp+Res proof system. It has been shown recently in [5], that CR_n formulas are even hard for regular MRes. In this paper, we lift this lower bound to all regular proof systems in MRes-$\mathcal{R}$ (Section 5).

   To establish the lower bound, we mostly follow the ideas of the lower bound proof of regular MRes from [5, Theorem 9]. In [5], the major part of the proof relied on the fact that MRes uses isomorphism, so in a resolution step, they could rule out the variables not in one hypothesis merge map as also not to be present in the other. However, this is not the case in MRes-$\mathcal{R}$. So we provide a new claim (Claim 16) with proof that even though MRes-$\mathcal{R}$ insists on consistency rather than isomorphism, the clauses in CR_n make it such that the above property holds. This result implies that the lower bound result in [5] for regular MRes is not because of the strictness of isomorphism checking in MRes or the usage of merge-maps, but it persists independent of the strategy representations.

Although these contributions are theoretical, they can be significant in practical DQBF/QBF solving in the near future: In contribution 2 above, we have shown that the ordered MRes-$\mathcal{R}$ proof systems (a subclass of MRes-$\mathcal{R}$) are closely related to the Davis-Putnam style algorithm designed in [8]. Therefore the lower bound established in contribution 3 for the regular MRes-$\mathcal{R}$ proof systems, also holds for the algorithm from [8]. Hence the theoretical results on our family of proof systems MRes-$\mathcal{R}$ is a significant step towards understanding the strengths

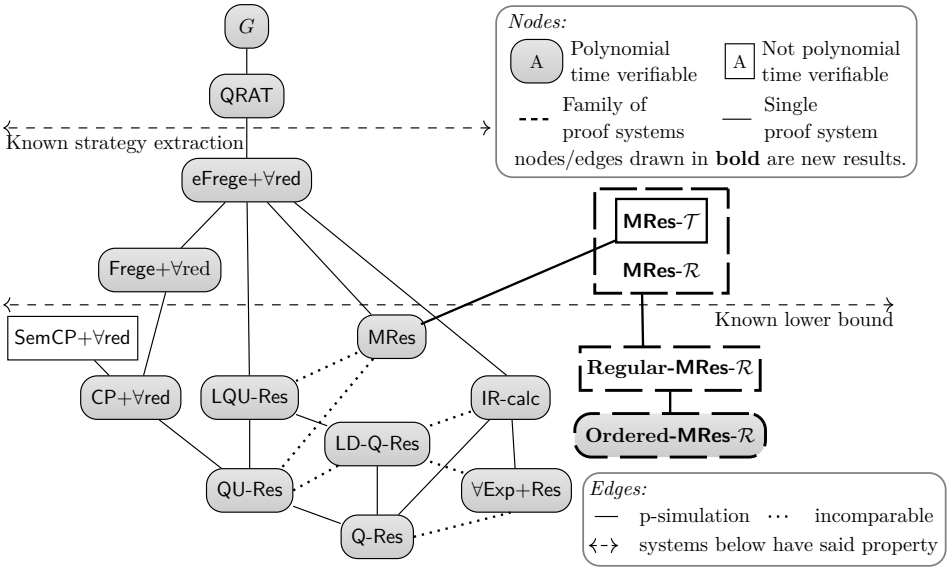


Figure 1 *Various QBF proof systems and p-simulations.* Regular MRes-$\mathcal{R}$ are below the "known lower bound" dashed line by Theorem 18. MRes-$\mathcal{T}$ p-simulates MRes by Proposition 10. Ordered MRes-$\mathcal{R}$ systems are polynomial time verifiable due to Proposition 12. For other known simulations refer[11, Fig.1], in-comparability results refer [20, 5, 6]

and limitations of the famous Davis-Putnam algorithm [14] in the DQBF/QBF domain. Today, the Davis-Putnam algorithm is widely used in various SAT-solvers thus understanding it for the QBF/DQBF domain is important for the field of Theoretical Computer Science. We sum up our contributions in Figure 1. The figure also shows current p-simulation order among QBF proof systems with the contributions of this paper being in bold.

2 Notations and Preliminaries

For a Boolean variable x, its literals can be x and $\overline{x}$. A clause C is a disjunction of literals and a conjunctive normal form (CNF) formula F is a conjunction of clauses. We denote the empty clause by $\bot$. $vars(C)$ is a set of all variables in C and width$(C) = |vars(C)|$.

Given a language $L \subseteq \{0,1\}^*$ and a string $x \in L$, a **proof system** f for L is an inference system, which is capable of showing that x is indeed in L. To do this, f derives a sequence of lines inferred via a set of predefined rules in a step by step fashion either from the hypothesis (i.e x) or from previously inferred lines. This sequence of lines are called an f-proof of the fact that $x \in L$. A proof system f for L is complete iff for every $x \in L$ we have a corresponding f-proof for x. A proof system f for L is sound iff the existence of an f-proof for x implies that $x \in L$. By definition, a proof system must be sound and complete for the language L. In addition, it must be polynomial time computable (verifiable). That is, given a sequence of lines, it must be check-able whether every line is derived by a valid rule of the system in time polynomial w.r.t the size of the input sequence, in which case, it is said to be a valid f-proof. A non-polynomial time verifiable proof system (**Non-P proof system**) for a language L is a proof system but without needing to be polynomial time verifiable.

Quantified Boolean formulas: QBFs are an extension of the propositional Boolean formulas where each variable is quantified with one of $\{\exists, \forall\}$, with their general semantic meaning of existential and universal quantifier respectively. In this paper, we assume that

QBFs are in closed prenex form with CNF matrix i.e., we consider the form $Q_1X_1...Q_kX_k.\phi(X_1 \cup \cdots \cup X_k)$, where X_i are pairwise disjoint sets of variables; $Q_i \in \{\exists, \forall\}$ and $Q_i \neq Q_{i+1}$, and the matrix ϕ is in CNF form. We denote QBFs as $\mathcal{F} := Q.\phi$ in this paper, where Q is the quantifier prefix. If $x \in X_i$ then we denote $Q(x)$ to be equal to Q_i. For a variable x if $Q(x) = \exists$ (resp. $Q(x) = \forall$), we call x an existential (resp. universal) variable. If a variable x is in the set X_i, any $y \in X_j$ where $j < i$ ($j > i$), we say that y occurs to the left (right) of x in the quantifier prefix and write $y \leq_Q x$ ($y \geq_Q x$). The set of existential variables to the left of a universal variable u will be denoted by $L_Q(u)$ in this paper.

Let $C \in \phi$ and $Q(u) = \forall$, then the "falsifying u-literal" is defined to be 0 if $u \in C$, and 1 if $\overline{u} \in C$ and "$*$" if $u \notin vars(C)$. Also, the existential subclause of C is the clause formed by only the existential literals from C. If S is any set of variables, a complete assignment of S will be an assignment which assigns all variables in S to either 1 or 0. Similarly, a partial assignment is an assignment which assigns a subset of variables in S to either 1 or 0 and the rest are denoted as having an assignment of "$*$". We denote $\langle S \rangle$ and $\langle\langle S \rangle\rangle$ as the sets of all possible complete assignments and partial assignments of S respectively. For some clause C and some partial assignment of variables α, $C[\alpha]$ is defined as replacing all occurrences of variables in C with the assignments from α (where defined) and simplifying it. For example if $C = (x_1 \vee \neg x_2 \vee x_3)$ and α be $\{x_1 = 0, x_2 = 1, x_4 = 0\}$. Then, $C[\alpha]$ is (x_3).

For a QBF $Q.\phi$, a **strategy** of universal player is a decision function that returns the assignment to all universal variables of Q, where the decision for each u depends only on the variables in $L_Q(u)$. If H^u is the strategy for the universal variable u then, $vars(H^u)$ is the subset of variables from $L_Q(u)$ which are actually used in building the strategy H^u. **Winning strategy** for the universal player is a strategy which for every possible assignment of existential variables, gives an assignment to all universal variables such that it falsifies the QBF. A QBF is false iff there exists a winning strategy for the universal player [1]. We say that a QBF proof system f admits **strategy extraction** if for any given valid f-proof of a false QBF $\mathcal{F}$, one can compute a winning strategy for the universal player in the time polynomial to the size of the f-proof. As said earlier, strategies are basically decision functions. For the portrayal of the same, many representations can be used like truth tables, directed acyclic graphs (DAGs), merge maps, etc. A **complete representation** is the one in which every possible finite decision function can be represented.

Resolution [22] is the most studied redundancy rule in both SAT and QBF worlds, we define the same as: $\frac{(C \vee x) \quad (D \vee \overline{x})}{(C \vee D)}$, where C, D are clauses and x is the pivot variable. We denote this step as "$res(C \vee x, D \vee \overline{x}, x)$" throughout the paper.
Next, we define a few QBF proof systems that we require in this paper.

Q-Res [19] is one of a basic QBF proof system. It is an extension of the resolution proof system for QBFs. It allows the resolution rule (defined above) with the pivot variable being existential. For dealing with the universal variables, it defines a "universal reduction" rule which allows dropping of a universal variable u from a clause C, provided no existential variable $x \in C$ appears to the right of u.

LD-Q-Res [2] was developed as an improvement to the Q-Res system. The main "Long distance rule" used in this system is:

$$\frac{(C_1 \cup U_1 \cup \{x\}) \quad (C_2 \cup U_2 \cup \{\overline{x}\})}{(C_1 \cup C_2 \cup U)},$$

where x is the $\exists$ pivot, $U_1 \& U_2$ are literals of $\forall$ variables ($\geq_Q x$) common in both hypothesis such that if $u_1 (\in U_1)$, $u_2 (\in U_2)$ are literals of the same variable z then either $u_1 = \overline{u_2}$ or $u_1/u_2 = z^*$. If $l_1 (\in C_1)$, $l_2 (\in C_2)$ & $var(l_1, l_2) = z$ then $l_1 = l_2 \neq z^*$. Then, $U = \{u^* | u \in U_1\}$.

2.1 Merge Resolution (MRes)

MRes is a proof system for false QBFs introduced in [4], we describe MRes briefly in this section. For a false QBF $Q.\phi$, an MRes refutation will be a sequence of lines of the form $L_i = (C_i, \{M_i^u\})$; where C_i is a clause of only existential-literals and $\{M_i^u\}$ is the set of merge maps of each universal variable $u \in Q$. The **merge map** M_i^u is a decision branching graph with definite strategies $\{0, 1, *\}$ at the leaves nodes ("$*$" is used when no strategy for u exists till that line) and the intermediate nodes branch on some existential variable (say x) $\in L_Q(u)$. That is, if $L_i = res(L_a, L_b, x)$ for some $a, b < i$, then M_i^u will get connected to M_a^u with an edge label of $\overline{x}$ and to M_b^u with an edge label of x.

An important property used in MRes rules is **Isomorphism:** Two merge maps are isomorphic iff there exists a bijection mapping from the nodes of one to that of another.

Operations needed for MRes rules are defined as follows: **Select** operation on two isomorphic merge maps, outputs one of them. Or if one of them is trivial (i.e "$*$"), outputs the other. **Merge**(M_a^u, M_b^u, n, x) operation, defined when $a, b < n$, returns a new merge map where the root node is connected to M_a^u with $\overline{x}$ and M_b^u with x.
Now we define the MRes proof system:

For a false QBF $Q.\phi$, the MRes proof $\Pi := L_1, L_2, ..., L_m$ where every line L_i is derived using either an "Axiom" step or a "Resolution" step. In the axiom step, C_i will be the existential subclause of some $C \in \phi$ and every M_i^u will be a leaf node with the falsifying u-literal of C. In the resolution step, C_i is obtained from $res(C_a, C_b, x)$ where x is an $\exists$-variable and $a, b < i$, also each M_i^u must either be select(M_a^u, M_b^u) or if $x <_Q u$ then can be merge(M_a^u, M_b^u, i, x). Π is a refutation iff $C_m = \bot$.

G_Π is the derivation graph corresponding to Π with edges directed from the hypothesis to the resolvent (i.e from the axioms to the final line). For some given line L, Π_L is defined as the sub-derivation of Π deriving the line L.

3 MRes-$\mathcal{R}$: A new family of proof systems for false QBFs

Inspired from MRes, we define a family MRes-$\mathcal{R}$ where every proof system $\mathcal{P}$ ($\in$ MRes-$\mathcal{R}$) has its own complete representation to represent the strategies. We use the idea of consistency checking in MRes-$\mathcal{R}$ from [8], which uses the same for DQBFs. For simplicity, we use the same notations from [8] whenever possible. We begin by defining some important notations and operations needed before formally defining the MRes-$\mathcal{R}$ systems.

3.1 Important notations used in MRes-$\mathcal{R}$

To begin, let us define what consistency means for any two assignments of a set of variables. Then we will extend it for two strategies.

▶ **Definition 1** ([8]). *Let X be any set of variables and $\varepsilon, \delta \in \langle\langle X \rangle\rangle$. We say that ε and δ are consistent, denoted by $\varepsilon \simeq \delta$, if for every $x \in X$ for which $\varepsilon(x), \delta(x) \neq *$ we have $\varepsilon(x) = \delta(x)$.*

Let H_u and H'_u be individual strategy functions for the universal variable u, we say that H_u and H'_u are **consistent** (written $H_u \simeq H'_u$) when $H_u(\varepsilon) \simeq H'_u(\varepsilon)$ for each $\varepsilon \in \langle L_Q(u) \rangle$. In other words H_u and H'_u are consistent, if the u-assignments given by $H_u(\varepsilon)$ and $H'_u(\varepsilon)$ are consistent for every possible $L_Q(u)$-assignment ε.

By a change in notation, we can see (partial) assignments as both functions and sets of literals, i.e. an assignment ε corresponds to the set of literals it satisfies. For example, $\{x_1, x_2, \overline{x_3}, \overline{x_4}\}$ represents an assignment which sets 1 to the variables x_1 and x_2 and 0 to x_3 and x_4. In this notation as sets of literals, a union ($\cup$) of assignments ε, δ is defined when $\varepsilon \simeq \delta$ and it is equal to $\varepsilon \cup \delta$.

We now define a **union operation** ("$\circ$") on two consistent strategies H_u and H'_u.

▶ **Definition 2** ([8]). *Given two consistent strategies H_u and H'_u (i.e., $H_u \simeq H'_u$), we define the union strategy H''_u of H_u and H'_u, denoted by $H''_u = H_u \circ H'_u$, as:*

$H''_u(\varepsilon) = H_u(\varepsilon) \cup H'_u(\varepsilon)$ *for each* $\varepsilon \in \langle L_Q(u) \rangle$.

For example, if H_u and H'_u are defined as below, then $H''_u = H_u \circ H'_u$ will be:

$$H_u = \begin{cases} 1 & : & x \\ * & : & \overline{x} \end{cases} \qquad H'_u = \begin{cases} * & : & x \\ 0 & : & \overline{x} \end{cases} \quad ; \quad H''_u = \begin{cases} 1 \cup * = 1 & : & x \\ * \cup 0 = 0 & : & \overline{x} \end{cases}$$

We now define a **if-else operation** ("$\bowtie$") on any two strategies H_u and H'_u.

▶ **Definition 3** ([8]). *Given any two strategies H_u and H'_u and an existential variable x, we define the if-else operation of H_u and H'_u on x to give the strategy H''_u, denoted by $H''_u = H_u \overset{x}{\bowtie} H'_u$, for every $\varepsilon \in \langle L_Q(u) \rangle$ as follows:*

$$H''_u(\varepsilon) = \begin{cases} H_u(\varepsilon) & : & \varepsilon(x) = 1 \\ H'_u(\varepsilon) & : & \varepsilon(x) = 0 \end{cases}$$

For example, if H_u and H'_u be defined as below, then $H''_u = H_u \overset{x}{\bowtie} H'_u$ will be:

$$H_u = \begin{cases} 1 & : & y \\ * & : & \overline{y} \end{cases} \qquad H'_u = 0 \quad ; \quad H''_u = \begin{cases} 1 & : & xy \\ * & : & x\overline{y} \\ 0 & : & \overline{x} \end{cases}$$

3.2 Definition of MRes-$\mathcal{R}$

Let $\Phi = Q.\phi$ be a QBF with existential variables X and universal variables U. A MRes-$\mathcal{R}$ derivation of L_m from Φ is a sequence $\pi = L_1, ..., L_m$ of lines where each $L_i = (C_i, \{H_i^u : u \in U\})$ in which at least one of the following holds for all $i \in [m]$:

a. **Axiom**. There exists a clause $C \in \phi$ such that C_i is the existential subclause of C, and for each $u \in U$, H_i^u is the strategy function mapping u to the falsifying u-literal for C or,

b. **Resolution**. There exist integers $a, b < i$ and an existential pivot $x \in X$ such that $C_i = res(C_a, C_b, x)$ and for each $u \in U$:

- **(i)** if $x <_Q u$, then $H_i^u = H_b^u \overset{x}{\bowtie} H_a^u$
- **(ii)** else if $x >_Q u$, then $H_i^u = H_a^u \circ H_b^u$.

π is a refutation of Φ iff $C_m = \bot$. Size of π is the number of lines i.e $|\pi| = m$.

Regular MRes-$\mathcal{R}$: Let S be a subset of existential variables X of a QBF $\mathcal{F}$. We say that a $\mathcal{P}$-refutation π of $\mathcal{F}$ ($\mathcal{P} \in$ MRes-$\mathcal{R}$) is S-regular if for every $x \in S$, there is no leaf to root path in G_π that uses x as pivot more than once. A X-regular proof is simply a regular proof.

Ordered MRes-$\mathcal{R}$: Let X be the set of all existential variables of a false QBF $\mathcal{F}$ and $\leq_X$ be a fixed ordering of variables in X. We say that a $\mathcal{P}$-refutation π of $\mathcal{F}$ ($\mathcal{P} \in$ MRes-$\mathcal{R}$) is ordered if it is regular and for each leaf-to-root path in G_π, the pivots follow $\leq_X$.

3.3 Soundness of MRes-$\mathcal{R}$

Soundness of MRes-$\mathcal{R}$ is proved by the next lemma, it follows closely to that of MRes in [4].

▶ **Lemma 4.** *Let $\pi = L_1, \ldots, L_m$ be a $\mathcal{P}$ refutation ($\mathcal{P} \in$ MRes-$\mathcal{R}$) of QBF Φ. Then, strategy functions $\{H_m^u : u \in U\}$ in the conclusion line L_m will form a countermodel for Φ.*

Proof. Given $\pi := L_1, \ldots, L_m$ be a $\mathcal{P}$-refutation of QBF $\Phi = Q.\phi$. Each L_i=$(C_i, \{H_i^u : u \in U\})$ and X, U are sets of all existential and universal variables in Q respectively. For $i \in [m]$,

- let $\alpha_i := \{\bar{l} : l \in C_i\}$ be the smallest assignment falsifying Ci ,
- let $A_i := \{\alpha \in \langle X \rangle : C_i \cap \alpha = \emptyset\}$ be all complete assignments to X consistent with α_i,
- for each $\alpha \in A_i$, let $l_i^u(\alpha) := H_i^u(\alpha)$ and $H_i(\alpha) := \{l_i^u(\alpha) : u \in U\} \setminus \{*\}$.

Induction statement: By induction on $i \in [m]$, we show, for each $\alpha \in A_i$, that the restriction of ϕ by $\alpha \cup H_i(\alpha)$ contains the empty clause.

Proof: For the base case $i = 1$, let $\alpha \in A_1$. As L_1 is introduced as an axiom, there exists a clause $C \in \phi$ such that C_1 is the existential subclause of C, and each H_1^u is the function mapping u to the falsifying u-literal for C. Hence, for each $u \in U$, $l_1^u(\alpha)$ is the falsifying u-literal for C, so $C[\alpha \cup H_1(\alpha)] = \emptyset$.

For the inductive step, let $i \geq 2$ and let $\alpha \in A_i$. The case where L_i is introduced as an axiom is identical to the base case, so we assume that L_i was derived by resolution. Then there exist integers $a, b < i$ and an existential pivot $x \in X$ such that $C_i = res(C_a, C_b, x)$. Suppose that $\overline{x} \in \alpha$ (a similar argument holds when $x \in \alpha$), each $u \in U$ has to satisfy either:

(i) $x <_Q u$ and $H_i^u = H_b^u \overset{x}{\bowtie} H_a^u$: In which case, $l_i^u(\alpha) = l_a^u(\alpha)$.

(ii) $x >_Q u$ and $H_i^u = H_a^u \circ H_b^u$: In which case, $l_i^u(\alpha) = \{l_a^u(\alpha) \cup l_b^u(\alpha)\}$.

It follows that $l_i^u \neq l_a^u$ only if $l_a^u = *$, and hence $H_a(\alpha) \subseteq H_i(\alpha)$. Since $C_a \setminus \{x\} \subseteq C_i$, we have $\alpha \in A_a$, so the restriction of ϕ by $\alpha \cup H_i(\alpha)$ contains the empty clause by the inductive hypothesis that $\alpha \cup H_a(\alpha)$ contains the empty clause. ⌟

Since α_m is the empty assignment, we have $A_m = \langle X \rangle$. We therefore prove the lemma at the final step $i = m$, as we show that $\{H_m^u : u \in U\}$ is a countermodel for Φ. ◀

3.4 Completeness of MRes-$\mathcal{R}$

Completeness of MRes-$\mathcal{R}$ is proved by the following Theorem 6 and Claim 7. Proof of Claim 7 is given in Section A of the appendix. We will need the following remark from the paper introducing MRes ([4]).

▶ Remark 5. [4, Proposition 10] Any two isomorphic merge maps compute the same function.

▶ **Theorem 6.** *MRes-$\mathcal{M}$ (MRes-$\mathcal{R}$ using merge maps as representation) p-simulates MRes.*

Proof. Given a QBF Φ and its MRes-proof $\pi = L_1, ..., L_m$, where every $L_i = \{C_i, \{M_i^u : u \in U\}\}$. We build an MRes-$\mathcal{M}$ proof $\Pi = L'_1, ..., L'_m$ for Φ, where each $L'_i = \{C'_i, \{H_i^u : u \in U\}\}$. For every line L_i in π starting from $i = 1$ to m, if L_i is an axiom step then directly $C'_i = C_i$ and $H_i^u = M_i^u$ for all $u \in U$. Otherwise, if L_i is an resolution step i.e for some $a, b < i$, $C_i = res(C_a, C_b, x)$; then set $C'_i = C_i$ and for each $u \in U$ if $x <_Q u$ then set $H_i^u = H_b^u \overset{x}{\bowtie} H_a^u$ else set $H_i^u = H_a^u \circ H_b^u$. These are sound steps as resolution in MRes can be either:

(i) $x >_Q u$ and $M_i = \text{select}(M_a^u, M_b^u)$; in this case we set $H_i^u = H_a^u \circ H_b^u$ which holds given the Remark 5 and that isomorphism ⇒ consistency.

(ii) $x <_Q u$ and $M_i = \text{merge}(M_a^u, M_b^u, i, x)$; in this case we set $H_i^u = H_b^u \overset{x}{\bowtie} H_a^u$ which is same as the merge function of MRes.

(iii) $x <_Q u$ and $M_i = \text{select}(M_a^u, M_b^u)$; in this case we set $H_i^u = H_b^u \overset{x}{\bowtie} H_a^u$ which is allowed as MRes did the isomorphism test on M_a^u and M_b^u, but we do not need it for $\bowtie$ in MRes-$\mathcal{R}$.

In case-(iii) above it remains to note that adding a $\bowtie$ to two isomorphic maps or when one of them is $*$, doesn't add any new strategy: it just dilutes the strategy represented by the corresponding merge map. That is, we are adding an if-else condition where both

Table 1 $\mathcal{P}$-refutation ($\mathcal{P} \in$ MRes-$\mathcal{R}$) of the false QBF in Example 8.

Line	Rule	C_i	H_i^u
L_1	axiom	$\{y, x\}$	0
L_2	axiom	$\{y, \overline{x}\}$	$*$
L_3	$res(L_1, L_2, x)$	$\{y\}$	$H_2^u \overset{x}{\bowtie} H_1^u$
L_4	axiom	$\{\overline{y}, x\}$	$*$
L_5	axiom	$\{\overline{y}, \overline{x}\}$	1
L_6	$res(L_4, L_5, x)$	$\{\overline{y}\}$	$H_5^u \overset{x}{\bowtie} H_4^u$
L_7	$res(L_3, L_6, y)$	$\{\}$	$H_3^u \circ H_6^u$

the outcomes are same or one of them is $*$. Hence doesn't affect future consistency checks which may arise in the proof (For further clarity, an instance is provided as Example 19 in Section A of the appendix. However it is not needed for the proof). Finally, the constructed Π is a valid MRes-$\mathcal{M}$ proof of Φ. ◀

▷ Claim 7. Every MRes-$\mathcal{M}$-proof can be transformed into an MRes-$\mathcal{R}$-proof for any representation R in exponential time.

These guarantee the completeness of proof systems in MRes-$\mathcal{R}$ as MRes is complete and any MRes-proof can be transformed into a MRes-$\mathcal{M}$-proof (by Theorem 6) which in-turn can be transformed as any MRes-$\mathcal{R}$-proof (by Claim 7).

Next, we present an example (Example 8) of MRes-$\mathcal{R}$ allowing few resolution steps which are forbidden in MRes. Such examples may be useful for the separation results between the proof systems in MRes-$\mathcal{R}$ and the existing MRes proof system.

▶ **Example 8.** Consider any proof system $\mathcal{P}$ in MRes-$\mathcal{R}$ which uses some complete R representation for strategies. Table 1 is a $\mathcal{P}$-refutation of the false QBF:

$\exists x \forall u \exists y\ (y \vee x \vee u) \wedge (y \vee \overline{x}) \wedge (\overline{y} \vee x) \wedge (\overline{y} \vee \overline{x} \vee \overline{u})$

The strategies H_3^u and H_6^u in function format are as follows:

$$H_3^u = \begin{cases} 0 & : \quad x = 0 \\ * & : \quad x = 1 \end{cases} \qquad H_6^u = \begin{cases} * & : \quad x = 0 \\ 1 & : \quad x = 1 \end{cases}$$

One can see that these strategies are consistent (but not isomorphic), hence the resolution of L_3, L_6 on y is allowed in the $\mathcal{P}$-refutation. But the corresponding resolution would be blocked in MRes since the corresponding merge maps M_3^u, M_6^u will not be isomorphic.

4 MRes-$\mathcal{T}$ proof system

In this section, we will define a particular proof system MRes-$\mathcal{T}$ from the family of MRes-$\mathcal{R}$ proof systems. The importance of this system is that any $\mathcal{P}$-refutation ($\mathcal{P} \in$ MRes-$\mathcal{R}$) can be efficiently converted into an MRes-$\mathcal{T}$-refutation. That is, all proof systems in MRes-$\mathcal{R}$ can be p-simulated by this MRes-$\mathcal{T}$ system. Later in this section, we will discuss how this system relates to the implicit proof system from the algorithm defined in [8].

Table 2 Truth table for # operator (It assumes inputs to be consistent).

A	B	A # B
1	1	1
0	0	0
$*$	0/1	0/1
0/1	$*$	0/1
$*$	$*$	$*$

4.1 Definition of MRes-$\mathcal{T}$

For a false QBF $\mathcal{F}$, the sequence of lines $\pi := (C_1, T_1), \ldots, (C_m, T_m)$ is an MRes-$\mathcal{T}$ refutation if $C_m = \bot$ and each T_i is built based on the derivation of C_i from parents C_j, C_k as follows:

$$T_i := \begin{cases} \text{Axiom node as in MRes} & \text{Axiom step of MRes-}\mathcal{R} \\ \text{Merge node over } T_j, T_k \text{ as in MRes} & \text{If-else step (`}C_j \overset{x}{\bowtie} C_k\text{') of MRes-}\mathcal{R} \\ \text{\# node (defined below) on } T_j, T_k & \text{Union step (`}C_j \circ C_k\text{') of MRes-}\mathcal{R} \end{cases}$$

The **# node** is defined assuming both its inputs are consistent, and it outputs the result of a union operation on them; Precisely, it's truth table is shown in Table 2. Note that $A = 1, B = 0$ and vice-versa cannot happen in a valid MRes-$\mathcal{R}$ proof owing to the definition of union("$\circ$"). Therefore, the corresponding rows are omitted from the truth table in Table 2. An illustrative example of an MRes-$\mathcal{T}$-proof, is given as Example 20 in Section B of the appendix.

Observe that the proposed T-representation is complete. That is, any valid finite function can be represented by a T-graph. This follows since, merge maps are a subset of T-graphs (i.e without # nodes) which are just branching programs, but known to be complete for all valid functions. Since T-representations are complete, it implies MRes-$\mathcal{T}$ $\in$ MRes-$\mathcal{R}$.

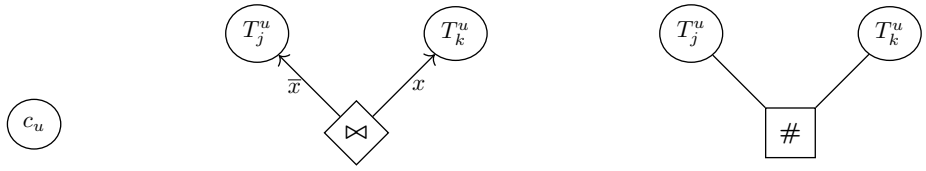
4.2 Conversion of MRes-$\mathcal{R}$ proofs into MRes-$\mathcal{T}$ proofs

In this section, we show how to convert a $\mathcal{P}$-proof π ($\mathcal{P} \in$ MRes-$\mathcal{R}$) into a MRes-$\mathcal{T}$ proof π'. Let $\pi = (C_1, R_1), \ldots, (C_m, R_m)$ be a $\mathcal{P}$ proof of a QBF $\mathcal{F}$. We show how to convert π into a MRes-$\mathcal{T}$ proof $\pi' = (C_1, T_1), \ldots, (C_m, T_m)$ of the same QBF $\mathcal{F}$. Note that here T_i is not the representation of R_i, but T_i is capturing how R_i has been constructed from some hypothesis R_j, R_k with $j, k < i$ using rules from Section 3.2. For this we do not need to interpret R_i's, but we can extract the required information from the clauses C_j, C_k and C_i of π (illustration of the same is given in Figure 2).

▶ **Theorem 9.** *Any $\mathcal{P}$-proof ($\mathcal{P} \in$ MRes-$\mathcal{R}$) can be converted efficiently into an MRes-$\mathcal{T}$ proof.*

Proof. For a false QBF $\mathcal{F}$, proofs of proof systems belonging to MRes-$\mathcal{R}$ can have arbitrary representations for the strategies computed. However, the rules allowed to construct a strategy R_i using any strategies R_j and R_k (where $j, k < i$) are fixed. They must follow the rules mentioned in Section 3.2. MRes-$\mathcal{T}$ proof π' captures these rules only.

To be precise, given a $\mathcal{P}$-proof π of $\mathcal{F}$ where $\pi = (C_1, P_1), ..., (C_m, P_m)$, we construct MRes-$\mathcal{T}$-proof π' as follows: From the clause part of the proof π i.e $C_1, ..., C_m$ (in this sequence) based on what step is being followed (axiom, or resolution where pivot is on left, or resolution where pivot is on right), we build the corresponding T-graphs as shown in



(a) Axiom.

(b) if-else node.
(res when x is left of u in prefix)

(c) # node (i.e union node).
(res when x is right of u in prefix)

Figure 2 Rules to construct T-graphs. In Figure 2a, c_u is the falsifying strategy of u for the axiom clause C_i. In Figure 2b, $C_i = res(C_j, C_k, x)$ and x is left of u in prefix i.e $T_i^u = T_k^u \overset{x}{\bowtie} T_j^u$. In Figure 2c, $C_i = res(C_j, C_k, x)$ and x is right of u in prefix i.e $T_i^u = T_j^u \circ T_k^u$.

Figure 2. After following this procedure for all lines in π, the sequence of lines so formed i.e $\pi' = (C_1, T_1), \ldots, (C_m, T_m)$ is a MRes-$\mathcal{T}$ proof as the clauses $C_1, ..., C_m$ are the same as in the original MRes-$\mathcal{R}$ proof hence we know that C_m is definitely $\bot$ and that $T_1, ..., T_m$ are built using the same rules as used when building the $\mathcal{P}$-proof π. ◀

Observe that due to Theorem 9, MRes-$\mathcal{T}$ p-simulates any MRes-$\mathcal{R}$ proof system, and therefore, it also p-simulates the MRes-$\mathcal{M}$ ($\in$ MRes-$\mathcal{R}$) proof system, which is known to simulate the MRes proof system (by Theorem 6). Thus we have the following:

▶ **Proposition 10.** *MRes-$\mathcal{T}$ p-simulates MRes.*

4.3 MRes-$\mathcal{T}$ versus Implicit proof system in [8]

The authors in [8] give an algorithm to work with DQBFs and a representation for strategies (OBDDs) to make the consistency check in the algorithm efficient. In this section we discuss how the implicit proof system from this algorithm relates to our newly defined proof systems.

The algorithm in [8] is designed to eliminate pivots in any fixed order by taking all possible resolvents at every stage. Hence, one can clearly see that the algorithm works implicitly on the ordered MRes-$\mathcal{R}$ system which uses OBDDs for the representation (denoted as MRes-$\mathcal{O}$). As the consistency check and union operation on OBDDs are shown to be efficient ([8, 13]), it makes the corresponding ordered MRes-$\mathcal{O}$ systems polynomial time verifiable.

We note that the "ordered"-$\mathcal{T}$ representation is much similar to an OBDD with an extra # node. We can efficiently convert an "ordered"-$\mathcal{T}$ strategy into an OBDD representation as follows: At every # node recursively from leaf-to-root, perform the union operation on the hypothesis strategies assuming them to be OBDDs and then replace the # node with the resultant OBDD-representation. This will end finally with a complete OBDD representation of the initial strategy represented by the ordered-$\mathcal{T}$-representation (An example of this conversion is shown in Section C of the appendix). When clubbed with ordered MRes-$\mathcal{O}$ being polynomial time verifiable, this implies the following proposition.

▶ **Proposition 11.** *Ordered MRes-$\mathcal{T}$ is polynomial time verifiable proof system.*

Proposition 11 along with the algorithm in Theorem 9 provided for conversion of any MRes-$\mathcal{R}$ proof into MRes-$\mathcal{T}$, deduces the following:

▶ **Proposition 12.** *Ordered MRes-$\mathcal{R}$ is a family of polynomial time verifiable proof systems.*

Observe that regular MRes-$\mathcal{T}$ cannot be guaranteed to be polynomial time verifiable. This is because, a regular MRes-$\mathcal{T}$ would need an FBDD (Free BDD) to use as a representation for strategies and according to [13], FBDDs cannot guarantee polynomial time verifiability.

Therefore, polynomial time verification, even with the usage of OBDDs, stops at ordered proofs itself, which are a restriction of regular proofs which in-turn are a restricted version of general proofs. Refer Figure 1 for the complete picture of the above discussed proof systems.

5 Lower Bound for Regular MRes-$\mathcal{R}$

▶ **Definition 13** (Completion Principle Formulas (CR_n) [17]).

$$CR_n = \exists x_{1,1}...\exists x_{1,n},...,\exists x_{n,1}...\exists x_{n,n},\ \forall z,\ \underset{i\in[n]}{\exists} a_i,\ \underset{j\in[n]}{\exists} b_j \left(\bigwedge_{i,j\in[n]} (A_{ij} \wedge B_{ij})\right) \wedge L_A \wedge L_B$$

where, $A_{ij} = x_{ij} \vee z \vee a_i$ $\quad B_{ij} = \overline{x_{ij}} \vee \overline{z} \vee b_j$

$L_A = \overline{a_1} \vee \cdots \vee \overline{a_n}$ $\quad L_B = \overline{b_1} \vee \cdots \vee \overline{b_n}$

We define the sets $A := \{a_1, a_2, ..., a_n\}$ *and* $B := \{b_1, b_2, ..., b_n\}$ *for ease of usage.*

In this section, we prove that the CR_n formulas are hard to refute in regular proof systems of MRes-$\mathcal{R}$. The lower bound result follows from a stronger result that we prove below in Theorem 14. We use the ideas from [5] and try to maintain the same notations wherever possible for simplicity. The proof setup is depicted in Figure 3. The basic idea of the proof is: As every clause in CR_n has a variable from the set $A \cup B$, but the refutation should derive a $\bot$ at the final line; there must be a "section" of the proof (see shaded region S' in Figure 3) which only has X variables in all its clauses. This section also includes the final line. The set of clauses at the "border" (see the bold line S in Figure 3) of this section of the proof are shown to be wide (in terms of number of literals) in Lemma 15. Using this and the argument that the conjunction of clauses in S itself forms a false CNF formula, we show in Theorem 14 that the number of clauses in S is exponential in n. This directly implies the lower bound.

To establish the width bound, we note that the pivots which are used while deriving clauses in S are variables from $A \cup B$ and that they are all to the right of z. Meaning that the corresponding resolutions must all be union steps i.e the incoming strategies must be consistent (not isomorphic as is the case in MRes). This especially makes it difficult to directly lift the lower bound proof of MRes from [5]. However we overcome this issue in Claim 16 by arguing how L_A, L_B are the only clauses with trivial strategies and how any other clause which resolves with these will mask this trivial-ness with its own definitive strategy. Further, by analysing what axiom clauses cannot be used in deriving the clauses in S, we show that many variables cannot be resolved before these lines. Hence, these variables will still be

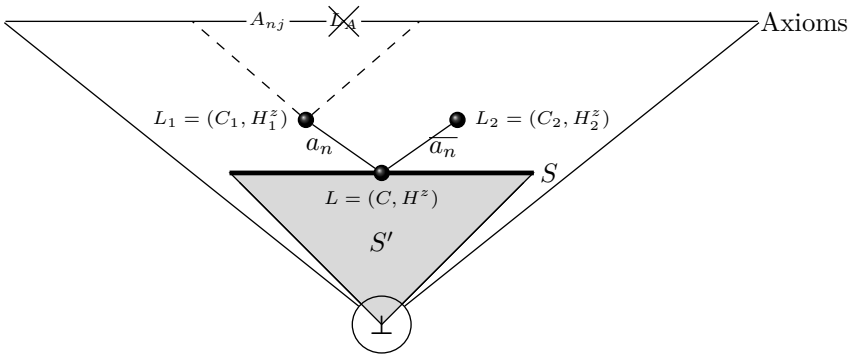


Figure 3 *Lower bound proof illustration.* Given a CR_n formula and it's $\mathcal{P}$-proof Π ($\mathcal{P} \in$ MRes-$\mathcal{R}$), this figure shows the graph G_Π. Claim 16 proves $x_{ij} \notin var(H_2^z)$ for $i \in [n-1], j \in [n]$. Claim 17 shows $|vars(C_2)| \geq n-1$. Lemma 15 shows $|vars(C)| \geq n-2$. Theorem 14 proves that $|S| \geq 2^{n-2}$.

present in the clause $\in S$, making it wide. We now clearly state the Theorem, Lemma and Claims explained above and give proofs for those with vital changes from corresponding proofs in [5]. The remaining detailed proofs are provided in Section D of the appendix.

▶ **Theorem 14.** *Every $(A \cup B)$-regular refutation of CR_n in any proof system belonging to MRes-$\mathcal{R}$ has size $2^{\Omega(n)}$.*

For $\mathcal{P} \in$ MRes-$\mathcal{R}$, let Π be a $\mathcal{P}$-refutation of CR_n (for $n > 2$). Let the set of lines S, S' be defined as follows:

S': This set consists of all the lines $L = (C, H^z)$ from Π such that $vars(C) \cap \{A \cup B\} = \emptyset$ and there exists a path from L to $\bot$ in G_Π consisting of lines only from S'.

S: This set contains all the lines $L \in S'$ such that $L = Res(L_1, L_2, v)$ where $L_1, L_2 \notin S'$. Observe that the pivot variable v must belong to $\{A \cup B\}$.

▶ **Lemma 15** ([5])**.** *For all $L = (C, H^z) \in S$, width$(C) \geq n - 2$.*

Proof. Observe that L is not an axiom as all axioms of CR_n have a variable from $A \cup B$ and so they cannot belong to S. So, let $L = res(L_1, L_2, v)$ where $L_1, L_2 \notin S'$. Since two lines not belonging in S' resolve to make the resultant $\in S'$, the pivot (i.e v) should be from $A \cup B$. Assume $v \in A$, a similar argument can be made when $v \in B$. Without loss of generality, assume that $v = a_n$[1]; and $a_n \in C_1$ and $\overline{a_n} \in C_2$.

Since Π is $(A \cup B)$-regular, a_n does not occur as a pivot in the sub-derivation Π_{L_1}. It implies that the axiom clause L_A cannot be used in deriving L_1, because otherwise C_1 will have both a_n & $\overline{a_n}$ making it a tautology. That implies, axioms with other positive literals a_i's cannot be used in Π_{L_1} as the negated literals $\overline{a_i}$'s are only available in L_A which in-turn cannot be used in Π_{L_1}. Positive literals of a_i's only $\in A_{ij}$ for all $j \in [n]$. Hence, axioms A_{ij} for $i \in [n-1], j \in [n]$ also cannot be used in deriving the line L_1. We know x_{ij} only occur in A_{ij}; so H_1^z has no x_{ij} variable for $i \in [n-1], j \in [n]$. Also, H_1^z is not a trivial strategy as some A_{nj} for $j \in [n]$ has been used because $a_n \in C_1$. Fix this j for the rest of the proof.

Since the pivot a_n at the resolution step obtaining line L is to the right of z, by the rules of MRes-$\mathcal{R}$, H_1^z and H_2^z are consistent. In Claim 16, we prove that even though MRes-$\mathcal{R}$ only insists on consistency, it still holds that for each $i \in [n-1]$, and each $j \in [n]$, $x_{ij} \notin \text{var}(H_2^z)$. Using this result we prove in Claim 17, that C_2 will have at least $n-1$ variables (including $\overline{a_n}$). Therefore, at least $n-2$ variables from C_2 belong in C. ◀

▷ Claim 16. For $i \in [n-1]$, and each $j \in [n]$, $x_{ij} \notin \text{var}(H_2^z)$.

Proof. At the point of use of this claim in the proof of Lemma 15, we definitely know that for $i \in [n-1]$ & $j \in [n]$; $x_{ij} \notin H_1^z$. That is, if f_1 is the function representing the strategy H_1^z, then for any assignment σ of x_{nj}'s and $i \in [n-1], j \in [n]$, it implies that:

$$f_1(\sigma, x_{ij} = 0) = f_1(\sigma, x_{ij} = 1) \tag{1}$$

Let f_2 be the function representing the strategy H_2^z. Since a_n is to the right of z, we know that H_1^z and H_2^z are consistent, i.e for any assignment σ' (an extension of σ) and for $i \in [n-1], j \in [n]$, it implies that:

$$f_2(\sigma', x_{ij} = 0) \simeq f_1(\sigma', x_{ij} = 0) \tag{2}$$

[1] Note that a_n is used only for ease in dividing the set A into partitions. Nowhere in the proof we use the fact that a_n is the last variable in A.

and,

$$f_2(\sigma', x_{ij} = 1) \simeq f_1(\sigma', x_{ij} = 1). \tag{3}$$

Only remaining question is if $f_2(\sigma', x_{ij} = 0) = f_2(\sigma', x_{ij} = 1)$? Observe that if this equality holds, then f_2 will be independent of x_{ij}'s, which implies that $x_{ij} \notin H_2^z$ for $i \in [n-1], j \in [n]$. Now, we are heading towards proving the equality holds. Note that if none of the terms in Equation 2 and Equation 3 give a "$*$" for any assignment of X, the equality in question definitely holds. So, now we prove that none of them can give a "$*$" for any given assignment.

The only axiom clauses of CR_n with trivial strategies are L_A, L_B and these axioms only contain variables of $A \cup B$, which are all to the right of z. Hence if any other clause is to be resolved with these clauses, the pivot has to be in $A \cup B$ i.e. a union step needs to be performed. At this point the trivial-ness of L_A (or L_B) is masked and does not show up in the final strategy of the resultant line; this is because union of any strategy with a trivial strategy will be the strategy itself. The only case by which a "$*$" can be in the resulting strategy is if L_A is resolved with L_B, which can clearly not happen as they have no common variable. Since C_1, C_2 are definitely not the axiom clauses L_A (or L_B), using the above argument it is simply not possible for the functions f_1 (or f_2) to output a "$*$" for any input assignment provided. This means the equality in question above holds; meaning that H_2^z also doesn't depend on x_{ij}'s when $i \in [n-1], j \in [n]$ i.e $x_{ij} \notin vars(H_2^z)$. ◁

▷ Claim 17 ([5]). Either for all $i \in [n-1]$, C_2 has a variable of the form x_{i*}, or for all $j \in [n]$, C_2 has a variable of the form x_{*j}.

From the above discussions and due to Theorem 14, we have the following:

▶ **Theorem 18.** *Every MRes-$\mathcal{R}$-regular refutation of CR_n has size $2^{\Omega(n)}$.*

6 Conclusion and Future work

This paper extends MRes proof system into a family of non-P proof systems MRes-$\mathcal{R}$ and provides a motivation example of forbidden steps of MRes being allowed in MRes-$\mathcal{R}$. This paper also deduces that "ordered" versions of proof systems in MRes-$\mathcal{R}$ are polynomial time verifiable and gives a lower bound for "regular" versions of proof systems in MRes-$\mathcal{R}$. Still several open problems remain in the scope of this paper. We point some of them as follows:

The relative strength of proof systems in MRes-$\mathcal{R}$ and MRes is still unclear. Since proof systems in MRes-$\mathcal{R}$ use strong consistency checking rules as compared to the isomorphism rule in MRes, we believe that there exists a family of QBFs which are easy for proof systems in MRes-$\mathcal{R}$ but hard for MRes.

Another direction is to establish a lower bound for proof systems in MRes-$\mathcal{R}$. It is open whether KBKF-lq formulas [3] (shown to be hard for the MRes proof system in [5]), are hard or easy for proof systems in MRes-$\mathcal{R}$. Note that, by slightly modifying the formula to KBKF-lq-split [20] it has been shown to be easy for MRes and hence making them easy for MRes-$\mathcal{R}$ as well.

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

2 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, August 2012. doi:10.1007/s10703-012-0152-6.

3 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *Theory and Applications of Satisfiability Testing – SAT 2014*, volume 8561 of *LNCS*, pages 154–169. Springer, 2014. doi:10.1007/978-3-319-09284-3_12.

4 Olaf Beyersdorff, Joshua Blinkhorn, and Meena Mahajan. Building strategies into QBF proofs. *J. Autom. Reason.*, 65(1):125–154, 2021. doi:10.1007/s10817-020-09560-1.

5 Olaf Beyersdorff, Joshua Blinkhorn, Meena Mahajan, Tomás Peitl, and Gaurav Sood. Hard QBFs for merge resolution. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 182 of *LIPIcs*, pages 12:1–12:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.12.

6 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. Proof complexity of resolution-based QBF calculi. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 76–89. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.76.

7 Olaf Beyersdorff, Leroy Chew, Meena Mahajan, and Anil Shukla. Understanding cutting planes for QBFs. *Inf. Comput.*, 262:141–161, 2018. doi:10.1016/j.ic.2018.08.002.

8 Joshua Blinkhorn, Tomás Peitl, and Friedrich Slivovsky. Davis and Putnam meet Henkin: Solving DQBF with resolution. In *Theory and Applications of Satisfiability Testing – SAT 2021*, volume 12831 of *LNCS*, pages 30–46. Springer, 2021. doi:10.1007/978-3-030-80223-3_4.

9 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

10 Leroy Chew. Hardness and optimality in QBF proof systems modulo NP. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021 – 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2021. doi:10.1007/978-3-030-80223-3_8.

11 Leroy Chew and Friedrich Slivovsky. Towards uniform certification in QBF. 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 22:1–22:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.22.

12 William J. Cook, Collette R. Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. doi:10.1016/0166-218X(87)90039-4.

13 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi:10.1613/jair.989.

14 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960. doi:10.1145/321033.321034.

15 Yuval Filmus, Pavel Hrubes, and Massimo Lauria. Semantic versus syntactic cutting planes. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS*, volume 47 of *LIPIcs*, pages 35:1–35:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.35.

16 Gottlob Frege. Begriffsschrift. In Jean Van Heijenoort, editor, *From Frege to Gödel*, pages 1–83. Cambridge: Harvard University Press, 1967. translated from Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache der reinen Denkens, Halle 1879.

17 Mikolás Janota and João Marques-Silva. Expansion-based QBF solving versus Q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015. doi:10.1016/j.tcs.2015.01.048.

18 Stasys Jukna. Exponential lower bounds for semantic resolution. In *Proof Complexity and Feasible Arithmetics, Proceedings of a DIMACS Workshop*, volume 39, pages 163–172. DIMACS/AMS, 1996. doi:10.1090/dimacs/039/10.

19 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Information and Computation*, 117(1):12–18, 1995. doi:10.1006/inco.1995.1025.

20 Meena Mahajan and Gaurav Sood. QBF merge resolution is powerful but unnatural. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.SAT.2022.22.

21 G. Peterson, J. Reif, and S. Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(7):957–992, 2001. doi:10.1016/S0898-1221(00)00333-3.

22 John Alan Robinson. Theorem-proving on the computer. *J. ACM*, 10(2):163–174, 1963. doi:10.1145/321160.321166.

23 Nathan Segerlind. The complexity of propositional proofs. *Bull. Symb. Log.*, 13(4):417–481, 2007. doi:10.2178/bsl/1203350879.

A Missing proof and example from Section 3.4

This example considers the situation corresponding to the case-(iii) of Theorem 6. That is, two isomorphic merge maps can be combined with an if-else and the resulting strategy will still output the same as input merge maps. Or when one of the input merge map being $*$, makes the resulting strategy diluted in the sense that for half the assignments it gives a $*$ and for others the same as the non-trivial input merge map.

▶ **Example 19.** Let $M_1^u = M_2^u = 1$ be leaf nodes in MRes proof. It implies that corresponding $H_1^u = 1$ and $H_2^u = 1$ in MRes-$\mathcal{R}$ proof. Now say MRes performs a resolution on pivot variable x which is to the left of u, resulting in $M_3^u = select(M_1^u, M_2^u)$. Whereas the corresponding MRes-$\mathcal{R}$ rule needs to be a $H_3^u = H_1^u \overset{x}{\bowtie} H_2^u$ from case(iii) (ref. Theorem 6). That is, H_3^u in function form would be defined as follows:

$$H_3^u = \begin{cases} 1 & : \quad x \\ 1 & : \quad \overline{x} \end{cases}$$

Notice how this is just a diluted way of writing the strategy $H_3^u = 1$. Hence when in the next line of MRes if a $M_4^u = 1$ which is isomorphic to M_3^u is encountered; the corresponding $H_4^u = 1$ in MRes-$\mathcal{R}$ will still remain to be consistent with H_3^u (though they might seem to be structurally different).
In the same example if $M_2^u = *$ (i.e. trivial), the strategy H_3^u would have been:

$$H_3^u = \begin{cases} 1 & : \quad x \\ * & : \quad \overline{x} \end{cases}$$

Notice how this is another way of diluting the strategy and is still consistent with $H_4^u = 1$.

▷ Claim 7. Every MRes-$\mathcal{M}$-proof can be transformed into an MRes-$\mathcal{R}$-proof for any representation R in exponential time.

Proof. Given a QBF Φ and its MRes-$\mathcal{M}$-proof $\pi = L_1, ..., L_m$, where every line $L_i = \{C_i, \{M_i^u : u \in U\}\}$. We intend to build an MRes-$\mathcal{R}$-proof $\Pi = L'_1, ..., L'_m$ for Φ, where each $L'_i = \{C'_i, \{H_i^u : u \in U\}\}$.

Table 3 A MRes-$\mathcal{T}$ refutation of the false QBF in Example 20.

Line	Rule	C_i	T_i^u	Type of node
L_1	axiom	$\{x, \overline{y}, a\}$	1	Leaf
L_2	axiom	$\{\overline{x}, \overline{y}, a\}$	*	Leaf
L_3	$res(L_1, L_2, x)$	$\{\overline{y}, a\}$	$T_2^u \overset{x}{\bowtie} T_1^u$	if-else
L_4	axiom	$\{\overline{x}, \overline{y}, \overline{a}\}$	1	Leaf
L_5	axiom	$\{x, \overline{y}, \overline{a}\}$	*	Leaf
L_6	$res(L_5, L_4, x)$	$\{\overline{y}, \overline{a}\}$	$T_4^u \overset{x}{\bowtie} T_5^u$	if-else
L_7	$res(L_3, L_6, a)$	$\{\overline{y}\}$	$T_3^u \circ T_6^u$	#
L_8	axiom	$\{\overline{x}, y, b\}$	1	Leaf
L_9	axiom	$\{x, y, b\}$	0	Leaf
L_{10}	$res(L_9, L_8, x)$	$\{y, b\}$	$T_8^u \overset{x}{\bowtie} T_9^u$	if-else
L_{11}	axiom	$\{y, \overline{b}\}$	*	Leaf
L_{12}	$res(L_{10}, L_{11}, b)$	$\{y\}$	$T_{10}^u \circ T_{11}^u$	#
L_{13}	$res(L_{12}, L_7, y)$	$\{\}$	$T_7^u \overset{y}{\bowtie} T_{12}^u$	if-else

For every line L_i in π, we keep the clause part intact while we convert the merge maps into plain functions. Further as R is a complete representation, these functions should have a corresponding representation in R; we extensively search for the same. This search terminates at some point owing to R being a complete representation. (This is the place where we used the property that R is a complete representation). The result is an MRes-$\mathcal{R}$-proof for Φ. This process is not polynomial in time but regardless still proves completeness for the family of proof systems MRes-$\mathcal{R}$. ◁

B Missing example from Section 4

▶ **Example 20.** Let $\Phi := \exists x, y, \forall u, \exists a, b\ (x, \overline{y}, \overline{u}, a) \wedge (\overline{x}, \overline{y}, a) \wedge (\overline{x}, \overline{y}, \overline{u}, \overline{a}) \wedge (x, \overline{y}, \overline{a}) \wedge (\overline{x}, y, \overline{u}, b) \wedge (x, y, u, b) \wedge (y, \overline{b})$. The MRes-$\mathcal{T}$ proof of Φ is shown below in Table 3:

The final T-graph of winning strategy for the only universal variable u from Example 20 is shown in Figure 4. One can see that this graph is a hybrid structure of both branching programs and circuits. Since it has both "branching" nodes ($\bowtie$ nodes) and "circuit" nodes (# nodes).

C Missing example from Section 4.3

In [8], the authors describe a way to use 2-valued OBDD functions to represent 3-valued strategy functions (including the don't care ($*$)). They represent each strategy function H_u as a pair of 2 functions $(H_u^\top, H_u^\bot)$ which can then represented by 2 OBDDs. This pair is simply defined as follows:

$$H_u^\top(\varepsilon) = \begin{cases} 1 & : \quad if H_u(\varepsilon) = 1 \\ 0 & : \quad otherwise \end{cases} \qquad H_u^\bot(\varepsilon) = \begin{cases} 1 & : \quad H_u(\varepsilon) = 0 \\ 0 & : \quad otherwise \end{cases}$$

In [8, Proposition 1], the authors also give the following formulas to perform union and if-else operation on OBDDs efficiently:

1. $(G_u \circ H_u)^\top = G_u^\top \vee H_u^\top$; $(G_u \circ H_u)^\bot = G_u^\bot \vee H_u^\bot$
   The algorithm for "$\vee$" on OBDDs (called bounded disjunction) is very well-known, it was introduced in [9]. (We will not discuss it here, rather will directly use it in Example 21).

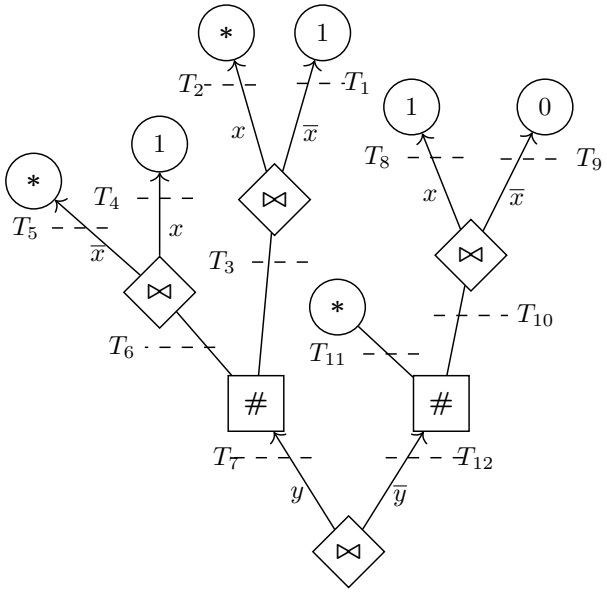


Figure 4 T^u_{13} graph for Example 20.

2. $(G_u \overset{x}{\bowtie} H_u)^\top = G_u^\top \overset{x}{\bowtie} H_u^\top$; $(G_u \circ H_u)^\perp = G_u^\perp \overset{x}{\bowtie} H_u^\perp$

Next, we will see how to convert a $\mathcal{T}$-represented strategy into an OBDD-represented strategy. For this we use the graph of T^u_{13} from previously shown Example 20, the graph of the same is already shown in Figure 4 for ease of cross-checking.

▶ **Example 21.** For the input $\mathcal{T}$-strategy, we use T^u_{13} from Example 20 (shown in Figure 4). As explained in Section 4.3, we follow recursively from leaf-to-root in this graph, converting it into OBDD pairs and applying $\circ$ and $\bowtie$ operations (as stated above) wherever applicable. Note that in OBDDs, we fix that the left edge of non-leaf node represents the positive edge and the right one is the negative edge (for ease in drawing). The step-by-step illustration of building OBDD-pairs is shown in Figure 5.

This completes the conversion of $\mathcal{T}$-graph to OBDD-pairs. To cross-check let us compute the resulting function from $(T^\top_{13}, T^\perp_{13})$. The process to extract the strategy (H_u) back from the OBDD-pair $(H_u^\top, H_u^\perp)$:

$$H_u(\varepsilon) = \begin{cases} 1 & : \quad if H_u^\top(\varepsilon) = 1 \\ 0 & : \quad if H_u^\perp(\varepsilon) = 1 \\ * & : \quad otherwise \end{cases}$$

So the truth-table of the resultant function (say T'_{13}) from $(T^\top_{13}, T^\perp_{13})$ is shown in Table 4. One can cross check that this is exactly the function computed by T^u_{13} from Figure 4.

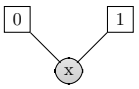
D Missing proofs from Section 5

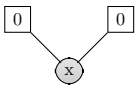
▶ **Theorem 14.** *Every $(A \cup B)$-regular refutation of CR_n in any proof system belonging to MRes-$\mathcal{R}$ has size $2^{\Omega(n)}$.*

Proof. For $\mathcal{P} \in$ MRes-$\mathcal{R}$, let Π be a $\mathcal{P}$-refutation of CR_n (for $n > 2$). Let the set of lines S, S' be defined as follows:

$T_1^\top = 1$, $T_1^\perp = 0$

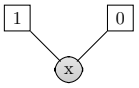
$T_2^\top = 0$, $T_2^\perp = 0$

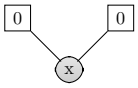
$T_3^\top =$ 

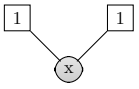
, $T_3^\perp =$ 

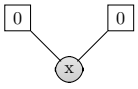
$T_4^\top = 1$, $T_4^\perp = 0$

$T_5^\top = 0$, $T_5^\perp = 0$

$T_6^\top =$ 

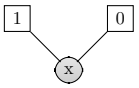
, $T_6^\perp =$ 

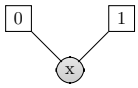
$T_7^\top =$ 

, $T_7^\perp =$ 

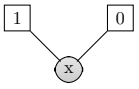
$T_8^\top = 1$, $T_8^\perp = 0$

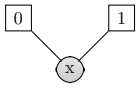
$T_9^\top = 0$, $T_9^\perp = 1$

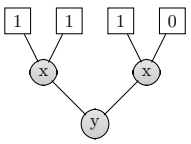
$T_{10}^\top =$ 

, $T_{10}^\perp =$ 

$T_{11}^\top = 0$, $T_{11}^\perp = 0$

$T_{12}^\top =$ 

, $T_{12}^\perp =$ 

$T_{13}^\top =$ 

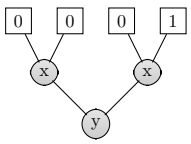
, $T_{13}^\perp =$ 

Figure 5 Building OBDD-pairs of strategy T_{13}^u from Example 20 (shown in Figure 4).

Table 4 Output of the strategy from OBDD-pairs of Figure 5.

x	y	T'_{13}
0	0	0
0	1	1
1	0	1
1	1	1

S': This set consists of all the lines $L = (C, H^z)$ from Π such that $vars(C) \cap \{A \cup B\} = \emptyset$ and there exists a path from L to $\bot$ in G_Π consisting of lines only from S'.

S: This set contains all the lines $L \in S'$ such that $L = Res(L_1, L_2, v)$ where $L_1, L_2 \notin S'$. Observe that the pivot variable v must belong to $\{A \cup B\}$.

Let $F = \bigwedge_{(C,H^z)\in S} C$. Note that F is a false CNF formula because there exists a sub-derivation $\widehat{\Pi} = \{C | \exists L = (C, H^z) \in S'\}$ which derives a $\bot$ given F. The variables in F are only x_{ij}'s where $i, j \in [n]$, therefore it consists of n^2 variables. In Lemma 15 we prove that each clause in F has width $\geq n-2$. That is each clause can be falsified by setting atleast $n-2$ variables to 0. Hence the number of complete assignments of X that can falsify a clause $\in F$ will be at most $2^{n^2-(n-2)}$. Since F is a false CNF formula, all assignments to X should falsify some clause of F. Therefore, the number of clauses in F should be $\geq 2^{n-2}$. This implies that the number of lines in S is at least 2^{n-2}. Therefore, the number of lines in Π must also be exponential in n. ◀

▷ Claim 17 ([5]). Either for all $i \in [n-1]$, C_2 has a variable of the form x_{i*}, or for all $j \in [n]$, C_2 has a variable of the form x_{*j}.

Proof. At this point in the proof of Lemma 15, we definitely know that $\overline{a_n} \in C_2$, and for all $i \in [n-1]$, for all $j \in [n]$, $x_{ij} \notin \text{var}(H_2^z)$. We prove this claim by contradiction. Suppose the claim is wrong i.e, there exists some $u \in [n-1]$ where for all $l \in [n]$ $x_{ul} \notin var(C_2)$ and some $v \in [n]$ where for all $k \in [n]$ $x_{kv} \notin var(C_2)$.

Let ρ be the minimum partial assignment falsifying C_2. Then we know that :

- ▷ ρ sets $a_n = 1$, leaves all other variables in $A \cup B$ unset, since they $\notin C_2$.
- ▷ ρ does not set any x_{ul} or x_{kv}, since by our assumptions they all are not in C_2.

Now, extend ρ to assignment α by setting:

- ▷ $a_u = b_v = 0$ and rest all unset variables from $A \cup B$ to 1.
- ▷ Also except x_{uv}, set $x_{u*} = 1$ and $x_{*v} = 0$.

Observe that the assignment α satisfies all axiom clauses except A_{uv} and B_{uv} and does not falsify any axiom.

Now extend α to α_0 and α_1 by setting $x_{uv} = 0$ and 1 respectively.

The extension α_0 satisfies one more axiom i.e. B_{uv}; similarly α_1 satisfies one more axiom i.e. A_{uv}. Note that they still do not falsify the remaining axiom. That is, α_0 does not falsify A_{uv} and similarly, α_1 does not falsify B_{uv}.

α_0 and α_1 agree everywhere except on x_{ij} , and since $x_{ij} \notin \text{var}(H_2^z)$, it follows that $H_2^z(\alpha_0) = H_2^z(\alpha_1)$, say this value is equal d.

From the proved Induction in Lemma 4, the partial strategy of universal player at every line combined with the extension of the existential assignment falsifying its clause part, should falsify some axiom of the QBF. Also, α_0 and α_1 falsify C_2, since they extend ρ. Hence, it is a contradiction that $(\alpha_{\overline{d}}, d)$ satisfies all axioms. Therefore, the claim needs to be true. ◁

On Hardness of Testing Equivalence to Sparse Polynomials Under Shifts

Suryajith Chillara
International Institute of Information Technology, Hyderabad, India

Coral Grichener
Google Research, Herzliya, Israel

Amir Shpilka
Tel Aviv University, Israel

Abstract

We say that two given polynomials $f, g \in R[x_1, \ldots, x_n]$, over a ring R, are equivalent under shifts if there exists a vector $(a_1, \ldots, a_n) \in R^n$ such that $f(x_1 + a_1, \ldots, x_n + a_n) = g(x_1, \ldots, x_n)$. This is a special variant of the polynomial projection problem in Algebraic Complexity Theory.

Grigoriev and Karpinski (FOCS 1990), Lakshman and Saunders (SIAM J. Computing, 1995), and Grigoriev and Lakshman (ISSAC 1995) studied the problem of testing polynomial equivalence of a given polynomial to *any* t-sparse polynomial, over the rational numbers, and gave exponential time algorithms. In this paper, we provide hardness results for this problem.

Formally, for a ring R, let $\mathsf{SparseShift}_R$ be the following decision problem – Given a polynomial $P(X)$, is there a vector $\mathbf{a}$ such that $P(X + \mathbf{a})$ contains fewer monomials than $P(X)$. We show that $\mathsf{SparseShift}_R$ is at least as hard as checking if a given system of polynomial equations over $R[x_1, \ldots, x_n]$ has a solution (Hilbert's Nullstellensatz). As a consequence of this reduction, we get the following results.

1. $\mathsf{SparseShift}_{\mathbb{Z}}$ is undecidable.
2. For any ring R (which is not a field) such that HN_R is NP_R-complete over the Blum-Shub-Smale model of computation, $\mathsf{SparseShift}_R$ is also NP_R-complete. In particular, $\mathsf{SparseShift}_{\mathbb{Z}}$ is also $\mathsf{NP}_{\mathbb{Z}}$-complete.

We also study the gap version of the $\mathsf{SparseShift}_R$ and show the following.

1. For every function $\beta : \mathbb{N} \to \mathbb{R}_+$ such that $\beta \in o(1)$, N^{β}-gap-$\mathsf{SparseShift}_{\mathbb{Z}}$ is also undecidable (where N is the input length).
2. For $R = \mathbb{F}_p, \mathbb{Q}, \mathbb{R}$ or $\mathbb{Z}_q$ and for every $\beta > 1$ the β-gap-$\mathsf{SparseShift}_R$ problem is NP-hard. Furthermore, there exists a constant $\alpha > 1$ such that for every $d = O(1)$ in the sparse representation model, and for every $d \leq n^{O(1)}$ in the arithmetic circuit model, the α^d-gap-$\mathsf{SparseShift}_R$ problem is NP-hard when given polynomials of degree at most d, in $O(nd)$ many variables, as input.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Algebraic complexity theory

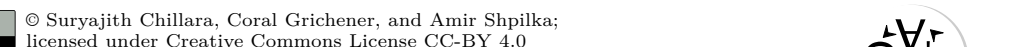
Keywords and phrases algebraic complexity, shift equivalence, polynomial equivalence, Hilbert's Nullstellensatz, hardness of approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.22

Related Version *Full Version*: `https://eccc.weizmann.ac.il/report/2022/106/`

Funding *Suryajith Chillara*: Part of this work was done while the author was visiting Tel Aviv University, hosted by Amir Shpilka.
Amir Shpilka: The research leading to these results received funding from the Israel Science Foundation (grant number 514/20) and from the Len Blavatnik and the Blavatnik Family foundation.

© Suryajith Chillara, Coral Grichener, and Amir Shpilka;
licensed under Creative Commons License CC-BY 4.0



1 Introduction

This paper studies the following question: given an n-variate polynomial $f(X)$, over a ring R[1], how difficult is the task of finding a shift $\mathbf{b} \in R^n$ such that $f(X+\mathbf{b})$ has fewer monomials than f.

Before proceeding we would like to discuss the issue of representation of polynomials. There are several natural settings – representation as vector of coefficients or as arithmetic circuits – and two different models – the white-box and black-box models. The most obvious representation is the *dense representation* in which n-variate polynomials of degree d are represented as a vectors of coefficients of length $\binom{n+d}{d}$. In this setting we assume that the vector is given as input to the algorithm. A more concise representation is the *sparse representation* in which a polynomial is represented as a list of pairs of exponent vectors and coefficients. In the black-box setting we only assume that the algorithm has black-box access to the polynomial (though the important parameters such as number of variables and degree are known to the algorithm). I.e., the algorithm is restricted to asking the polynomial for its values on different inputs. Another natural model is representing polynomials as arithmetic circuits. That is, the algorithm will get as input an arithmetic circuit computing the polynomial. In the white-box setting the algorithm is explicitly given the circuit so it has access to the graph of computation etc. In the black-box model the algorithm only has black-box access to the circuit (though the important parameters such as size, depth, number of variables etc. are known to the algorithm).

One of the most important questions in the area of Algebraic Complexity Theory is the problem of checking if two polynomials are equivalent under affine transformations. In generality this problem is also called the polynomial projection problem. Ignoring issues of representations the problem is the following.

Polynomial Projection ($\mathsf{PolyProj}_{\mathbb{F}}$):
Given two polynomials $f \in \mathbb{F}[y_1,\ldots,y_m]$ and $g \in \mathbb{F}[x_1,\ldots,x_n]$, over a field $\mathbb{F}$, output an $m \times n$ matrix A and a vector $\mathbf{b} \in \mathbb{F}^m$ such that $g(x_1,\ldots,x_n) = f(A \cdot \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix}^T + \mathbf{b})$ if such a pair exists, or output "FAIL" otherwise.

For example, the holy grail of algebraic complexity, Valiant's Extended Hypothesis is an instance of the polynomial projection problem. Recall that the hypothesis says that the permanent of an $n \times n$ matrix cannot be represented as a polynomial projection of determinant of any $m \times m$ matrix, for any m that is polynomial in n [27]. Kayal [19] showed that the problem of polynomial projection is NP-hard in general. However, for specific instances of the polynomial g, under the requirement that the matrix A has full rank (or that it is random), Kayal [19] gave efficient randomized algorithms in the black-box model (i.e. assuming only black-box access to f).

Since studying polynomial equivalence under such projections is NP-hard in general, the following *simpler* question was considered.

Polynomial Equivalence under Shifts ($\mathsf{ShiftEquiv}_{\mathbb{F}}$):
Given two polynomials $f, g \in \mathbb{F}[x_1,\ldots,x_n]$ output a vector $(b_1,\ldots,b_n) \in \mathbb{F}^n$ such that $g(x_1,\ldots,x_n) = f(x_1+b_1,\ldots,x_n+b_n)$ if such a vector exists, or output "FAIL" otherwise.

[1] From now on, R always denotes an integral domain, i.e. a commutative ring with a unit, which is also a domain, and $\mathbb{F}$ a field ($\mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$ are, as usual, the rational, real and complex fields, respectively).

To the best of our knowledge the notion of studying polynomial equivalence under shifts first appeared in [12] and it was formally addressed by Grigoriev in [10]. For polynomials of degree d over n variables, Grigoriev [10] gave a deterministic algorithm over fields of zero characteristic, a randomized algorithm over prime residue fields, and a quantum algorithm over fields of characteristic 2, all of which run in time polynomial in the dense representation. That is, the running time is polynomial in $\binom{n+d}{d}$. If the degree of the polynomial grows as a function of the number of variables or vice versa, the algorithms presented by Grigoriev require *exponential* time in the number of variables, even if the polynomial can be represented by a small arithmetic circuit or if it has polynomially many monomials. It is a natural question to ask if the complexity of the algorithms can be brought down when the input to the algorithm is provided in some succinct representation – for example, as an arithmetic circuit. In such a setting, Dvir, Oliveira and Shpilka [8] showed that given just a black box access to the polynomials f and g on n variables, and given a bound on the degree d and circuit size s, there is a randomized algorithm that runs in time $\text{poly}(n, d, s)$ and solves the polynomial equivalence under shifts problem. The randomness in their algorithm only stems from polynomial identity testing (PIT), which is a sub-routine of their algorithm, and hence equivalence under shifts in this setting can be derandomized if and only if PIT can be derandomized (clearly PIT is a special case of equivalence under shifts when g is the zero polynomial).

A polynomial f is said to be t-sparse if the number of monomials with non-zero coefficients in f is at most t. In the literature, an n variate polynomial is generally said to be sparse if the number of monomials in it with non-zero coefficients is at most $\text{poly}(n)$. Equivalently, a sparse polynomial is a polynomial that can be computed by a depth two $\Sigma\Pi$ arithmetic circuit with a polynomial bound on the top fan-in. Sparse polynomials are extremely well studied because of their simplicity and as a result many efficient algorithmic results are known for them [1, 18, 4, 6, 13, 11, 12, 21, 25].

A variant of the polynomial projection problem asks if a given polynomial is equivalent to a sparse polynomial under affine transformations. This can be seen as a variant of the classical Minimum Circuit Size Problem (MCSP) where given the truth table of a function we wish to find the minimal circuit computing it. In this case the circuit we are seeking is a very structured $\Sigma\Pi\Sigma$ circuit that is obtained by composing a $\Sigma\Pi$ circuit with an affine transformation. As this set of polynomials is dense inside the class $\Sigma\Pi\Sigma$ it is an interesting family to study (see [23]). Grigoriev and Karpinski [12] were the first to consider this variant of the polynomial projection problem. Specifically, they studied the following problem (in the dense representation model) – given a polynomial $P(X)$, over the rationals, and a parameter t output a matrix A and a vector $\mathbf{b}$, if they exist, such that the polynomial $P(A \cdot X + \mathbf{b})$ has at most t monomials. They gave an algorithm whose complexity is $O(M \cdot d^{n^4})$ where M is a bound on the size of coefficients of the input polynomial. Lakshman and Saunders [21] considered the problem of testing the equivalence of univariate polynomials (over $\mathbb{Q}$) to t-sparse polynomials under just shifts instead of affine linear transformations. They provided sufficient conditions for uniqueness and rationality of a t-sparsifying shift. Grigoriev and Lakshman [14] extended these criterion to multivariate polynomials. They also gave algorithms for polynomials with finitely many sparsifying shifts[2] that run in deterministic time $(dt)^{O(n)}$ and randomized time $t^{O(n)}$. In the past two decades, these exponential time algorithms could not be improved and this is a major motivation behind our study of hardness of this problem. We state the following more general problem to allow polynomials over rings.

[2] Over $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ etc., it may happen that there are infinitely many t-sparsifying shifts for a given polynomial. Grigoriev and Lakshman [14] give algorithms for polynomials that are guaranteed to have finitely many t-sparsifying shifts.

Sparsification of Polynomials via Shifts ($\mathsf{SparseShift}_R$):
Given a polynomial $f \in R[x_1, \ldots, x_n]$, decide if there exists a vector $(a_1, \ldots, a_n) \in R^n$ such that $f(x_1 + a_1, \ldots, x_n + a_n)$ has strictly fewer monomials with non-zero coefficients than $f(x_1, \ldots, x_n)$, or output "FAIL" if no such vector exists.

In this paper we show that the problem $\mathsf{SparseShift}_R$ is at least as hard as as checking if a given system of polynomial equations over $R[x_1, \ldots, x_n]$ has a solution (Hilbert's Nullstellensatz).

Hilbert's Nullstellensatz: Given a system S of polynomial equations $S = \{f_1 = 0, \ldots, f_r = 0\}$ over the polynomial ring $R[x_1, \ldots, x_n]$, we say that the system is satisfiable if there exists an assignment $\mathbf{a} \in R^n$ to the variables that simultaneously satisfies all equations in S. This problem has a great significance in Algebraic Geometry and has other important applications in diverse areas. We state a slightly restricted version of Hilbert's Nullstellensatz problem that asks for a common solution in a specific domain (the general version asks for a solution in the algebraic closure). This definition is similar to the definition in the Blum, Shub and Smale model of computation [2].

Hilbert's Nullstellensatz over a ring R (HN_R):
Given a system of polynomial equations $\{f_1 = 0, \ldots, f_r = 0\}$ over $R[x_1, \ldots, x_n]$, decide whether there exist a vector $(a_1, \ldots, a_n) \in R^n$ such that for all $i \in [r]$, $f_i(a_1, \ldots, a_n) = 0$, or output "FAIL" if no such vector exists.

With this background, we shall now state our first main result that gives a reduction from Hilbert's Nullstellensatz problem to polynomial sparsification.

▶ **Theorem 1.** *Let R be an integral domain, which is not a field. Then $\mathsf{SparseShift}_R$ is HN_R-hard, in any of the white-box representations.*

If R is arbitrary, then the polynomials could have coefficients with arbitrary bit complexity. Thus, it is important for us to also specify the model of computation over which this problem is being considered. In the Turing machine model, assuming that $f_1, \ldots, f_r \in \mathbb{C}[x_1, \ldots, x_n]$ have integral coefficients, Koiran [20] showed (by assuming that the Generalized Riemann Hypothesis is true) that $\mathsf{HN}_{\mathbb{C}}$ can be solved in the second level of polynomial hierarchy. Without the GRH assumption, the only known upper bound for $\mathsf{HN}_{\mathbb{C}}$ is PSPACE. For $R = \mathbb{Z}$, Matiyasevich [22] showed that this problem is undecidable (also see [7]). Putting these together with Theorem 1 we get the following consequence.

▶ **Corollary 2.** $\mathsf{SparseShift}_{\mathbb{Z}}$ *is undecidable.*

It is important to note that under sparse or dense representations, $\mathsf{SparseShift}_R$ is in NP_R. That is, given $\mathbf{a}$, we can efficiently verify if it is a sparsifying shift for a polynomial $P(X)$ using at most polynomially many algebraic operations using sparse polynomial interpolation, and sparse polynomial identity testing. Thus for any integral domain R (which is not a field) such that HN_R is NP_R-complete, over the Blum-Shub-Smale model of computation [2], we get the following corollary from the aforementioned statements and Theorem 1.

▶ **Corollary 3.** *Let R be an integral domain (but not a field) such that HN_R is NP_R-complete over the Blum-Shub-Smale model of computation. Then $\mathsf{SparseShift}_R$ is also NP_R-complete.*

In particular, we get that $\mathsf{SparseShift}_{\mathbb{Z}}$ is also $\mathsf{NP}_{\mathbb{Z}}$-complete.

These results, to some extent, shed a light on why this problem in general has been evading the efforts to provide efficient algorithms.

Note that our problem $\mathsf{SparseShift}_R$ can also be viewed as a gap decision problem – given a polynomial $P(X)$ of sparsity t, is there a vector $\mathbf{a}$ such that $P(X+\mathbf{a})$ has at most $t-1$ monomials. Let us formally define a more general gap version of $\mathsf{SparseShift}_R$.

α-gap-$\mathsf{SparseShift}_R$:
Let $\alpha > 1$ be a parameter. Given a polynomial $P \in R[X]$ and a parameter t, ▬ output YES if there exists a vector $\mathbf{a}$ such that $P(X+\mathbf{a})$ has at most t monomials, and ▬ output NO if for all vectors $\mathbf{a}$, $P(X+\mathbf{a})$ has at least αt monomials.

Using gap amplification we reduce $\mathsf{SparseShift}_R$ to N^β-gap-$\mathsf{SparseShift}_R$ for all functions $\beta \in o(1)$. We thus get our second main result.

▶ **Theorem 4.** *For every function $\beta : \mathbb{N} \to \mathbb{R}_+$ such that $\beta \in o(1)$, N^β-gap-$\mathsf{SparseShift}_\mathbb{Z}$ is undecidable (where N is the input length).*

In Theorem 4, we used the undecidability of $\mathsf{HN}_\mathbb{Z}$ to infer the undecidability of N^β-gap-$\mathsf{SparseShift}_\mathbb{Z}$. However, we do not have such results for rings $R \neq \mathbb{Z}$ (over Turing machine model). Furthermore, $\mathsf{HN}_\mathbb{Q}$ is not known to be undecidable and, as mentioned above, over $\mathbb{C}$ it is decidable as well as over finite fields. Thus for $R = \mathbb{F}_p, \mathbb{Q}, \mathbb{R}$ or $\mathbb{Z}_q$, we present a different reduction of gap problems – from $(1-\varepsilon,\delta)$-gap-Max-3Lin$_R$ to α-gap-$\mathsf{SparseShift}_R$ and infer NP-hardness results for α-gap-$\mathsf{SparseShift}_\mathbb{R}$.

$(1-\varepsilon,\delta)$-gap-Max-3Lin$_R$:
Given a system of linear equations $\{L_1 = 0, \ldots, L_m = 0\}$ over $R[x_1, \ldots, x_n]$ each of which depends on exactly 3 variables, ▬ output YES if at least $(1-\varepsilon)$ fraction of equations can be simultaneously satisfied, and ▬ output NO if at most δ fraction of equations can be simultaneously satisfied.

We say that it is NP-hard to $(1-\varepsilon,\delta)$-approximate Max-3Lin$_R$ if the decision problem $(1-\varepsilon,\delta)$-gap-Max-3Lin$_R$ is NP-hard. Using this notion, we summarize non-exhaustively some known NP-hardness results for $(1-\varepsilon,\delta)$-approximating Max-3Lin$_R$.

Table 1 Non-exhaustive list of known NP-hardness results for approximating Max-3Lin$_R$.

Result	Ring R	NP-Hardness for
Håstad [17]	$\mathbb{F}_p$	$\forall\ \varepsilon > 0$, $(1-\varepsilon, \frac{1+\varepsilon}{p})$-approximation
Håstad [17]	$\mathbb{Z}_q$ for $q \in \mathbb{N}$	$\forall\ \varepsilon,\delta > 0$, $(1-\varepsilon, \frac{1}{q}+\delta)$-approximation
Feldman, Gopalan, Khot and Ponnuswami [9]	$\mathbb{Q}$	$\forall\ \varepsilon > 0$, $(1-\varepsilon,\varepsilon)$-approximation
Gurswami and Raghavendra [15, 16]	$\mathbb{Q}, \mathbb{R}$	$\forall\ \varepsilon,\delta > 0$, $(1-\varepsilon,\delta)$-approximation

Thus, a gap reduction from $(1-\varepsilon',\delta')$-gap-Max-3Lin$_R$ (where ε' and δ' are as in the last column of Table 1) to α-gap-$\mathsf{SparseShift}_R$ (for $\alpha = \alpha(\varepsilon',\delta',R)$) implies hardness of α-gap-$\mathsf{SparseShift}_R$ and using amplification we get our third main result. We first state it in the sparse representation model and then in the arithmetic circuit model.

▶ **Theorem 5** (Sparse representation). *For $R = \mathbb{F}_p, \mathbb{Q}, \mathbb{R}$ or $\mathbb{Z}_q$ and for every $\beta > 1$ the β-gap-$\mathsf{SparseShift}_R$ problem is NP-hard. Furthermore, there exists a constant $\alpha > 1$ such that for every $d = O(1)$ the α^d-gap-$\mathsf{SparseShift}_R$ problem is NP-hard when given polynomials of degree at most d as input.*

The theorem is stated for the sparse representation model but as the polynomials under consideration have many non-zero terms it can also be stated without any modification in the dense representation model. We next state the theorem in the arithmetic circuit model.

▶ **Theorem 6** (Arithmetic circuit representation). *There exists a constant $\alpha > 1$ such that the following holds for $R = \mathbb{F}_p, \mathbb{Q}, \mathbb{R}$ or $\mathbb{Z}_q$. For every d the α^d-gap problem is NP-hard when given polynomials of degree at most d as input. Furthermore, our hard instances have circuit size $(nd + 1)$.*

Observe that if we take e.g. $d = n^c$ in the theorem above then the input size is $N = n^{c+1}$ and the gap is $\exp(N^{1-1/c})$.

2 Preliminaries

We use $[n]$ to refer to the set $\{1, 2, \ldots, n\}$. We use capital letters A and C to represent matrices, capital letters U, S and T to represent systems of equations, and capital letters X, Y and Z to represent sets of variables. We reserve letters x, y and z with, or without subscripts, to represent variables. We use bold letters $\mathbf{a}, \mathbf{b}, \ldots$ to indicate vectors and non-bold letters (apart from x, y and z) with, or without subscripts, $e, b_i, a_j, A_{i,j}, C_{k,\ell}, \ldots$ to indicate scalars.

Let S be a system of polynomial equations $\{f_i = 0\}_{i=1}^r$, where $\deg(f_i) = d_i$. We use $\mathrm{Vars}(f)$ to denote the variable support of the polynomial f, and for a system S of polynomial equations we use $\mathrm{Vars}(S)$ to denote the union of $\mathrm{Vars}(f_i)$ for all equations $f_i = 0$ in S. We denote with L an upper bound on the bit-complexity of the coefficients of the polynomials in the system.[3] The total degree of S is $d = \sum_i d_i$.

In this paper we shall consider two types of representations of polynomials (and hence of polynomial equations). The representation that is typically studied in the context of polynomial equations is the so called "sparse representation". In this representation polynomials are given as a set of pairs consisting of exponent vectors together with the coefficients of the corresponding monomials. E.g. the polynomial $2x^2z - y \in \mathbb{F}[x, y, z,]$ is represented as $\{((2, 0, 1), 2), (0, 1, 0), -1)\}$. This is called the sparse representation as we do not charge for monomials whose coefficients are equal to 0. In particular the size of the representation of a degree d polynomial can be much smaller than $\binom{n+d}{d}$. For a system of polynomial equations $S = \{f_i = 0\}_{i=1}^r$, the complexity of S, or its size, is defined to be the total bit size of the sparse representations of the polynomials $\{f_i\}_{i\in[r]}$. We note that this is always upper bounded by $\sum_{i=1}^r \binom{n+d_i}{d_i} \cdot L$.

The second type of representation that we consider is when the polynomials f_i are given as the outputs of arithmetic circuits (see [26, 24] for more details). In this paper we only consider the white-box version of this representation, i.e., when the computation graph of the circuit is explicitly given to the algorithm. In this case the complexity (or size) of the system S is the total size of the input circuits times the maximal bit complexity of coefficients in the circuits.

[3] When the underlying ring is an abstract ring one has to define this complexity, but for the usual rings and fields such as $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{F}_q$ this is the natural definition. In the BSS model this complexity is called height and is indeed only defined for these natural domains [2].

As we shall later see (Lemmata 7 and 8), given a system of equations, either via arithmetic circuits or in the sparse representation model, one can easily construct an equivalent system T of polynomial equations of degree 2 and roughly of the same complexity, such that the system S has a solution if and only if the system T does[4]. Hence, these two different representations have the same computational power. However, this reduction is not gap-preserving so we will have to give separate arguments for the gap problems.

3 Reduction from HN_R to $\mathsf{SparseShift}_R$

In this section, we shall first show that given a system S of r many polynomial equations over $R[X]$, we can algorithmically construct a system T of polynomial equations over $R[X'']$ such that $X \subseteq X''$; each polynomial in T is of degree at most 2; and if $\mathbf{a} \in R^{|X|}$ is a solution for the system S then there exists an extension $\mathbf{a}'$ of $\mathbf{a}$ such that $\mathbf{a}'$ is a solution for the system T. And vice versa, from a solution to T we deduce a solution to S. From T we shall then construct a polynomial $P_S \in R[X'', W]$ such that S has a solution if and only if the polynomial P_S can be sparsified.

3.1 Reduction to a system of polynomial equations of degree at most 2

Given a system S of polynomial equations, Lemma 7 and Lemma 8 summarize the construction of a system T which is as described in the paragraph above, for inputs given in sparse, and arithmetic circuit representations respectively.

▶ **Lemma 7** (Sparse representation). *Let $S = \{f_i(X) = 0\}_{i=1}^{r}$ be a system of polynomial equations over the polynomial ring $R[X]$ such that for each $i \in [r]$, $f_i(X)$ is a polynomial of degree d_i and sparsity s_i, and the bit complexity of each coefficient is at most L. Then, Algorithm 1 runs in time $\mathrm{poly}(|X|, r, \max_i\{d_i\}, \max_i\{s_i\}, L)$ and returns a set T of polynomial equations $\{g_j(X, Y, Z) = 0\}_{j=1}^{t}$ over the polynomial ring $R[X, Y, Z]$ such that*

- $t = \mathrm{poly}(|X|, r, \max_i\{d_i\}, \max_i\{s_i\}, L)$.
- $|X \sqcup Y \sqcup Z| = \mathrm{poly}(|X|, t, \max_i\{d_i\}, \max_i\{s_i\})$.
- *The bit-complexity of the coefficients of the polynomial equations in T is also at most L.*
- *Each polynomial equation $g_j(X, Y, Z) = 0$ in T is either a quadratic binomial polynomial equation or an affine linear polynomial equation.*
- *S has a solution $\mathbf{a} \in R^{|X|}$ if and only T has a solution $\mathbf{a}' \in R^{|X \sqcup Y \sqcup Z|}$.*

▶ **Lemma 8** (Arithmetic circuit representation). *Let $S = \{f_i(X) = 0\}_{i=1}^{r}$ be a system of polynomial equations over the polynomial ring $R[X]$ such that for each $i \in [r]$, the polynomial $f_i(X)$ is provided as an arithmetic circuit Φ_i of size s_i. Then, when given this as input, Algorithm 3 runs in time $\mathrm{poly}(|X|, r, \max_i\{s_i\})$ and returns a system T of polynomial equations $\{g_j(X, Y) = 0\}_{j=1}^{t}$ over the polynomial ring $R[X, Y]$ such that*

- $t = \mathrm{poly}(|X|, r, \max_i\{s_i\})$.
- $|Y| \leq r \cdot \max_i\{s_i\}$.
- *Each polynomial equation $g_j(X, Y) = 0$ in T is either a quadratic binomial polynomial equation or an affine linear polynomial equation.*
- *T has a solution in $R^{|X \sqcup Y|}$ if and only S has a solution in $R^{|X|}$.*

Due to the lack of space, we present the relevant details and proofs of Lemmata 7 and 8 in Appendix A.

[4] Such reductions were studied before. See [3, Proposition 1] and [5, Chapter 9].

3.2 Construction of P_S

Given a system S of polynomial equations over a set of variables X, in Section 3.1 we constructed the system T of polynomial equations of degree at most 2 over the set of variables X, Y and Z,[5] such that $|X \sqcup Y \sqcup Z|$ is at most polynomial in the input size. Without loss of generality, let the variables in $X \sqcup Y \sqcup Z$ be renamed as the variable set $X' = \{x_1, \ldots, x_N\}$ where $N = |X \sqcup Y \sqcup Z|$.

Let $\{g_1(X') = 0, \ldots, g_t(X') = 0\}$ be the enumeration of polynomial equations in T. Without loss of generality, we can assume that the number of equations with a non-zero constant term is equal to 1. Otherwise, given a system T of polynomial equations, with $t' > 1$ many of these polynomial equations with non-zero constant terms, we shall construct a new system T' such that the number of polynomial equations in T' that have non-zero constant terms is exactly equal to 1, and a solution of T is a solution of T' and vice versa. Without loss of generality assume that $\{g_1(X') = 0, \ldots, g_{t'}(X') = 0\}$ are the polynomial equations in T with non-zero constant terms. By Lemmata 7 and 8 it follows that $\{g_1(X') = 0, \ldots, g_{t'}(X') = 0\}$ are affine linear equations. Denote the free term in $g_1(X'), \ldots, g_{t'}(X')$ with $c_1, \ldots, c_{t'}$, respectively. We obtain T' from T by just updating each of the polynomials $g_i(X')$ (for $2 \leq i \leq t'$) as follows: $g_i(X') \leftarrow c_1 \cdot g_i(X') - c_i \cdot g_1(X')$. The rest of the polynomials from T are directly added to T'. It is easy to see that any solution to the system T is also a solution to the system T' and vice-versa (as R is an integral domain). Furthermore, the only equation in T' with a non-zero constant term is an affine linear equation.

Conditioned on the aforementioned discussion, we shall assume that all polynomial equations in T other than $g_1(X') = 0$, have no constant terms. For a new variable x_0, let $X'' = X' \sqcup \{x_0\}$ and thus $|X''| = N + 1$. Let $W = \{w_1, \ldots, w_t\}$ be a new set of variables disjoint from X''. Let γ be an element in R without a multiplicative inverse (recall that in Theorem 1 we assume that R is not a field). We shall now define our polynomial P_S in the polynomial ring $R[X'', W]$ as follows

$$P_S(X'', W) = \underbrace{w_1 \cdot g_1(X')}_{I} + \underbrace{\left(\sum_{i=2}^{t} w_i \cdot \left(\gamma \cdot g_i(X') + \sum_{k=0}^{N} x_k\right)\right)}_{II}. \tag{1}$$

Observe that $\deg(P_s) \leq 3$.

▶ Remark 9. The sparsity of the polynomial $P_S(X'', W)$, σ, is equal to the sum of sparsities of polynomials in each of its summands, and it is equal to $(t-1) \cdot (N+1) + \sum_{i=1}^{t} s'_i$ where s'_i is the sparsity of the polynomial $g_i(X')$. On the other hand, $P_S(X'', W)$ can also be represented as a depth four arithmetic circuit, with at most $3t$ non-leaf nodes.

We shall now show that it is sufficient to consider shifts with a certain structure for P_S. Further we shall show that a solution to the system S of polynomial equations exists if and only if there exists a vector $\mathbf{b}$ such that $P_S(X'' + \mathbf{b}, W)$ has fewer monomials than $P_S(X'', W)$.

▶ **Lemma 10.** *Let $\mathbf{a} = \{a_1, \ldots, a_N\} \in R^N$ be a solution to the system T of polynomial equations. Let $\mathbf{b} = \{b_0, \ldots, b_N\}, \mathbf{b}' = \{b'_0, \ldots, b'_N\} \in R^{N+1}$ and $\mathbf{a}' = \{a'_0, \ldots, a'_N\}$ be such that*

- *$b_i = b'_i$ for all $i \in [N]$,*
- *$b'_0 \neq b_0$ and $b_0 = -\sum_{i \in [N]} b_i$,*

[5] In case the polynomials in the system S of polynomial equations are provided as circuits, $Z = \emptyset$.

- $a'_i = a_i$ *for all* $i \in [N]$, *and*
- $a'_0 = -\sum_{i\in[N]} a_i$.

Let $\mathbf{b}''$ *and* $\mathbf{c}$ *be any vectors in* R^{N+1} *and* R^t *respectively. Then,*

1. *The sparsity of* $P_S(X'' + \mathbf{b}'', W + \mathbf{c})$ *is at least that of* $P_S(X'' + \mathbf{b}'', W)$,
2. *The sparsity of* $P_S(X'' + \mathbf{b}', W)$ *is at least that of* $P_S(X'' + \mathbf{b}, W)$.
3. *The sparsity of* $P_S(X'', W)$ *is* 1 *more than that of* $P_S(X'' + \mathbf{a}', W)$.

Proof. Given the structure of the polynomial $P_S(X'', W)$, proof of Item 1 follows directly from the fact that the polynomial $P_S(X'', W)$ is linear in the W variables and thus all terms of $P_S(X'' + \mathbf{b}'', W)$ also appear in $P_S(X'' + \mathbf{b}'', W + \mathbf{c})$. Further, the difference $P_S(X'' + \mathbf{b}'', W + \mathbf{c}) - P_S(X'' + \mathbf{b}'', W)$ does not depend on any W variable.

From their definition, the vectors $\mathbf{b}$ and $\mathbf{b}'$ are identical when projected down to their last N coordinates and these exactly correspond to shifts of variables in X'. We shall use $\mathbf{b}|_{X'}$ to denote this projection. In particular, for all $i \in [t]$, $g_i(X' + \mathbf{b}|_{X'}) = g_i(X' + \mathbf{b}'|_{X'})$. It is easy to see that the polynomial $\sum_{i=0}^{N} x_i$ is invariant under shift by $\mathbf{b}$ (as $\sum_{i=0}^{N} b_i = 0$) but not under shift by $\mathbf{b}'$. Putting both of these facts together we can now say that $P_S(X'' + \mathbf{b}', W)$ contains all the terms that are contained in $P_S(X'' + \mathbf{b}, W)$, and it additionally contains a non-trivial linear polynomial in the W variables. This proves Item 2 of the lemma.

Towards proving Item 3 of the lemma, we claim that under a shift by $\mathbf{a}'$, as defined in the statement of the lemma, sparsity of part II in Equation (1) does not change, and sparsity of part I definitely decreases.

All polynomial equations $\{g_i(X') = 0 \mid 2 \le i \le t\}$, are constant free and can either be quadratic binomial polynomial equations of the form $(x_p - x_q \cdot x_e) = 0$ (for some $p, q, e \in [N]$) or homogeneous linear polynomial equations of the form $\sum_{j=1}^{k} c_{i_j} x_{i_j} = 0$ (for some $i_1, \ldots, i_k \in [N]$ and scalars c_{i_j}).

When the equation is a quadratic binomial polynomial equation: Since $\mathbf{a} = \mathbf{a}'|_{X'}$ solves T we get that $a'_p - a'_q \cdot a'_e = 0$ and using this fact we can show that for each summand of this kind in part II, the sparsity does not change.

$$\begin{aligned}
&\gamma \cdot ((x_p + a'_p) - (x_q + a'_q) \cdot (x_e + a'_e)) + \sum_{i=0}^{N} (x_i + a'_i) \\
&= \gamma \cdot ((x_p - x_q \cdot x_e) - (a'_q x_e + a'_e x_q)) + \sum_{i=0}^{N} x_i \quad \text{(Since } a'_p - a'_q \cdot a'_e = 0 \text{ and } \sum_{i=0}^{N} a'_i = 0\text{)} \\
&= \gamma \cdot (x_p - x_q \cdot x_e) + \left(\sum_{i\in[N]\setminus\{q,e\}} x_i \right) + (1 - \gamma \cdot a'_q) \cdot x_e + (1 - \gamma \cdot a'_e) \cdot x_q.
\end{aligned}$$

Since γ has no multiplicative inverse, neither $(1 - \gamma \cdot a'_q)$ nor $(1 - \gamma \cdot a'_e)$ can be equal to 0.

When the polynomial equation is a homogeneous linear polynomial equation: Since $\mathbf{a} = \mathbf{a}'|_{X'}$ solves T we get that $\sum_{j=1}^{k} c_{i_j} a'_{i_j} = 0$ and thus the sparsity remains invariant for such summands in part II.

Finally consider the non-homogeneous linear polynomial equation $g_1(X') = 0$. Without loss of generality, let $g_1(X') = c_1 x_1 + \ldots + c_k x_k + c$. Note that $\sum_{i=1}^{k} c_i a'_i + c = 0$ as $\mathbf{a} = \mathbf{a}'|_{X'}$ solves T and thus sparsity reduces by 1 under shift by such a vector $\mathbf{a}'$:

$$\sum_{i=1}^{k} c_i (x_i + a'_i) + c = \sum_i c_i x_i + \left(\sum_{i=1}^{k} c_i a'_i + c \right) = \sum_{i=1}^{k} c_i x_i \ .$$

By putting together the analysis for all the summands we get that the polynomial $P_S(X'' + \mathbf{a}', W)$ has one monomial less than $P_S(X'', W)$. ◀

▶ **Lemma 11.** *Let* $\mathbf{b} = (b_0, b_1, \ldots, b_N) \in R^{N+1}$ *such that* $b_0 = -\sum_{i=1}^{N} b_i$*, be a shift that sparsifies the polynomial* $P_S(X'', W)$ *by at least one monomial. Let* $\mathbf{a} \in R^N$ *be the projection of vector* $\mathbf{b}$ *to its last* N *coordinates. Then* $\mathbf{a}$ *solves* T*.*

Proof. The proof of Lemma 10 shows that given the structure of the shift $\mathbf{b}$, a reduction in sparsity can only come from $g_1(X')$. That is, the sparsity of polynomials $g_i(X')$ for $i \geq 2$, can only increase upon a shift.

For the sake of contradiction, let us assume that there exists a polynomial equation in T that is not satisfied by $\mathbf{a}$. If for some $i \geq 2$, $g_i(X') = 0$ is a polynomial equation that is not satisfied by $\mathbf{a}$, then this contributes an increase of 1 to sparsity of the polynomial $P_S(X'', W)$ upon the shift by $\mathbf{b}$ (by adding a term of the form $c \cdot w_i$). Else if $g_1(X') = 0$ is not satisfied by $\mathbf{a}$, then there is no contribution to reduction in sparsity from part I. This is due to the fact that the term of the form $c \cdot w_1$ vanishes upon a shift by $\mathbf{b}$ if and only if $\mathbf{a}$ solves $g_1(X') = 0$. In either of these cases, the sparsity of $P_s(X'' + \mathbf{b}, W)$ is not strictly less than that of $P_S(X'', W)$. This contradicts our assumption that $\mathbf{b}$ sparsifies $P_S(X'', W)$ by at least one monomial. ◀

By putting together Lemmata 7, 8, 10, and 11, we get the following formal statement.

▶ **Theorem 12** (HN_R reduces to $\mathsf{SparseShift}_R$)**.** *Given a system* S *of polynomial equations over the polynomial ring* $R[X]$*, there exists a polynomial* $P_S(X'', W) \in R[X'', W]$ *(where* $X \subseteq X''$*) such that the system* S *is solvable if and only if there exists a shift that sparsifies the polynomial* P_S *by a monomial. Furthermore, the size of the polynomial instance* P_S *is polynomially related to the input size of the system* S *of polynomial equations. This holds true in both the sparse-representation and circuit-representation.*

We thus get that if $\mathsf{SparseShift}_R$ can be solved efficiently (in general) then HN_R can also be solved efficiently. In other words, $\mathsf{SparseShift}_R$ is at least as hard as HN_R. This completes the proof of Theorem 1. Putting Theorem 1 together with the fact that $\mathsf{HN}_{\mathbb{Z}}$ is undecidable (due to [22]).

4 Undecidability of β-gap-$\mathsf{SparseShift}_{\mathbb{Z}}$ problem

Note that $\mathsf{SparseShift}_R$ can be rephrased as the following gap problem – given a polynomial of sparsity σ, decide if there is a shift that sparsifies the polynomial to at most $\sigma - 1$ monomials, or there is no shift that sparsifies the polynomial below σ monomials. We shall now show a reduction from this ($\mathsf{SparseShift}_R$ problem) to β-gap-$\mathsf{SparseShift}_R$ for any $\beta > 1$.

Let the sets X'' and W be as defined in the construction of the polynomial P_S in Section 3.2. Let d be a parameter that we shall soon fix. Let $X^{(1)}, \ldots, X^{(d)}$ and $W^{(1)}, \ldots, W^{(d)}$ be d many disjoint copies of variable sets X'' and W respectively. Let $X_d = \sqcup_{k=1}^{d} X^{(i)}$ and $W_d = \sqcup_{k=1}^{d} W^{(i)}$. For the sake of brevity, let us use the following notation: Let $Y = X'' \sqcup W$. For all $k \in [d]$, let $Y^{(k)} = X^{(k)} \sqcup W^{(k)}$ and $\left|Y^{(k)}\right| = N'$. Let $Y_d = \sqcup_{k=1}^{d} Y^{(i)}$, so that $|Y_d| = N'd$. Let the polynomial $Q_d(Y_d)$ be defined as follows.

$$Q_d(Y_d) = \prod_{k=1}^{d} P_S(Y^{(k)}). \tag{2}$$

Observe that the sparsity of $Q_d(Y_d)$ is given by the product of sparsities of d many instances of $P_S(X'', W)$.

▶ **Lemma 13.** *Let $P_S(X'', W)$ and $Q_d(Y_d)$ be the polynomials as defined above. Let σ be equal to the sparsity of the polynomial P_S. Then,*

1. $\deg(Q_d) \leq 3d$.
2. *$Q_d(Y_d)$ has sparsity equal to σ^d.*
3. *If P_S has a depth four circuit of size $s \leq 3t + N'$ (recall Remark 9) then Q_d has a depth five circuit of size $sd + 1$.*
4. *There is a vector $\mathbf{a} \in R^{N'}$ such that $P_S(Y + \mathbf{a})$ has at most $\sigma - 1$ monomials if and only if there exists a vector $\mathbf{a}_d \in R^{N' \cdot d}$ such that $Q_d(Y_d + \mathbf{a}_d)$ has at most $(\sigma - 1)^d$ monomials.*
5. *For all vectors $\mathbf{a} \in R^{N'}$, $P_S(Y + \mathbf{a})$ has at least σ monomials if and only if for all vectors $\mathbf{a}_d \in R^{N' \cdot d}$ $Q_d(Y_d + \mathbf{a}_d)$ has at least σ^d monomials.*

Proof. The claim regarding the degree of Q_d follows immediately from the fact that $\deg(P_S) \leq 3$. Given that $P_S(X'', W)$ has a sparsity of σ and since $Q_d(Y_d)$ is defined to be a product of d distinct copies of $P_S(X'', W)$, sparsity of $Q_d(Y_d)$ is equal to σ^d. Similarly, if P_S can be computed by a circuit of size s, then there is a depth five circuit of size $(sd + 1)$ that computes the polynomial $Q_d(Y_d)$ – its output node is a product node into which d copies of circuits of $P_S(X'', W)$ feed into.

If there is a vector $\mathbf{a} \in R^{N'}$ such that $P_S(Y + \mathbf{a})$ has at most $\sigma - 1$ monomials then by taking $\mathbf{a}_d$ to be the concatenation of $\mathbf{a}$, d many times, we get that $Q_d(Y_d + \mathbf{a}_d)$ has at most $(\sigma - 1)^d$ monomials. If there is $\mathbf{a}_d \in R^{N' \cdot d}$ such that $Q_d(Y_d + \mathbf{a}_d)$ has at most $(\sigma - 1)^d$ monomials then it cannot happen that there is no $\mathbf{a} \in R^{N'}$ such that $P_S(Y + \mathbf{a})$ has at most $\sigma - 1$ monomials.

If for all vectors $\mathbf{a} \in R^{N'}$, $P_S(Y + \mathbf{a})$ has at least σ monomials, then $Q_d(Y_d + \mathbf{a}_d)$ must have at least σ^d monomials for all $\mathbf{a}_d \in R^{N' \cdot d}$. On the other hand if for all vectors $\mathbf{a}_d \in R^{N' \cdot d}$, $Q_d(Y_d + \mathbf{a}_d)$ has at least σ^d monomials, (for the sake of contradiction) let us suppose that there is a vector $\mathbf{a}' \in R^{N'}$ such that $P_S(Y + \mathbf{a}')$ has at most $\sigma - 1$ monomials. As before (due to the product structure of Q_d) we get that there is a corresponding vector $\mathbf{a}'_d \in R^{N' \cdot d}$ such that $Q_d(Y_d + \mathbf{a}'_d)$ has at most $(\sigma - 1)^d$ monomials. This contradicts our assumption. Thus, for all $\mathbf{a} \in R^{N'}$, $P_S(Y + \mathbf{a})$ has at least σ monomials. ◀

▶ **Theorem 14.** *Let R be an integral domain but not a field. Given a system S of polynomial equations over the polynomial ring in n variables $R[X]$, for any function $\beta : \mathbb{N} \to \mathbb{R}_+$ such that $\beta \in o(1)$, there exist $d = d(S, \beta) \in \mathbb{Z}_{>0}$ and a polynomial $Q_d(Y_d)$, in $N'd$ variables, of degree at most $3d$, such that the system S is solvable if and only if the M^β-gap-$\mathsf{SparseShift}_R$ problem for $Q_d(Y_d)$ is solvable, where M is the representation length of $Q_d(Y_d)$ in the sparse representation.*[6]

Proof. Given a system S of polynomial equations, we can construct the polynomial $P_S(X'', W)$ (as defined in Equation (1)). Let σ be the sparsity of $P_S(X'', W)$. Recall from Theorem 12 that system S has a solution if and only if there exists a shift that sparsifies the polynomial $P_S(X'', W)$ by a monomial.

Recall that $Q_d(Y_d)$ has $M' = \sigma^d$ many monomials. Let $\alpha = \frac{\sigma}{\sigma - 1}$. By putting together Theorem 12 and Lemma 13, we get that the system S of polynomial equations is solvable if and only if the α^d-gap problem for $Q_d(Y_d)$ is solvable. Calculating we get that $\alpha^d = \left(\frac{\sigma}{\sigma-1}\right)^d \approx e^{d/(\sigma-1)} = M'^{\frac{1}{(\sigma-1)\log\sigma}}$. Picking d large enough so that $M' \geq \sqrt{M}$ and $\beta(M) < \frac{1}{2(\sigma-1)\log\sigma}$ the claim follows. ◀

[6] Recall that in sparse representation polynomials are given as a set of pairs consisting of exponent vectors together with the coefficient of the corresponding monomial. Thus, $M \approx (\#\text{ monomials in } Q_d) \times ((N'd) \times O(\log d)) \times O(d \cdot b)$, where b is the maximal bit complexity of a coefficient in $P_S(X'', W)$.

Putting Theorem 14 together with the fact that $\mathsf{HN}_{\mathbb{Z}}$ is undecidable (due to [22]) we get Theorem 4.

5 Hardness of β-gap-SparseShift$_R$ problem for $R = \mathbb{F}_p, \mathbb{Q}, \mathbb{R}$, or $\mathbb{Z}_q$

In this section we prove Theorems 5 and 6 by giving a reduction from Max-3Lin$_R$ to the α-gap-SparseShift$_R$ problem, for different domains $R = \mathbb{F}_p, \mathbb{Q}, \mathbb{R}$ or $\mathbb{Z}_q$. Observe that we now do not require that our ring R is not a field.

Let $X = \{x_1, \ldots, x_n\}$. Let the given system S of linear equations be $\{L_1(X) = 0, \ldots, L_m(X) = 0\}$, where $L_i(X) \in R[X]$, and each equation $L_i(X) = 0$ depends on exactly 3 variables. Let the given system of equations be expressed together as $A \cdot X + \mathbf{b} = \mathbf{0}$ such that for all $i \in [m]$, $A_i \cdot X + b_i = 0$ is the i'th linear equation $L_i(X) = 0$, where A_i is the i'th row of the matrix A. Note that there are exactly three non-zero entries in each row of A. Let $w = \max\{2n, 2m\}$. Let C be a $w \times w$ matrix such that

$$\text{for all } 1 \le i, j \le w, \quad C_{i,j} = \begin{cases} A_{i,j-w+n} & \text{if } i \le m \text{ and } j \ge w - n + 1; \\ 0 & \text{otherwise.} \end{cases}$$

In other words, A is the top right block of C and the rest of C is zeros. Let $\mathbf{e} = (e_1, \ldots, e_w) \in R^w$ be such that $e_i = b_i$ for all $1 \le i \le m$ and $e_i = 0$ otherwise. Let e_0 be some constant. Let $Y = \{y_1, \ldots, y_w\}$ be a new set of variables disjoint from X. Let the polynomial $Q_S(Y) \in R[Y]$ be defined as follows.

$$Q_S(Y) = \sum_{i,j \in [w]} C_{i,j} \cdot y_i y_j + \sum_{i \in [w]} e_i \cdot y_i + e_0 \,.$$

Note that there are at most $3m$ many non-zero entries in C, and there are at most m many non-constant linear terms. Thus the sparsity of this polynomial is at most $4m + 1$.

For some vector $\mathbf{a} = (a_1, \ldots, a_w) \in R^w$ let us examine the structure of the polynomial $Q_S(Y + \mathbf{a})$.

$$\begin{aligned} Q_S(Y + \mathbf{a}) &= \sum_{i,j \in [w]} C_{i,j} \cdot (y_i + a_i)(y_j + a_j) + \sum_{i \in [w]} e_i \cdot (y_i + a_i) + e_0 \\ &= \sum_{i,j \in [w]} C_{i,j} \cdot (y_i y_j + a_i y_j + a_j y_i + a_i a_j) + \sum_{i \in [w]} e_i \cdot (y_i + a_i) + e_0 \\ &= \sum_{i,j \in [w]} C_{i,j} \cdot (y_i y_j + a_i a_j) + \sum_{i,j \in [w]} a_i \cdot y_j \cdot C_{i,j} + \sum_{i,j \in [w]} a_j \cdot y_i \cdot C_{i,j} + \sum_{i \in [w]} e_i \cdot (y_i + a_i) + e_0 \\ &= \sum_{i,j \in [w]} C_{i,j} \cdot y_i y_j + \sum_{i \in [w]} y_i \cdot (e_i + \sum_{j \in [w]} a_j \cdot (C_{i,j} + C_{j,i})) + \sum_{i,j \in [w]} C_{i,j} \cdot a_i a_j + \sum_{i \in [w]} a_i \cdot e_i + e_0 \,. \end{aligned}$$

Observe that the quadratic part of the polynomial $Q_S(Y)$ remains unperturbed under the shift but the affine linear part of it could get perturbed. We shall now show that every non-zero coefficient in the linear part corresponds to a linear equation in S.

▶ **Lemma 15.** *Let $\mathbf{a} = (a_1, \ldots, a_w) \in R^w$. Let $\mathbf{a}' \in R^n$ be the projection of $\mathbf{a}$ down to its last n elements, that is, for all $j \in [n]$, $a'_j = a_{j+w-n}$. Then, for all $i \in [m]$, the coefficient of y_i in $Q_S(Y + \mathbf{a})$ is zero if and only if $\mathbf{a}'$ satisfies the i'th linear equation $L_i(X) = 0$. Moreover, for all $i > m$, the coefficient of y_i is zero.*

Proof. For all $i \in [m]$, the coefficient of y_i in the polynomial $Q_S(Y + \mathbf{a})$ is equal to

$$e_i + \sum_{j \in [w]} a_j (C_{i,j} + C_{j,i}) \,.$$

Note that for this regime of $i \in [m]$, $e_i = b_i$, $C_{j,i} = 0$, and $C_{i,j} = A_{i,j-w+n}$ for j in $[w-n+1, w]$ and zero otherwise. Thus, the coefficient of such a y_i in $Q_S(Y + \mathbf{a})$ reduces as follows.

$$e_i + \sum_{j\in[w]} a_j(C_{i,j} + C_{j,i}) = b_i + \sum_{j'=1}^{n} a_{w-n+j'} A_{i,j'} = b_i + \sum_{j'=1}^{n} a'_{j'} A_{i,j'}.$$

This is exactly the value obtained by evaluating the i'th linear polynomial L_i at $\mathbf{a}'$. Thus, we get that coefficient of y_i (for $i \in [m]$) is zero if and only if $\mathbf{a}'$ satisfies the i'th linear equation, i.e., $L_i(\mathbf{a}') = 0$.

For all $i > m$, $e_i = 0$ and $C_{i,j} = 0$. Further, $\mathbf{a}_j = 0$ for all $j \leq w - n$. Hence,

$$e_i + \sum_{j\in[w]} a_j(C_{i,j} + C_{j,i}) = \sum_{j\in[w]} a_j \cdot C_{j,i} = \sum_{j=w-n+1}^{w} a_j \cdot C_{j,i} = 0\,.$$

The last equality in the math block above is due to the fact that the entries $C_{j,i}$ are equal to zero for $j \geq w - n + 1$ and $i \geq m + 1$ regardless of whether $n \geq m$ or $m \geq n$, from the construction of the matrix C. Thus the coefficients of the terms y_i for all $i > m$ are zero. ◀

Using this correspondence, we can show the following reduction.

▶ **Lemma 16.** *Let $\mathbf{a} = (a_1, \ldots, a_w) \in R^w$. Let $\mathbf{a}' \in R^n$ be the projection of $\mathbf{a}$ down to its last n elements, that is, for all $j \in [n]$, $a'_j = a_{j+w-n}$. Then*

1. *$\mathbf{a}'$ satisfies at most δ fraction of equations in S if and only if $Q_S(Y + \mathbf{a})$ has at least $(4-\delta)m$ non-constant monomials, and*
2. *$\mathbf{a}'$ satisfies at least $1-\varepsilon$ fraction of equations in S if and only if $Q_S(Y + \mathbf{a})$ has at most $(3+\varepsilon)m + 1$ monomials.*

Proof. From the aforementioned discussion, the sparsity of the polynomial $Q_S(Y + \mathbf{a})$ is decided by the coefficients of the linear terms. Further, Lemma 15 characterizes that the coefficient of a linear term is zero if and only if the *corresponding* linear polynomial equation is *satisfied*. Thus at most δ fraction of equations in S are satisfied if and only if at most δ fraction of coefficients of linear terms are equal to zero. In other words, if at least $(1-\delta)$ fraction of coefficients of linear terms are non-zero. The constant term e_0 could get cancelled out, that is, $\sum_{i,j\in[w]} C_{i,j} \cdot a_i a_j + \sum_{i\in[w]} a_i \cdot e_i + e_0$ could be zero. Thus, $\mathbf{a}'$ satisfies at most δ fraction of equations in S if and only if $Q_S(Y + \mathbf{a})$ has at least $3m + (1-\delta)m = (4-\delta)m$ non-constant monomials.

Similarly at least $1-\varepsilon$ fraction of equations in S are satisfied if and only if at least $1-\varepsilon$ fraction of coefficients of linear terms are equal to zero. In other words, if at most ε fraction of coefficients of linear terms are non-zero. Thus, $\mathbf{a}'$ satisfies at least $1-\varepsilon$ fraction of equations in S if and only if $Q_S(Y + \mathbf{a})$ has at most $(3+\varepsilon)m + 1$ monomials. ◀

We first prove a more restricted version of Theorems 5 and 6 that shows hardness of α-approximate $\mathsf{SparseShift}_R$ for some small α. Then we shall amplify this hardness for any $\beta > 1$.

▶ **Theorem 17.** *Let $R = \mathbb{F}_p, \mathbb{R}, \mathbb{Q}$ or $\mathbb{Z}_q$. For all $\varepsilon, \delta > 0$ as given in the last column of Table 1, there exists an $\alpha = \alpha(\varepsilon, \delta, R)$ such that it is* NP*-hard to α-approximate $\mathsf{SparseShift}_R$, in either the sparse, dense or arithmetic circuit representation.*

Proof. We first note that as the input is a degree 2 polynomial, all three representations are polynomially equivalent.

Suppose for a regime of values of $\varepsilon', \delta' > 0$ we are guaranteed the following. Given an instance of Max-3Lin$_R$, it is NP-hard to distinguish the following cases – if there is a assignment that satisfies at least $(1-\varepsilon')$ fraction of linear equations or for all assignments at most δ' fraction of linear equations are satisfied. Putting this together with Lemma 16, we get that it is NP-hard to distinguish if there is a vector $\mathbf{a}$ such that the polynomial $Q_S(Y+\mathbf{a})$ has at most $t = 3m+\varepsilon' m+1$ monomials or if for all $\mathbf{a}$, the polynomial $Q_S(Y+\mathbf{a})$ has at least $\alpha t = (4-\delta')m$ non-constant monomials. Thus, we get that it is NP-hard to α-approximate SparseShift$_R$ where $\alpha = \alpha(\varepsilon', \delta')$ is obtained as follows.

$$\alpha = \frac{4-\delta'}{3+\varepsilon'+\frac{1}{m}} = \frac{4}{3} - \frac{4\varepsilon' + 3\delta' + o(1)}{9 + 3\varepsilon' + o(1)}.$$

Each row of Table 1 gives us a guarantee of the form that we assumed at the beginning of this proof. Thus by iterating through the rows of Table 1, we get our parameter $\alpha = \alpha(\varepsilon, \delta, R)$ for various settings of R. This completes the proof. ◀

Let d be a parameter that we shall soon fix. Let $Y^{(1)}, \ldots, Y^{(d)}$ be d many disjoint copies of the variable set $Y = \{y_1, \ldots, y_w\}$. Let $Y_d = \sqcup_{k=1}^{d} Y^{(i)}$. Let the polynomial $F_{n,d}(Y_d)$ be defined as follows.

$$F_{n,d}(Y_d) = \prod_{k=1}^{d} Q_S(Y^{(k)}). \tag{3}$$

Observe that the sparsity of $F_{n,d}(Y_d)$ is given by the product of sparsities of d many instances of $Q_S(Y)$. Further, if the polynomial $Q_S(Y)$ is computed by a circuit of size s then the polynomial $F_{n,d}(Y_d)$ has a circuit of size at most $sd+1$.

▶ **Lemma 18.** *Let S be a system of linear equations, and $F_{n,d}(Y_d)$ be the polynomial as defined above.*

1. *All vectors $\mathbf{a} \in R^n$ satisfy at most δ fraction of equations in S if and only if all vectors $\mathbf{b}_d \in R^{wd}$ are such that $F_{n,d}(Y_d + \mathbf{b}_d)$ has at least $((4-\delta)m)^d$ non-constant monomials.*
2. *There exists a vector $\mathbf{a} \in R^n$ such that it satisfies at least $1-\varepsilon$ fraction of equations in S if and only if there exists a vector $\mathbf{b}_d \in R^{wd}$ such that $F_{n,d}(Y_d + \mathbf{b}_d)$ has at most $((3+\varepsilon)m+1)^d$ monomials.*

Proof. From the product structure of $F_{n,d}(Y_d)$, we get that for all vectors $\mathbf{b}_d \in R^{wd}$, the polynomial $F_{n,d}(Y_d + \mathbf{b}_d)$ has at least $((4-\delta)m)^d$ non-constant monomials if and only if for all vectors $\mathbf{a} \in R^n$, the polynomial $Q_S(Y+\mathbf{a})$ has at least $(4-\delta)m$ non-constant monomials. For the sake of contradiction, let us suppose that there is a vector $\mathbf{b}'_d \in R^{wd}$ such that $F_{n,d}(Y_d + \mathbf{b}'_d)$ has at most $((4-\delta)m)^d - 1$ monomials. Because of the product structure of $F_{n,d}(Y_d)$, it must the case that there is a copy of $Q_S(Y)$, say $Q_S(Y^{(i)})$ such that $Q_S(Y^{(i)} + \mathbf{b}'_d|_{Y^{(i)}})$ has at most $(4-\delta)m - 1$ non-constant monomials which contradicts our assumption. The other direction also follows trivially from the product structure. From Lemma 16, we get that $\mathbf{a}'$ satisfies at most δ fraction of equations in S if and only if $Q_S(Y+\mathbf{a})$ has at least $(4-\delta)m$ non-constant monomials. By putting both of these together, we get Item 1.

By invoking Lemma 16 again, we get that there is a vector $\mathbf{a} \in R^n$ such that it satisfies at least $1-\varepsilon$ fraction of equations in S if and only if there is a vector $\mathbf{b}$ such that $Q_S(Y+\mathbf{b})$ has at most $(3+\varepsilon)m+1$ monomials. By taking $\mathbf{b}_d$ to be the concatenation of $\mathbf{b}$, d many times,

we get that $F_{n,d}(Y_d + \mathbf{b}_d)$ has at most $((3+\varepsilon)m+1)^d$ monomials. On the other hand, if there exists a vector $\mathbf{b}_d \in R^{wd}$ such that $F_{n,d}(Y_d + \mathbf{b}_d)$ has at most $((3+\varepsilon)m+1)^d$ monomials then because of the product structure of $F_{n,d}(Y_d)$, $Q_S(Y + \mathbf{a})$ has at most $(3+\varepsilon)m+1$ monomials where $\mathbf{a} = \mathbf{b}_d|_{Y^{(1)}}$. This completes the proof of Item 2. ◀

Proof of Theorems 5 and 6. Given any $\beta > 1$ and α as given by Theorem 17, let $d = \log_\alpha \beta$. Thus, $\beta = \alpha^d$. Let $F_{n,d}(Y_d)$ be the polynomial as defined in Equation (3). From Lemma 18, we get that α-gap-$\mathsf{SparseShift}_R$ gap reduces to α^d-gap-$\mathsf{SparseShift}_R$.

To prove Theorem 5 we note that if $Q_S(Y)$ is provided in sparse representation (recall that sparsity of $Q_S(Y)$, denoted by t, is at most $4m+1$) and if α^d-gap-$\mathsf{SparseShift}_R$ problem for $F_{n,d}(Y_d)$ can be solved efficiently in time $N^{O(1)}$ (where $N = t^d$ is the sparsity of the polynomial $F_{n,d}(Y_d)$) then α-gap-$\mathsf{SparseShift}_R$ problem for $Q_S(Y)$ can be solved in time $t^{O(d)}$. Thus, as long as $d = O(1)$ the gap reduction runs in polynomial time.

Similarly, to prove Theorem 6 we note that if $Q_S(Y)$ is provided as a circuit of size s and if α^d-gap-$\mathsf{SparseShift}_R$ problem for $F_{n,d}(Y_d)$ can be solved efficiently in time $N^{O(1)}$ (where $N = sd+1$ is the input size of the instance provided as a circuit) then α-gap-$\mathsf{SparseShift}_R$ problem $Q_S(Y)$ can be solved in time $(sd)^{O(1)}$. As long as d is at most a polynomial in s, the gap reduction runs in polynomial time. ◀

References

1. Michael Ben-Or and Prasoon Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 301–309. ACM, 1988. doi:10.1145/62212.62241.
2. Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer, 1998. URL: https://link.springer.com/book/10.1007/978-1-4612-0701-6.
3. Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: NP- completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21(1):1–46, 1989. doi:bams/1183555121.
4. Allan Borodin and Prasoon Tiwari. On the decidability of sparse univariate polynomial interpolation. *Comput. Complex.*, 1:67–90, 1991. doi:10.1007/BF01200058.
5. Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial derivatives in arithmetic complexity and beyond. *Foundations and Trends® in Theoretical Computer Science*, 6(1–2):1–138, 2011. doi:10.1561/0400000043.
6. Michael Clausen, Andreas W. M. Dress, Johannes Grabmeier, and Marek Karpinski. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theor. Comput. Sci.*, 84(2):151–164, 1991. doi:10.1016/0304-3975(91)90157-W.
7. Martin Davis. Hilbert's tenth problem is unsolvable. *The American Mathematical Monthly*, 80(3):233–269, 1973. doi:10.1080/00029890.1973.11993265.
8. Zeev Dvir, Rafael Mendes de Oliveira, and Amir Shpilka. Testing equivalence of polynomials under shifts. 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 417–428. Springer, 2014. doi:10.1007/978-3-662-43948-7_35.
9. Vitaly Feldman, Parikshit Gopalan, Subhash Khot, and Ashok Kumar Ponnuswami. New results for learning noisy parities and halfspaces. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 563–574. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.51.
10. Dima Grigoriev. Testing shift-equivalence of polynomials by deterministic, probabilistic and quantum machines. *Theor. Comput. Sci.*, 180(1-2):217–228, 1997. doi:10.1016/S0304-3975(96)00188-0.

11 Dima Grigoriev and Marek Karpinski. Algorithms for sparse rational interpolation. In Stephen M. Watt, editor, *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation, ISSAC '91, Bonn, Germany, July 15-17, 1991*, pages 7–13. ACM, 1991. doi:10.1145/120694.120696.

12 Dima Grigoriev and Marek Karpinski. A zero-test and an interpolation algorithm for the shifted sparse polynominals. In Gérard D. Cohen, Teo Mora, and Oscar Moreno, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 10th International Symposium, AAECC-10, San Juan de Puerto Rico, Puerto Rico, May 10-14, 1993, Proceedings*, volume 673 of *Lecture Notes in Computer Science*, pages 162–169. Springer, 1993. doi:10.1007/3-540-56686-4_41.

13 Dima Grigoriev, Marek Karpinski, and Michael F. Singer. Interpolation of sparse rational functions without knowing bounds on exponents. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 840–846. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89616.

14 Dima Grigoriev and Yagati N. Lakshman. Algorithms for computing sparse shifts for multivariate polynomials. *Appl. Algebra Eng. Commun. Comput.*, 11(1):43–67, 2000. doi:10.1007/s002000050004.

15 Venkatesan Guruswami and Prasad Raghavendra. Hardness of learning halfspaces with noise. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 543–552. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.33.

16 Venkatesan Guruswami and Prasad Raghavendra. A 3-query PCP over integers. 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 198–206. ACM, 2007. doi:10.1145/1250790.1250819.

17 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

18 Erich Kaltofen, Yagati N. Lakshman, and J.-M. Wiley. Modular rational sparse multivariate polynomial interpolation. In Shunro Watanabe and Morio Nagata, editors, *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC '90, Tokyo, Japan, August 20-24, 1990*, pages 135–139. ACM, 1990. doi:10.1145/96877.96912.

19 Neeraj Kayal. Affine projections of polynomials: extended abstract. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19–22, 2012*, pages 643–662. ACM, 2012. doi:10.1145/2213977.2214036.

20 Pascal Koiran. Hilbert's nullstellensatz is in the polynomial hierarchy. *J. Complex.*, 12(4):273–286, 1996. doi:10.1006/jcom.1996.0019.

21 Yagati N. Lakshman and B. David Saunders. Sparse polynomial interpolation in nonstandard bases. *SIAM J. Comput.*, 24(2):387–397, 1995. doi:10.1137/S0097539792237784.

22 Yuri V. Matiyasevich. The diophantineness of enumerable sets. *Dokl. Akad. Nauk SSSR*, 191(2):279–283, 1970. URL: http://mi.mathnet.ru/dan35274.

23 Dori Medini and Amir Shpilka. Hitting sets and reconstruction for dense orbits in vp_{e} and ΣΠΣ circuits. In Valentine Kabanets, editor, *36th Computational Complexity Conference, CCC 2021, July 20-23, 2021, Toronto, Ontario, Canada (Virtual Conference)*, volume 200 of *LIPIcs*, pages 19:1–19:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CCC.2021.19.

24 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity, version 9.0.3. Github survey, 2021. URL: https://github.com/dasarpmar/lowerbounds-survey/releases/tag/v9.0.3.

25 Shubhangi Saraf and Sergey Yekhanin. Noisy interpolation of sparse polynomials, and applications. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, USA, June 8-10, 2011*, pages 86–92. IEEE Computer Society, 2011. doi:10.1109/CCC.2011.38.

26 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010. doi:10.1561/0400000039.

27 Leslie G. Valiant. Completeness classes in algebra. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 – May 2, 1979, Atlanta, Georgia, USA*, pages 249–261. ACM, 1979. doi:10.1145/800135.804419.

A Reduction to a system of polynomial equations of degree at most 2

Case when input is provided in sparse representation

Let $S = \{f_1 = 0, \ldots, f_r = 0\}$ be our input system of polynomial equations such that each f_i is provided in sparse representation. Let s_i denote the number of monomials with non-zero coefficients in the polynomial f_i. Let Y and Z be new disjoint sets of variables, that are disjoint from X such that $Y = \{y_{j,k}^{(i)} \mid i \in [r], j \in [s_i] \text{ and } k \geq 1\}$ and $Z = \{z_j^{(i)} \mid i \in [r], j \in [s_i]\}$.

Let the variables in Y have a lexicographic ordering based on the indices i, j and k, variables in Z have a lexicographic ordering based on the indices i and j, and the variables in X have some arbitrary ordering. Across the sets X, Y and Z, let the ordering be $Z \succ Y \succ X$. Given $S = \{f_1 = 0, \ldots, f_r = 0\}$, we construct an extended system T of polynomial equations over the variables $X \sqcup Y \sqcup Z$ such that each polynomial equation is of degree at most 2 and is such that there is a solution $\mathbf{a} \in R^{|X|}$ for the system S if and only if there exists an extension $\mathbf{a}'$ of $\mathbf{a}$ such that $\mathbf{a}'$ is a solution for the system T. This is a well known reduction (see, e.g., Lemma 6 in Chapter 2 of [2]) but for completeness we repeat it here.

Algorithm 1 and Algorithm 2 describe the construction of the extended system of equations. What the algorithms do is, roughly, for any monomial $m = x_{i_1} \cdot x_{i_2} \cdot x_{i_3} \cdot \ldots \cdot x_{i_j}$ of degree greater than 2, introduce a new variable, say y, replace m with the the monomial $y \cdot x_{i_3} \cdot \ldots \cdot x_{i_j}$ and introduce a new equation $y - x_{i_1} \cdot x_{i_2} = 0$ and for any monomial m', of degree 1, introduce a new variable z and a new equation $z - m' = 0$. Finally, an affine linear equation of the form $\sum c_i z_i + c_0$ is added to account for the fact that the original sum of monomials has to be zero. In particular, at the termination of the algorithm, the system T consists of constant-free quadratic binomial[7] equations and affine linear polynomial equations. It is also clear that there exists $\mathbf{a}'$ that satisfies T if and only if there is $\mathbf{a}$ that satisfies S.

▶ **Lemma 19** (Lemma 7 restated). *Let $S = \{f_i(X) = 0\}_{i=1}^r$ be a system of polynomial equations over the polynomial ring $R[X]$ such that for each $i \in [r]$, $f_i(X)$ is a polynomial of degree d_i and sparsity s_i, and the bit complexity of each coefficient is at most L. Then, Algorithm 1 runs in time $\mathrm{poly}(|X|, r, \max_i\{d_i\}, \max_i\{s_i\}, L)$ and returns a set T of polynomial equations $\{g_j(X, Y, Z) = 0\}_{j=1}^t$ over the polynomial ring $R[X, Y, Z]$ such that*

- $t = \mathrm{poly}(|X|, r, \max_i\{d_i\}, \max_i\{s_i\}, L)$.
- $|X \sqcup Y \sqcup Z| = \mathrm{poly}(|X|, t, \max_i\{d_i\}, \max_i\{s_i\})$.
- *The bit-complexity of the coefficients of the polynomial equations in T is also at most L.*
- *Each polynomial equation $g_j(X, Y, Z) = 0$ in T is either a quadratic binomial polynomial equation or an affine linear polynomial equation.*
- *S has a solution $\mathbf{a} \in R^{|X|}$ if and only T has a solution $\mathbf{a}' \in R^{|X \sqcup Y \sqcup Z|}$.*

Proof. Note that by the aforementioned ordering of variables, in all quadratic binomial equations included into the set T that have the form $u - v \cdot w = 0$, we have that u is of the form $y_{j,k}^{(i)}$, and v and w could be of the form $y_{j,k'}^{(i)}$ or $x_{i'}$, and the term u is leading with respect

[7] We use the phrase constant-free quadratic binomial equation to refer to an equation with two non-constant monomials of degree at most 2.

Algorithm 1 ConstructExtendedSystem-SparseRepresentation(S).

Result: Given a system $S = \{f_1 = 0, \ldots, f_r = 0\}$ of polynomial equations provided in sparse representation, we construct a system T of polynomial equations over an extended set of variables such that each polynomial in T has degree at most 2 and if $\mathbf{a}$ solves S then there exists an extension $\mathbf{a}'$ of $\mathbf{a}$ such that $\mathbf{a}'$ solves T and vice-versa.

$T \leftarrow \emptyset$;
for $i \in [r]$ **do**
  $f' \leftarrow 0$;
  Let s_i be the sparsity of f_i;
  for $j \in [s_i]$ **do**
    Let $c_{i,j}$ be the coefficient of j'th monomial $m_{i,j}$ in f_i;
    if $\deg(m_{i,j}) \geq 1$ **then**
      $U \leftarrow \text{ReduceMonomial}(m_{i,j}, i, j)$;
      $T \leftarrow T \cup U$;
      $f' \leftarrow f' + c_{i,j} \cdot z_j^{(i)}$;
    else
      $f' \leftarrow f' + c_{i,j}$;
    end
  end
  $T \leftarrow T \cup \{f' = 0\}$;
end
return T

to the terms v and w. Further, all the quadratic equations in the set T can be assumed to have some sort of a *topological order*. Thus the values of all the Y variables that appear in the variable support can be inductively inferred by just setting the X variables. That is, if we want to satisfy all such equations, then the value of the term u can be inferred from the value of terms v and w for every invocation of u, v and w.

Further note that some of the linear polynomial equations in T take the form $u' - v' = 0$ (from Algorithm 2 of Algorithm 2) where u' is of the form $z_j^{(i)}$, and v' could be of the form $y_{j,k}^{(i)}$ or $x_{i'}$. Similar to the case above, the value of the term u' can be inferred from the value of the term v' for every invocation of u' and v', and the value of v' is already fixed as the values of all the X and Y variables that appear in the variable support were set in the aforementioned discussion. Observe that the rest of the linear polynomial equations in T (from Algorithm 1 of Algorithm 1) correspond to the polynomial equations in S, and setting of Z variables in the variable support of T, by the above procedure, satisfies all the linear polynomial equations in T.

Given an assignment $\mathbf{a}$ to X let $\mathbf{a}'$ be its unique extension to the variable set $X \sqcup Y \sqcup Z$ according to the process described above. The argument above shows that $\mathbf{a}$ is a solution to the system S if and only if $\mathbf{a}'$ is a solution to the system T. ◀

Case when input is provided in white-box circuit form

Let $S = \{f_i = 0\}_{i=1}^r$ be our input system of polynomial equations such that each f_i (for $i \in [r]$) is provided as an arithmetic circuit Φ_i of size s_i. Without loss of generality, for all $i \in [r]$, we can assume that every product gate in Φ_i has a fan-in of 2.

Algorithm 2 ReduceMonomial(m, i, j).

Result: Given the j'th monomial (of degree at least 1) of the i'th polynomial, $m_{i,j}$, generate a collection of polynomial equations U from it.

$U \leftarrow \emptyset$;
$k = 1$;
while $\deg(m) \geq 2$ **do**
  Let u, v be the renaming of the two trailing variables under the ordering of X and Y variables as described above;
  $U \leftarrow U \cup \{y^{(i)}_{j,k} - u \cdot v = 0\}$;
  $m \leftarrow \frac{m}{u \cdot v} \cdot y^{(i)}_{j,k}$;
  $k \leftarrow k + 1$;
end
$U \leftarrow U \cup \{z^{(i)}_j - m = 0\}$;
return U

Let $Y = \left\{ y^{(i)}_j \mid i \in [r] \text{ and } j \in [s_i] \right\}$ be a new set of variables disjoint from X. For $i \in [r]$, let $g^{(i)}_1, \ldots, g^{(i)}_{s_i}$ be a topologically sorted enumeration of all nodes in circuit Φ_i. For all $j \in [s_i]$, let node $g^{(i)}_j$ be labelled by the variable $y^{(i)}_j$. Corresponding to each node in Φ_i, we shall now define a polynomial equation of degree at most 2 over the variable sets X and Y.

Algorithm 3 describes the construction of the extended system of equations. For each input node $g^{(i)}_j$, labeled by u (where u is either a variable x_k or a constant c) in Φ_i, the algorithm adds the polynomial equation $y^{(i)}_j - u = 0$ to the system T. For each product node $g^{(i)}_j$ with children labelled u and v (where u, v could be of the form $y^{(i)}_{j'}$ for some $j > j'$ or $x_{k'}$), the algorithm introduces a new polynomial equation $y^{(i)}_j - u \cdot v = 0$ to the system T, and for each sum node $g^{(i)}_j$ with children labelled $u_1, \ldots, u_k$ (where $u_1, \ldots, u_k$ could be of the form $y^{(i)}_{j'}$ for some $j > j'$ or $x_{k'}$), it introduces a new polynomial equation $y^{(i)}_j - \sum_{i=1}^{k} u_i = 0$. At the termination of the algorithm, the system T consists of either constant-free quadratic binomial equations or affine linear polynomial equations. It is also clear that there exists $\mathbf{a}'$ that satisfies T if and only if there is $\mathbf{a}$ that satisfies S. As before this is easy to see: by following the flow of computation in an arithmetic circuit from leaves to the root, we can infer the values of $y^{(i)}_j$ for all $i \in [t]$ and $j \in [s_i]$. This discussion can be summarized as Lemma 20 below.

▶ **Lemma 20** (Lemma 8 restated). *Let $S = \{f_i(X) = 0\}_{i=1}^{r}$ be a system of polynomial equations over the polynomial ring $R[X]$ such that for each $i \in [r]$, the polynomial $f_i(X)$ is provided as an arithmetic circuit Φ_i of size s_i. Then, when given this as input, Algorithm 3 runs in time $\mathrm{poly}(|X|, r, \max_i\{s_i\})$ and returns a system T of polynomial equations $\{g_j(X, Y) = 0\}_{j=1}^{t}$ over the polynomial ring $R[X, Y]$ such that*

- $t = \mathrm{poly}(|X|, r, \max_i\{s_i\})$.
- $|Y| \leq r \cdot \max_i\{s_i\}$.
- *Each polynomial equation $g_j(X, Y) = 0$ in T is either a quadratic binomial polynomial equation or an affine linear polynomial equation.*
- *T has a solution in $R^{|X \sqcup Y|}$ if and only S has a solution in $R^{|X|}$.*

Algorithm 3 ConstructExtendedSystem-CircuitRepresentation(S).

Result: Given a system $S = \{f_1 = 0, \ldots, f_r = 0\}$ of polynomial equations provided in white-box circuit representation, we construct a system T of polynomial equations over an extended set of variables such that each polynomial in T has degree of at most 2 and if $\mathbf{a}$ solves S then there exists an extension $\mathbf{a}'$ of $\mathbf{a}$ such that $\mathbf{a}'$ solves T and vice-versa.

$T \leftarrow \emptyset$;
for $i \in [r]$ **do**
  Let Φ_i be the circuit computing f_i, and $s_i = |\Phi_i|$;
  for $j \in [s_i]$ **do**
    if $g_j^{(i)}$ *is a leaf node in* Φ_i **then**
      Let u be a variable or a constant labeling the input node in Φ_i;
      $T \leftarrow T \cup \{y_j^{(i)} - u = 0\}$;
    else
      if $g_j^{(i)}$ *is a product node in* Φ_i **then**
        Let u and v be the renaming of the labels of the children of $g_j^{(i)}$;
        $T \leftarrow T \cup \{y_j^{(i)} - u \cdot v = 0\}$;
      else
        $g_j^{(i)}$ is a sum node in Φ_i;
        Let $u_1, \ldots, u_k$ be the renaming of the labels of the children of $g_j^{(i)}$;
        $T \leftarrow T \cup \{y_j^{(i)} - \sum_{i=1}^{k} u_i = 0\}$;
      end
    end
  end
end
return T;

Online Paging with Heterogeneous Cache Slots

Marek Chrobak ✉
University of California at Riverside, CA, USA

Samuel Haney ✉
Tumult Labs, Durham, NC, USA

Mehraneh Liaee ✉
Northeastern University, Boston, MA, USA

Debmalya Panigrahi ✉
Duke University, Durham, NC, USA

Rajmohan Rajaraman ✉
Northeastern University, Boston, MA, USA

Ravi Sundaram ✉
Northeastern University, Boston, MA, USA

Neal E. Young ✉
University of California at Riverside, CA, USA

Abstract

It is natural to generalize the online k-Server problem by allowing each request to specify not only a point p, but also a subset S of servers that may serve it. To initiate a systematic study of this generalization, we focus on uniform and star metrics. For uniform metrics, the problem is equivalent to a generalization of Paging in which each request specifies not only a page p, but also a subset S of cache slots, and is satisfied by having a copy of p in some slot in S. We call this problem *Slot-Heterogenous Paging*.

In realistic settings only certain subsets of cache slots or servers would appear in requests. Therefore we parameterize the problem by specifying a family $\mathcal{S} \subseteq 2^{[k]}$ of requestable slot sets, and we establish bounds on the competitive ratio as a function of the cache size k and family $\mathcal{S}$. If all request sets are allowed ($\mathcal{S} = 2^{[k]}$), the optimal deterministic and randomized competitive ratios are exponentially worse than for standard Paging ($\mathcal{S} = \{[k]\}$). As a function of $|\mathcal{S}|$ and k, the optimal deterministic ratio is polynomial: at most $O(k^2|\mathcal{S}|)$ and at least $\Omega(\sqrt{|\mathcal{S}|})$. For any laminar family $\mathcal{S}$ of height h, the optimal ratios are $O(hk)$ (deterministic) and $O(h^2 \log k)$ (randomized). The special case that we call *All-or-One Paging* extends standard Paging by allowing each request to specify a specific slot to put the requested page in. For All-or-One Paging the optimal competitive ratios are $\Theta(k)$ (deterministic) and $\Theta(\log k)$ (randomized), while the offline problem is $\mathbb{NP}$-hard. We extend the deterministic upper bound to the *weighted* variant of All-or-One Paging (a generalization of standard Weighted Paging), showing that it is also $\Theta(k)$.

Some results for the laminar case are shown via a reduction to the generalization of Paging in which each request specifies a set P of *pages*, and is satisfied by fetching any page from P into the cache. The optimal ratios for the latter problem (with laminar family of height h) are at most hk (deterministic) and hH_k (randomized).

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Caching and paging algorithms, k-server, weighted paging, laminar family

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.23

Related Version *Full Version*: `https://arxiv.org/abs/2206.05579` [23]

Funding *Marek Chrobak*: Supported by NSF grants CCF-1536026 and CCF-2153723.
Samuel Haney: Supported by NSF grants CCF-1527084 and CCF-1535972.
Mehraneh Liaee: Supported by NSF grants CCF-1535929 and CCF-1909363.
Debmalya Panigrahi: Supported by NSF grants CCF-1527084, CCF-1535972, CCF-1750140, CCF-1955703, ARO grant W911NF2110230, and Indo-US Joint Center for Algorithms under Uncertainty.
Rajmohan Rajaraman: Supported by NSF grants CCF-1535929 and CCF-1909363.
Ravi Sundaram: Supported by NSF grants CCF-1535929 and IIS-2039945.
Neal E. Young: Supported by NSF grant CCF-1619463.

© Marek Chrobak, Samuel Haney, Mehraneh Liaee, Debmalya Panigrahi, Rajmohan Rajaraman, Ravi Sundaram, and Neal E. Young;
licensed under Creative Commons License CC-BY 4.0

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 23; pp. 23:1–23:24

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

1 Introduction

The standard k-Server and Paging models assume homogenous (interchangeable) servers and cache slots. They don't model applications where servers have different capabilities, nor the fact that modern cache systems often partition the slots, sometimes dynamically, with some parts exclusively accessible by specific processors, cores, processes, threads, or page sets (e.g., [27, 37, 44–46]).

This motivates us to generalize the online k-Server problem to allow each request to specify not only a point p, but also a subset S of servers that may serve it. We call this generalization *Heterogenous* k-Server. To date, only a few special cases of this problem have been studied [20, 41]. Here, following the strategy taken for other hard generalizations of k-Server [6, 7, 11, 21, 28, 36], we initiate a systematic study of this problem by focusing on its restriction to uniform and star metrics. For uniform metrics, the problem is equivalent to a variant of Paging in which each request specifies a page p and a subset S of k cache slots, to be satisfied by having a copy of p in some slot in S. We call this problem *Slot-Heterogenous Paging*. For star metrics the problem reduces to a weighted variant, where the cost of retrieving a page is the weight of the page. For reasons discussed below, we parameterize these problems by allowing the requestable sets S to be restricted to an arbitrary but pre-specified family $\mathcal{S} \subseteq 2^{[k]}$. (Restricting to $\mathcal{S} = \{[k]\}$ gives standard Paging and k-Server.) Next is a summary of our results, followed by a summary of related work.

Slot-Heterogenous Paging (Section 3). As we point out, Slot-Heterogenous Paging easily reduces (preserving the competitive ratio) to the Generalized k-Server problem in uniform metrics, for which upper bounds of $k2^k$ and $O(k^2 \log k)$ on the deterministic and randomized ratios are known [7, 11].

- We show that the optimal deterministic and randomized competitive ratios for Slot-Heterogenous Paging are at least $\Omega(2^k/\sqrt{k})$ and $\Omega(k)$, respectively (Theorems 1 (i) and 3).

Hence, the optimal ratios for Slot-Heterogenous Paging are exponentially worse than for standard Paging. The proofs of Theorems 1 and 3 employ some novel ideas that may be useful for other problems: the lower bound in Theorems 1 (i) uses an adversary argument that requires the construction of a set family not yet studied in the literature, while the proof of Theorem 3 is carried out via a reduction *from* standard Paging with a cache of size $\exp(\Theta(k))$.

The large competitive ratios in these lower bounds occur only for instances that use exponentially many distinct request sets S. And in realistic settings only certain subsets of cache slots or servers would appear in requests, namely those that represent capabilities or functionalities relevant in a given setting. This motivates us to study the optimal ratios as a function of the cache size k and the family $\mathcal{S}$ of requestable slot sets, and to try to identify natural families that admit more reasonable ratios.

- We show that the optimal deterministic ratio is at most $k^2|\mathcal{S}|$ for any family $\mathcal{S}$ (Theorem 5). Theorem 1 (ii) shows a complementary lower bound: for infinitely many families $\mathcal{S}$, every deterministic online algorithm has competitive ratio $\Omega(\sqrt{|\mathcal{S}|})$.

Together Theorems 5 and 1 (ii) imply that, as a function of $|\mathcal{S}|$ and k, the optimal deterministic ratio for Slot-Heterogenous Paging is polynomial.

Slot-Laminar Paging (Section 4). We then consider the specific structure of $\mathcal{S}$, showing better bounds when $\mathcal{S}$ is *laminar*. This case, which we call *Slot-Laminar Paging*, models applications where slot (or server) capabilities are hierarchical. Laminarity implies that $|\mathcal{S}| < 2k$, so (per Theorem 5 above) the optimal deterministic ratio is $O(k^3)$.

- We show that the optimal deterministic and randomized ratios for Slot-Laminar Paging are $O(h^2k)$ and $O(h^2 \log k)$, where $h \leq k$ is the height of $\mathcal{S}$ (Theorem 8). We next tighten the deterministic bound to $O(hk)$ (Theorem 10).

The proof of Theorem 8 is via a reduction to an intermediate problem that we call *Page-Laminar Paging* (introduced below), while the proof of Theorem 10 refines the generic algorithm from Theorem 5. The dependence on k in these bounds is asymptotically tight, as Slot-Laminar Paging generalizes standard Paging.

Page-Laminar Paging (Section 4.1). Page-Laminar Paging is a natural generalization of Paging in which each request is a set P of *pages* from an arbitrary but fixed laminar family $\mathcal{P}$, and is satisfiable by fetching any page from P into any slot in the cache.

- We show that the optimal deterministic and randomized ratios for Page-Laminar Paging are at most hk and hH_k, where h is the height of the laminar family and $H_k = \sum_{i=1}^{k} 1/i = \ln k + O(1)$ (Theorem 6).

The proof is by a reduction, which replaces each set request P by a request to one carefully chosen page in P, yielding an instance of Paging, while increasing the optimal cost by at most a factor of h.

Reducing Slot-Laminar Paging to Page-Laminar Paging. The reduction of Slot-Laminar Paging to Page-Laminar Paging in Theorem 8 is achieved via a relaxation of Slot-Laminar Paging that drops the constraint that each slot holds at most one page, while still enforcing the cache-capacity constraint of k. This relaxed instance is naturally equivalent to an instance of Page-Laminar Paging. The proof then shows how any solution for the relaxed instance can be "rounded" back to a solution for the original Slot-Laminar Paging instance, losing an $O(h)$ factor in the cost and competitive ratio.

Weighted All-Or-One Paging (Section 5). *All-or-One Paging* is the restriction of Slot-Laminar Paging (with height $h = 2$) to $\mathcal{S} = \{[k]\} \cup \{\{j\}\}_{j \in [k]}$. That is, only two types of requests are allowed: *general requests* (allowing the requested page to be anywhere in the cache), and *specific requests* (requiring the page to be in a specified slot). Specific requests don't give the algorithm any choice, so may appear easy to handle, but in fact make the problem substantially harder than standard Paging— recent independent work on All-or-One Paging [20] has shown that the optimal deterministic ratio is twice that of Paging, to within an additive constant. The optimal randomized ratio of All-or-One Paging is also at least twice that for Paging, as we show in the full paper [23]. Note that Theorem 8 upper bounds the optimal randomized ratio to within a constant factor of that for Paging. The full paper also has a proof that the offline problem is $\mathbb{NP}$-hard [23], in sharp contrast to even k-Server, which can be solved in polynomial time for arbitrary metrics.

We initiate a study of Heterogenous k-Server in non-uniform metrics through *Weighted All-Or-One Paging*, which extends All-or-One Paging so that each page has a non-negative weight and the cost of each retrieval is the weight of the page instead of 1.

- We show that the optimal deterministic ratio for Weighted All-Or-One Paging is $O(k)$, matching the ratio for standard Weighted Paging up to a small constant factor (Theorem 14).

The algorithm in the proof is implicitly a linear-programming primal-dual algorithm. With this approach, the crucial obstacle to overcome is that the standard linear program for standard Weighted Paging does not force pages into specific slots. Indeed, doing so makes the natural integer linear program an $\mathbb{NP}$-hard multicommodity-flow problem. (Section 5 has an example that illustrates the challenge.) We show how to augment the linear program to partially model the slot constraints.

Table 1 Summary of upper ($\leq$) and lower ($\geq$) bounds on optimal competitive ratios. Here $\mathsf{mass}(\mathcal{S}) = \sum_{S\in\mathcal{S}} |S|$ and $\mathcal{S}^* = \bigcup_{S\in\mathcal{S}} 2^S$. The lower bound for One-of-m Paging holds for some but not all m and k—see Theorem 1(ii). The upper bound for Slot-Laminar Paging in the deterministic case (Theorem 8) is in fact $2\cdot\mathsf{mass}(\mathcal{S}) - k$, which is at most $(2h-1)k$. Also, offline All-or-One Paging and its generalizations are $\mathbb{NP}$-hard [23], as is offline Page-Subset Paging ([21]).

Paging problem	set family	deterministic	randomized	where
Slot-Heterogeneous	$2^{[k]}\setminus\{\emptyset\}$	$\leq k2^k$	$\leq O(k^2\log k)$	via [7, 11]
– ”	arbitrary $\mathcal{S}$	$\leq k\min(\lvert\mathcal{S}^*\rvert, \mathsf{mass}(\mathcal{S}))$		Thm. 5
– One-of-m, $m\approx k/2$	$\binom{[k]}{m}$	$\geq \Omega(2^k/\sqrt{k})$	$\geq\Omega(k)$	Thms. 1(i), 3
– One-of-m, any m	$\binom{[k]}{m}$	$\gtrsim \Omega((4k/m)^{m/2}/\sqrt{m})$		Thm. 1(ii)
– Slot-Laminar	laminar $\mathcal{S}$, height h	$\leq (2h-1)k$	$\leq 3h^2H_k$	Thms. 8, 10
– All-or-One	$\{[k]\}\cup\{\{s\}:s\in[k]\}$	$\geq 2k-1$	$\geq 2H_k-1$	[20, 23, 32]
– ”	”	$\leq 2k+14$		[20]
Weighted All-or-One	$\{[k]\}\cup\{\{s\}:s\in[k]\}$	$\leq O(k)$		Thm. 14
Page-Subset	$\mathcal{P}=\binom{\text{all pages}}{m}$	$\geq\binom{k+m}{k}-1$		[28]
”		$\leq k(\binom{k+m}{k}-1)$	$\leq O(k^3\log m)$	[21]
– Page-Laminar	$\mathcal{P}$ laminar height h	$\leq hk$	$\leq hH_k$	Thm. 6

Related work. Paging and k-Server have played a central role in the theory of online computation since their introduction in the 1980s [12, 38, 43]. For k-Server, the optimal deterministic ratio is between k and $2k-1$ [35]. Recent work [26] offers hope for closing this gap, and substantial progress towards resolving the randomized case has been reported in [4, 16]. For Weighted Paging the optimal ratios are k and $\Theta(\log k)$ [1, 5, 29, 39, 43].

Restricted Caching is one previously studied model with *heterogenous* cache slots. It is the restriction of Slot-Heterogenous Paging in which each page p has one *fixed* set $S_p\subseteq[k]$ of slots, and each request to p requires p to be in some slot in S_p. For this problem the optimal randomized ratio is $O(\log^2 k)$ [18]. Better bounds are possible given further restrictions on the sets, as in *Companion Caching*, which models a cache partitioned into set-associative and fully-associative parts [14, 15, 30, 40]. It is natural to ask whether *Restricted k-Server*—the restriction of Heterogenous k-Server that requires each point p to be served by a server in a *fixed* set S_p— is easier than Heterogenous k-Server; while the two problems are different for many metric classes, they can be shown to be equivalent in metric spaces with no isolated points, such as Euclidean spaces. The $\mathbb{NP}$-hardness result for Restricted Caching from [15] implies that offline Slot-Heterogenous Paging with $\mathcal{S}=\{\{s,k\}:s\in[k-1]\}$ is $\mathbb{NP}$-hard.

Other sophisticated online caching models include *Snoopy Caching*—in which multiple processors each have their own cache and coordinate to maintain consistency across writes [34], *Multi-Level Caching*—where the cost to access a slot depends on the slot [24], and *Writeback-Aware Caching*—where each page has multiple copies, each with a distinct level and weight, and each request specifies a page and a level, and can be satisfied by fetching a copy of this page at the given or a higher level [8, 9]. (This is a special case of weighted Page-Laminar Paging where $\mathcal{P}$ consists of pairwise-disjoint chains.) *Multi-Core Caching* models the fact that faults can change the request sequence (e.g. [33]).

Patel's master thesis [41] studies Heterogenous k-Server with just two types of requests—general requests (to be served by any server) and specialized requests (to be served by any server in a fixed subset S' of "specialized" servers)— and bounds the optimal ratios for uniform metrics and the line. Recent independent work on deterministic algorithms for online All-or-One Paging establishes a $2k-1$ lower bound and a $2k+14$ upper bound [20]. Earlier work in [32] presents a $2k-1$ lower bound and a $3k$ upper bound on deterministic algorithms.

Heterogenous k-Server reduces (see Section 3) to the *Generalized k-Server* problem, in which each server moves in its own metric space, each request specifies one point in each space, and the request is satisfied by moving any one server to the requested point in its space [36]. For uniform metrics, the optimal competitive ratios for this problem are between 2^k and $k2^k$ (deterministic) and between $\Omega(k)$ and $O(k^2 \log k)$ (randomized) [7, 11]. These ratios are exponentially worse than the ratios for standard k-Server. Heterogenous k-Server, parameterized by $\mathcal{S}$, provides a spectrum of problems that bridges the two extremes.

Weighted k-Server is a restriction of Generalized k-Server in which servers move in the same metric space but have different weights, and the cost is the weighted distance [31]. For this problem (in non-uniform metrics) the deterministic and randomized ratios are at least (respectively) doubly exponential [6, 7] and exponential [3, 22].

Page-Subset Paging, restricted to m-element sets of pages, has been studied as (uniform) *Metrical Service Systems with Multiple Servers* [21, 28]. For this problem the deterministic ratio is at least $\binom{k+m}{k} - 1$ [28], while the randomized ratio is $O(k^3 \log m)$ [21]. The *k-Chasing* problem extends k-Server by having each request P be a convex subset of $\mathbb{R}^d$, to be satisfied by moving any server to any point in P [17]. For k-Chasing, no online algorithm is competitive even for $d = k = 2$ [17], while for $k = 1$ the ratios grow with d [2, 42].

In the *k-Taxi problem* each request (p, q) requires any server to move to p then (for free) to q. For this problem the optimal ratios are exponentially worse than for standard k-Server [19, 25].

2 Formal Definitions

Slot-Heterogenous Paging. A problem instance consists of a set $[k] = \{1, 2, \ldots, k\}$ of cache slots, a family $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$ of requestable slot sets, and a request sequence $\sigma = \{\sigma_t\}_{t=1}^{T}$, where each request has the form $\sigma_t = \langle p_t, S_t \rangle$ for some *page* p_t and set $S_t \in \mathcal{S}$. A *cache configuration* C is a function that specifies the content of each slot $s \in [k]$; this content is either a single page or empty. Configuration C is said to *satisfy* a request $\langle p, S \rangle$ if it assigns page p to at least one slot in S. A *solution* for a given request sequence σ is a sequence $\{C_t\}_{t=1}^{T}$ of cache configurations such that, for each $t \in [T]$, C_t satisfies request σ_t. The objective is to minimize the number of *retrievals*, where a page p is retrieved in slot s at time t if C_t assigns p to s, but C_{t-1} does not (or $t = 1$). An event when C_{t-1} does not assign p_t to any slot in S_t is called a *fault*. Obviously a fault triggers a retrieval but, while this is not strictly necessary, it is convenient to also allow an algorithm to make spontaneous retrievals, either by fetching into the cache a non-requested page or by moving pages within the cache.

Slot-Laminar Paging. This is the restriction of Slot-Heterogenous Paging to instances where $\mathcal{S}$ is *laminar*: every pair $R, R' \in \mathcal{S}$ of sets is either disjoint or nested. (This implies $|\mathcal{S}| \leq 2k$.) A laminar family $\mathcal{S}$ can be naturally represented by a forest (a collection of disjoint trees), with a set R being a descendant of R' if $R \subseteq R'$. When discussing Slot-Laminar Paging we will routinely use tree-related terminology; for example, we will refer to some sets in $\mathcal{S}$

as leaves, roots, or internal nodes. The height h of a laminar family $\mathcal{S}$ is one less than the maximum height of a tree in $\mathcal{S}$, that is the maximum h for which $\mathcal{S}$ contains a sequence of h strictly nested sets: $R_1 \subsetneq R_2 \subsetneq \ldots \subsetneq R_h$.

All-or-One Paging. This is the restriction of Slot-Laminar Paging to instances with $\mathcal{S} = \{[k]\} \cup \{\{j\}\}_{j\in[k]}$. That is, there are two types of requests: *general*, of the form $\langle p, [k]\rangle$, requiring page p to be in at least one slot of the cache, and *specific*, of the form $\langle p, \{j\}\rangle$, $j \in [k]$, requiring page p to be in slot j. For convenience, $\langle p, *\rangle$ is a synonym for $\langle p, [k]\rangle$, while $\langle p, j\rangle$ is a synonym for $\langle p, \{j\}\rangle$.

Weighted All-Or-One Paging. This is the natural extension of All-or-One Paging in which each page p is assigned a non-negative weight $\mathsf{wt}(p)$, and the cost of retrieving p is $\mathsf{wt}(p)$ instead of 1.

One-of-m Paging. This is the restriction of Slot-Heterogenous Paging to instances with $\mathcal{S} = \binom{[k]}{m} = \{S \subseteq [k] : |S| = m\}$, that is, every request specifies a set of m slots.

Page-Subset Paging. An instance consists of k cache slots, a collection $\mathcal{P}$ of requestable sets of pages, and a request sequence $\pi = \{P_t\}_{t=1}^{T}$, where each P_t is drawn from $\mathcal{P}$. A solution is a sequence $\{C_t\}_{t=1}^{T}$ of cache configurations (as previously defined) such that, at each time $t \in [T]$, C_t assigns at least one page in P_t to at least one slot. The objective is to minimize the number of retrievals. (Slots are interchangeable here, so a cache configuration could be defined as a multiset of at most k pages, but using slot assignments is technically more convenient.)

Page-Laminar Paging. This is the restriction of Page-Subset Paging to instances where $\mathcal{P}$ is *laminar*.

Generalized k-Server. In this variant of k-Server, each server moves in its own metric space; each request specifies one point in each space, and the request is satisfied by moving any one server to the requested point in its space [36].

Approximation algorithms. An algorithm $\mathbb{A}$ for a given cost minimization problem is called a *c-approximation algorithm* if, for each instance σ, $\mathbb{A}$ satisfies $\mathsf{cost}_{\mathbb{A}}(\sigma) \leq c \cdot \mathsf{opt}(\sigma) + b$, where $\mathsf{cost}_{\mathbb{A}}(\sigma)$ is the cost of $\mathbb{A}$ on σ, $\mathsf{opt}(\sigma)$ is the optimum cost of σ, and b is a constant independent of σ.

Online algorithms and competitive ratio. In the online variants of the paging problems studied in this paper the requests arrive online, one per time step, and an online algorithm needs to satisfy each request before the next one is revealed. To simplify presentation we assume that the algorithm knows the underlying set family $\mathcal{S}$ (or $\mathcal{P}$), but many of our algorithms work (or can be adapted to work) without knowing the set family in advance. An online algorithm $\mathbb{A}$ is called *c-competitive* if $\mathbb{A}$ is a c-approximation algorithm. As common in the literature, we will use the term "optimal deterministic (resp. randomized) competitive ratio" to refer to the optimal ratio of of a deterministic (resp. randomized) online algorithm for the given problem.

3 Slot-Heterogenous Paging

Any instance of Slot-Heterogenous Paging can be reduced to an instance of Generalized k-Server in uniform spaces, as follows. Represent each cache slot by a server in a uniform metric space whose points are the pages, then simulate each request $\langle p, S\rangle$ by a sufficiently long sequence of requests, each of which specifies point p for each server in S and alternates between two different points for the remaining servers, in $[k] \setminus S$. Composing this reduction with the upper bounds from [7] yields immediate upper bounds of $O(k2^k)$ and $O(k^3 \log k)$ on the deterministic and randomized ratios for unrestricted Slot-Heterogenous Paging (that is, with $\mathcal{S} = 2^{[k]} \setminus \{\emptyset\}$).

Theorems 1 (i) and 3 (Section 3.1) show that these bounds are tight within poly(k) factors: the optimal ratios are at least $\Omega(2^k/\sqrt{k})$ and $\Omega(k)$, respectively. But restricting $\mathcal{S}$ allows better ratios: Theorem 5 (Section 3.2) shows an upper bound of $k^2|\mathcal{S}|$ on the optimal deterministic ratio for any family $\mathcal{S}$. For One-of-m Paging, Theorems 5 and 1 (ii) (Section 3.1) imply that the optimal deterministic ratio is $O(k^{m+1})$ and $\Omega((4k/m)^{m/2}/\sqrt{m})$.

3.1 Lower bounds for Slot-Heterogenous Paging

We establish our lower bounds for Slot-Heterogenous Paging and One-of-m Paging given in Table 1.

▶ **Theorem 1.**

(i) *For all odd k, the optimal deterministic ratio for One-of-m Paging with $m = (k+1)/2$ is at least $\binom{k}{m} = \Omega(2^k/\sqrt{k})$. For all k, the optimal ratio with $m = \lfloor (k+1)/2 \rfloor$ is $\Omega(2^k/\sqrt{k})$.*

(ii) *For any even $m \geq 2$ and any $k > m$ that is an odd multiple of $m-1$, the optimal deterministic ratio for One-of-m Paging is at least $\binom{m-1}{m/2}\left(\frac{k}{m-1}\right)^{m/2} = \Theta((4k/m)^{m/2}/\sqrt{m}) = \Omega(\sqrt{|\mathcal{S}|})$, where $\mathcal{S} = \binom{[k]}{m}$.*

Before proving Theorem 1, we prove Lemma 2. It states that for any $\mathcal{S}$ the existence of a family $\mathcal{Z} \subseteq 2^{[k]}$ with certain properties implies a lower bound of $|\mathcal{Z}|$ on the competitive ratio. The proof of the theorem then constructs such families $\mathcal{Z}$ for appropriate families $\mathcal{S}$ of requestable sets. Throughout this section $\overline{X}$ denotes the complement of set $X \subseteq [k]$, that is $\overline{X} = [k] \setminus X$.

▶ **Lemma 2.** *For some $\mathcal{S} \subseteq 2^{[k]}$, suppose there are two set families $\mathcal{G} \subseteq \mathcal{S}$ and $\mathcal{Z} \subseteq 2^{[k]}$ such that*

(gz0) *For each $X \subseteq [k]$ there is $S \in \mathcal{G}$ such that $S \subseteq X$ or $S \subseteq \overline{X}$.*

(gz1) *If $Z \in \mathcal{Z}$ then $\overline{Z} \notin \mathcal{Z}$.*

(gz2) *For each $S \in \mathcal{G}$ there is $Y \in \mathcal{Z}$ such that $S \not\subseteq Z$ and $S \not\subseteq \overline{Z}$ for all $Z \in \mathcal{Z} \setminus \{Y\}$.*

Then the optimal deterministic ratio for Slot-Heterogenous Paging with family $\mathcal{S}$ is at least $|\mathcal{Z}|$.

Proof. The proof is an adversary argument based on the following idea. At each step, the adversary chooses a request that forces the algorithm to fault but causes at most two faults total among a fixed set of $2|\mathcal{Z}|$ other solutions. At the end, the algorithm's total cost is at least $|\mathcal{Z}|$ times the average cost of these other solutions, so its competitive ratio is at least $|\mathcal{Z}|$. This general approach is common for lower bounds on deterministic online algorithms (see e.g. lower bounds on the optimal ratios for k-Server [38], for Metrical Task Systems [13] and for Generalized k-Server on uniform metrics [36]).

Here are the details. Let $\mathbb{A}$ be any deterministic online algorithm for Slot-Heterogenous Paging with slot-set family $\mathcal{S}$. The adversary will request just two pages, p_0 and p_1. For a set $X \subseteq [k]$, let C_X denote the cache configuration where the slots in X contain p_0 and the slots in $\overline{X}$ contain p_1. Without loss of generality assume that each slot of $\mathbb{A}$'s cache always holds p_0 or p_1—its cache configuration is C_X for some X.

At each step, if the current configuration of $\mathbb{A}$ is C_X, the adversary chooses $S \in \mathcal{G}$ such that either $S \subseteq X$ or $S \subseteq \overline{X}$. (Such an S exists by Property (gz0).) If $S \subseteq X$, then all slots in S hold p_0, and the adversary requests $\langle p_1, S\rangle$, causing a fault. Otherwise, $S \subseteq \overline{X}$, so all slots in S hold p_1. In this case the adversary requests $\langle p_0, S\rangle$, causing a fault. The adversary repeats this K times, where K is arbitrarily large. Since $\mathbb{A}$ faults at each step, the overall cost of $\mathbb{A}$ is at least K.

It remains to bound the optimal cost. Let $\widetilde{\mathcal{Z}} = \{\overline{Z} : Z \in \mathcal{Z}\}$. By (gz1), we have $\widetilde{\mathcal{Z}} \cap \mathcal{Z} = \emptyset$. For each $Z \in \mathcal{Z} \cup \widetilde{\mathcal{Z}}$ define a solution called the *Z-strategy*, as follows. The solution starts in configuration C_Z. It stays in C_Z for the whole computation, except that on requests $\langle p_a, S\rangle$ that are not served by C_Z (that is, when all slots of C_Z in S contain p_{1-a}), it retrieves p_a to any slot $j \in S$, serves the request, then retrieves p_{1-a} back into slot j, restoring configuration C_Z. This costs 2.

We next observe that in each step at most one Z-strategy faults (and pays 2). Assume that the request at a given step is to p_0 (the case of a request to p_1 is symmetric). Let this request be $\langle p_0, S\rangle$, where $S \in \mathcal{G}$. Let $Y \subseteq [k]$ be the set from Property (gz2). For all $Z \in (\mathcal{Z} \cup \widetilde{\mathcal{Z}}) \setminus \{Y, \overline{Y}\}$, then, $S \cap Z \neq \emptyset$, implying that configuration C_Z has a slot in S that contains p_0—in other words, configuration C_Z satisfies S. Also, either $S \cap Y \neq \emptyset$ or $S \cap \overline{Y} \neq \emptyset$, so one of the configurations C_Y or $C_{\overline{Y}}$ also satisfies S. Therefore only one Z-strategy (Y or $\overline{Y}$) might not satisfy S. So, in each step, at most one Z-strategy faults (and pays 2).

Thus the combined total cost for all Z-strategies (not counting the cost of at most k for moving to Z at the beginning) is at most $2K$. There are $2|\mathcal{Z}|$ such strategies, so their average cost is at most $(2K+k)/2|\mathcal{Z}|$. The cost of $\mathbb{A}$ is at least K, so the ratio is at least $\frac{K}{(2K+k)/2|\mathcal{Z}|} = \frac{|\mathcal{Z}|}{1+k/2K}$. Taking K arbitrarily large, the lemma follows. ◀

Proof of Theorem 1. *Part (i).* Recall that $m = \lfloor (k+1)/2 \rfloor$. First consider the case when k is odd. Apply Lemma 2, taking both $\mathcal{G}$ and $\mathcal{Z}$ to be $\binom{[k]}{m}$. Properties (gz0) and (gz1) follow directly from k being odd and the definitions of $\mathcal{G}$ and $\mathcal{Z}$. Property (gz2) also holds with $Y = S$. (For any $S \in \mathcal{G}$, every $Z \in \mathcal{Z}$ satisfies $|Z| = |S| > |\overline{Z}|$, so $S \not\subseteq \overline{Z}$, while $S \subseteq Z$ implies $Z = S$.) Thus, by Lemma 2, the ratio is at least $|\mathcal{Z}| = \binom{k}{(k+1)/2} = \Omega(2^k/\sqrt{k})$. This proves Part (i) for odd k.

For even k, let $k' = k - 1$. Then apply Part (i) to k' using just cache slots in $[k']$, that is, using slot-set family $\mathcal{S}' = \binom{[k']}{m} \subseteq \binom{[k]}{m} = \mathcal{S}$, with slot k playing no role as it is never requested.

Part (ii). Fix such an m and k. Let $\ell = k/(m-1)$ so $\ell \geq 3$ is odd. Recall that $\mathcal{S} = \binom{[k]}{m}$ is the family of requestable slot sets. Partition $[k]$ arbitrarily into $m-1$ disjoint subsets $B^1, B^2, \ldots, B^{m-1}$, each of cardinality ℓ. For each B^e, order its slots arbitrarily as $B^e = \{b_1^e, b_2^e, \ldots, b_\ell^e\}$. For any index $c \in \{1, 2, \ldots, \ell\}$ and an integer i, let $c \oplus i$ denote $((c + i - 1) \bmod \ell) + 1$. In other words, we view each B^e as an odd-length cycle, and this cyclic structure is important in the proof. Any consecutive pair $\{b_c^e, b_{c\oplus 1}^e\}$ of slots on this cycle is called an *edge*. Thus each cycle B^e has ℓ edges.

First we define $\mathcal{G} \subseteq \mathcal{S}$ for Lemma 2. The sets S in $\mathcal{G}$ are those obtainable as follows: choose any $m/2$ edges, no two from the same cycle, then let S contain the m slots in those $m/2$ chosen edges. (The six slots inside the three dashed ovals in Figure 1 show one S in $\mathcal{G}$.) This set of $m/2$ edges uniquely determines S, and vice versa.

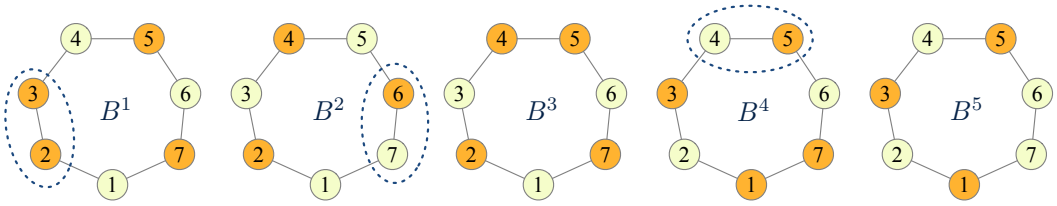


Figure 1 Illustration of the proof of Theorem 1 Part (ii) for $k = 35$, $m = 6$, and $\ell = 7$. The figure shows the partition of all slots into $m - 1 = 5$ sets $B^1, \ldots, B^5$, each represented by a cycle. To avoid clutter, each slot b^e_c is represented by its index c within B^e. The picture shows set $S = \{b^1_2, b^1_3, b^2_6, b^2_7, b^4_4, b^4_5\} \in \mathcal{G}$, marked by dashed ovals. It also shows $Z_{S'} \in \mathcal{Z}$, represented by orange/shaded circles, for $S' = \{b^1_2, b^1_3, b^3_4, b^3_5, b^4_7, b^4_1\}$.

We verify that $\mathcal{G}$ has Property (gz0) from Lemma 2. Indeed, consider any $X \subseteq [k]$. Call the slots in X *white* and the slots in $\overline{X}$ *black*. Each cycle B^e has odd length, so has an edge $\{b^e_c, b^e_{c\oplus 1}\}$ that is white (with two white slots) or black (with two black slots). So either (i) at least half the cycles have a white edge, or (ii) at least half have a black edge. Consider the first case (the other is symmetric). There are $m-1$ cycles, and m is even, so at least $m/2$ cycles have a white edge. So there are $m/2$ white edges with no two in the same cycle. The set S comprised of the m white slots from those edges is in $\mathcal{G}$, and is contained in X (because its slots are white). So $\mathcal{G}$ has Property (gz0).

Next we define $\mathcal{Z} \subseteq 2^{[k]}$ for Lemma 2. The set $\mathcal{Z}$ contains, for each set $S' \in \mathcal{G}$, one set $Z_{S'}$, defined as follows. For each of the $m/2$ cycles B^e having an edge $\{b^e_c, b^e_{c\oplus 1}\}$ in S', add to $Z_{S'}$ the two slots on that edge, together with the $(\ell-3)/2$ slots $b^e_{c\oplus 3}, b^e_{c\oplus 5}, \ldots, b^e_{c\oplus(\ell-2)}$. For each of the $m/2-1$ remaining cycles B^e, add to $Z_{S'}$ the $(\ell-1)/2$ slots $b^e_1, b^e_3, \ldots, b^e_{\ell-2}$. (The orange/shaded slots in Figure 1 show one set $Z_{S'}$ in $\mathcal{Z}$.) Then $Z_{S'}$ contains exactly $m/2$ edges (the ones in S') while its complement $\overline{Z}_{S'}$ contains exactly $m/2-1$ edges (one from each cycle with no edge in S'). This implies Property (gz1). Note that $Z_{S'} \neq Z_{S''}$ for different sets $S', S'' \in \mathcal{G}$.

Next we show Property (gz2). Given any set $S \in \mathcal{G}$, let $Y = Z_S \in \mathcal{Z}$. Consider any $Z_{S'} \in \mathcal{Z}$ such that $S \subseteq Z_{S'}$ or $S \subseteq \overline{Z}_{S'}$. We need to show $Z_{S'} = Z_S$, i.e., $S' = S$. It cannot be that $S \subseteq \overline{Z}_{S'}$, because S contains $m/2$ edges, whereas $\overline{Z}_{S'}$ contains $m/2-1$ edges. So $S \subseteq Z_{S'}$. But S and $Z_{S'}$ each contain exactly $m/2$ edges, which therefore must be the same. It follows from the definition of $Z_{S'}$ that $S' = S$. So Property (gz2) holds.

So $\mathcal{G}$ and $\mathcal{Z}$ have Properties (gz0)-(gz2) from Lemma 2. Directly from definition we have $|\mathcal{Z}| = |\mathcal{G}|$, while $|\mathcal{G}| = \binom{m-1}{m/2}\ell^{m/2}$ because there are $\binom{m-1}{m/2}$ ways to choose $m/2$ distinct cycles, and then for each of these $m/2$ cycles there are ℓ ways to choose one edge. Lemma 2 and $\ell = k/(m-1)$ imply that the optimal deterministic ratio is at least $f(m,k) = \binom{m-1}{m/2}(k/(m-1))^{m/2}$. To complete the proof of part (ii) we lower-bound $f(m,k)$. We observe that

$$4^m \;=\; \Omega(\sqrt{m}\,(k/(k-m))^{k-m+1/2}). \tag{1}$$

This can be verified by considering two cases: If $k \geq m+2$ then, using $1+z \leq \mathrm{e}^z$, we have $\sqrt{m}\,(k/(k-m))^{k-m+1/2} = \sqrt{m}(1+m/(k-m))^{k-m+1/2} \leq \sqrt{m}\cdot \mathrm{e}^{5m/4} \leq 2\cdot 4^m$, for all $m \geq 1$. In the remaining case, for $k = m+1$, we have $\sqrt{m}(k/(k-m))^{k-m+1/2} = \sqrt{m}(1+m)^{3/2} \leq 2\cdot 4^m$. Thus (1) indeed holds. Now, recalling that $f(m,k) = \binom{m-1}{m/2}(k/(m-1))^{m/2}$, we derive

$$\begin{aligned} f(m,k) \;&=\; \Theta\big((2^m/\sqrt{m})\cdot(k/(m-1))^{m/2}\big) && \text{(\textit{Stirling's approximation})} \\ &=\; \Theta\big((4k/m)^{m/2}\cdot(1+1/(m-1))^{m/2}/\sqrt{m}\big) && \text{(\textit{rewriting})} \\ &=\; \Theta\big((4k/m)^{m/2}/\sqrt{m}\big) && ((1+1/(m-1))^{m/2} \leq \mathrm{e}) \end{aligned} \tag{2}$$

input: Slot-Heterogenous Paging instance $(k, \mathcal{S}, \sigma = (\sigma_1, \ldots, \sigma_T))$
1. let the initial cache configuration C_0 be arbitrary; let $\ell \leftarrow 1$ — *ℓ is the start of the current phase*
2. for each time $t \leftarrow 1, 2, \ldots, T$:
2.1. if current configuration C_{t-1} satisfies request σ_t: ignore the request (set $C_t = C_{t-1}$)
2.2. else:
2.2.1. if any configuration satisfies all requests $\sigma_\ell, \sigma_{\ell+1}, \ldots, \sigma_t$: let C_t be any such configuration
2.2.2. else: let $\ell \leftarrow t$; let C_t be any configuration satisfying σ_t — *start the next phase*

Figure 2 Online algorithm ExhSearch for Slot-Heterogenous Paging.

This gives us one estimate on the competitive ratio in Theorem 1(ii). To obtain a second estimate, squaring both sides of (2), we obtain

$$\begin{aligned} f(m,k)^2 &= \Omega\big((4k/m)^m/m\big) = \Omega\big((k/m)^m \cdot 4^m/m\big) \\ &= \Omega\big((k/m)^m \cdot (k/(k-m))^{k-m+1/2}/\sqrt{m}\big) && \text{(using (1))} \\ &= \Omega(\tbinom{k}{m}) = \Omega(|\mathcal{S}|) && \text{(Stirling's approximation)} \end{aligned}$$

Therefore $f(m,k) = \Omega(\sqrt{|\mathcal{S}|})$, as claimed, completing the proof of Theorem 1(ii). ◀

Next we present a lower bound on the optimal competitive ratio for randomized algorithms:

▶ **Theorem 3.** *The optimal randomized ratio for One-of-m Paging with $m = \lfloor k/2 \rfloor$ is $\Omega(k)$.*

The proof is by a reduction from standard Paging with some N pages and a cache of size $N-1$. For any N, this problem has optimal randomized competitive ratio $H_{N-1} = \Theta(\log N)$ [29]. This and the next lemma, whose proof is deferred to the full paper [23], imply the theorem.

▶ **Lemma 4.** *Every $f(k)$-competitive (randomized) online algorithm $\mathbb{A}$ for One-of-m Paging with $m = \lfloor k/2 \rfloor$ can be converted into an $O(f(k))$-competitive (randomized) online algorithm $\mathbb{B}$ for standard Paging with N pages and a cache of size $N-1$, where $N = 2^{\Theta(k)}$.*

3.2 Upper bounds for deterministic Slot-Heterogenous Paging

This section gives upper bounds on the optimal deterministic competitive ratio for Slot-Heterogenous Paging with any slot-set family $\mathcal{S}$, as a function of $\mathsf{mass}(\mathcal{S}) = \sum_{S \in \mathcal{S}} |S| \le k|\mathcal{S}|$ and $|\mathcal{S}^*|$, where $\mathcal{S}^* = \bigcup_{S \in \mathcal{S}} 2^S$. The first bound follows from an easy counting argument. The second bound uses a refinement of the rank method of [7], which bounds the number of steps of a natural exhaustive-search algorithm by the rank of a certain upper-triangular matrix.

▶ **Theorem 5.** *Fix any $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$. The competitive ratio of Algorithm* ExhSearch *in Figure 2 for Slot-Heterogenous Paging with requestable sets from $\mathcal{S}$ is at most $k \cdot \min\{|\mathcal{S}^*|, \mathsf{mass}(\mathcal{S})\}$.*

The theorem implies that the competitive ratio of One-of-m Paging is polynomial in k when m is constant. Theorem 5 can be strengthened slightly by making the algorithm retrieve at most $\min(t,k)$ pages for the tth request in each phase.

Proof of Theorem 5. By inspection of the algorithm, the length of each phase does not depend on previous phases. Focus on any one phase. We first bound the length, say L, of the phase. To ease notation and without loss of generality, assume the phase is the first (with $\ell = 1$) and the algorithm faults in each step t, that is C_{t-1} does not satisfy $\sigma_t = \langle p_t, S_t\rangle$. (Otherwise first remove such requests; this doesn't change the algorithm's cost or increase the optimal cost.) So the following holds:

(UT) *For each time $t \in [L]$, configuration C_{t-1} satisfies requests $\sigma_1, \sigma_2, \ldots, \sigma_{t-1}$, but not σ_t.*

The final configuration C_L in the phase satisfies all requests in the phase. In particular, for each $S \in \mathcal{S}$, for each request $\langle p, S\rangle$ in the phase, C_L has p in some slot in S, so *(i) there are at most $|S|$ distinct requests in the phase that use any given set $S \in \mathcal{S}$.* Property (UT) implies that *(ii) every request σ_t in this phase is distinct* (indeed, for any $t' < t$, C_{t-1} satisfies $\sigma_{t'}$ but not σ_t). Observations (i) and (ii) imply the following bound $L \le \sum_{S\in\mathcal{S}} |S| = \mathsf{mass}(\mathcal{S})$.

(As an aside, the above argument uses only that every request in the phase is distinct, a weaker condition than (UT). Given only that property, the above bound on L is tight for every $\mathcal{S}$ in the following sense: consider any configuration C that puts a distinct page in each slot $s \in [k]$, and a request sequence σ that requests in any order every pair $\langle p, S\rangle$ such that $S \in \mathcal{S}$ and C assigns p to a slot in S. Then σ is satisfied by a single configuration, while having $\mathsf{mass}(\mathcal{S})$ distinct requests.)

Next we give a second bound on L that is tighter for some families $\mathcal{S}$. Identify each page p with a distinct but arbitrary real number. For each cache configuration C_t, let $C_t^i \in \mathbb{R}$ denote the page in slot i, if any, else 0. Define matrix $M \in \mathbb{R}^{L\times L}$ by

$$M_{st} = \prod_{i\in S_t} (C_{s-1}^i - p_t),$$

so that $M_{st} = 0$ if and only if $C_{s-1}^i = p_t$ for some $i \in S_t$, that is, if and only if C_{s-1} satisfies σ_t. So Property (UT) implies that M is upper-triangular and non-zero on the diagonal. So M has rank L.

Expanding the formula for M_{st}, we obtain

$$M_{st} = \sum_{S\subseteq S_t} \Big(\prod_{i\in S} C_{s-1}^i\Big)\cdot\Big(\prod_{i\in S_t\setminus S} -p_t\Big) = \sum_{S\subseteq S_t} \Big(\prod_{i\in S} C_{s-1}^i\Big)\cdot(-p_t)^{|S_t|-|S|} = \sum_{S\in\mathcal{S}^*} A_{sS}\cdot B_{St}\,,$$

where matrices $A \in \mathbb{R}^{L\times\mathcal{S}^*}$ and $B \in \mathbb{R}^{\mathcal{S}^*\times L}$ are defined by

$$A_{sS} = \prod_{i\in S} C_{s-1}^i \quad\text{and}\quad B_{St} = \begin{cases} (-p_t)^{|S_t|-|S|} & \text{if } S\subseteq S_t \\ 0 & \text{otherwise.}\end{cases}$$

That is, $M = AB$; A and B (and M) have rank at most $|\mathcal{S}^*|$. And M has rank L, so $L \le |\mathcal{S}^*|$. To bound the optimum cost, consider any phase other than the last. Let t' and t'' be the start and end times. Suppose for contradiction that the optimal solution incurs no cost (has no retrievals) during $[t'+1, t''+1]$. Then its configuration at time t' satisfies all requests in $[t', t''+1]$, contradicting the algorithm's condition for terminating the phase. So the optimal solution pays at least 1 per phase (other than the last). In any phase of length L the algorithm pays at most kL (at most k per step). This and two upper bounds on L imply Theorem 5. ◀

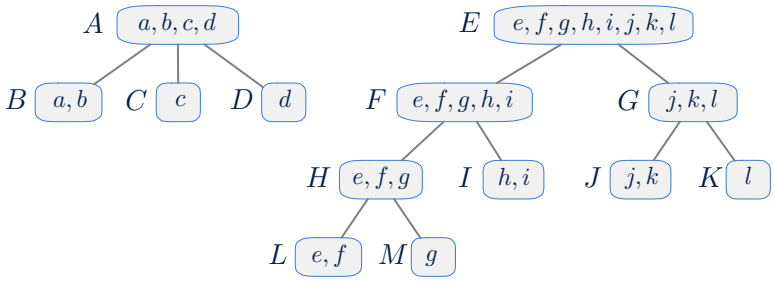


Figure 3 An example of a laminar family $\mathcal{P}$ of height 4.

4 Slot-Laminar Paging

In this section we prove upper bounds for Slot-Laminar Paging given in Table 1. Recall that in Slot-Laminar Paging family $\mathcal{S}$ is assumed to be a laminar family of slot sets whose height we denote by h. Theorem 8 bounds the optimal ratios by $3h^2k$ (deterministic), $3h^2H_k$ (randomized) and $3h^2$ (offline polynomial-time approximation). The proof of Theorem 8 (Section 4.2) is by a reduction of Slot-Laminar Paging to Page-Laminar Paging, which we study in Section 4.1. Theorem 10, presented in Section 4.3, tightens the deterministic upper bound to $2hk$.

4.1 Page-Laminar Paging

Recall that Page-Laminar Paging generalizes Paging by allowing each request to be a set P of pages. The request P is satisfiable by having any page $p \in P$ in the cache. We require $P \in \mathcal{P}$, where $\mathcal{P}$ is a pre-specified laminar collection of sets of pages, whose height we denote by h. (See the example in Figure 3.) To our knowledge, this problem has not been yet studied in the literature. In particular, we do not know whether the optimum solution can be computed in polynomial time. (If the number of non-leaf sets in $\mathcal{P}$ is constant, the problem can be solved in polynomial time using dynamic programming and Belady-like rules for choosing pages to fetch and evict.)

▶ **Theorem 6.** *Page-Laminar Paging admits the following polynomial-time algorithms: an hk-competitive deterministic online algorithm, an hH_k-competitive randomized online algorithm, and an offline h-approximation algorithm.*

The proof is by reduction to standard Paging. Known polynomial-time algorithms for standard Paging include an optimal offline algorithm [10], a deterministic k-competitive online algorithm [43] and a randomized H_k-competitive online algorithm [1]. Theorem 7 follows directly from composing these known results with the following lemma, whose proof is deferred to the full paper [23].

▶ **Lemma 7.** *Every $f(k)$-approximation algorithm $\mathbb{A}$ for Paging can be converted into an $hf(k)$-approximation algorithm $\mathbb{B}$ for Page-Laminar Paging, preserving the following properties: being polynomial-time, online, and/or deterministic.*

4.2 Upper bounds for randomized and offline Slot-Laminar Paging

▶ **Theorem 8.** *Slot-Laminar Paging admits the following polynomial-time algorithms: a deterministic $3h^2k$-competitive online algorithm, a randomized $3h^2H_k$-competitive online algorithm, and an offline $3h^2$-approximation algorithm.*

Our focus here is on uniform treatment of the three variants of Slot-Laminar Paging in the above theorem, and the ratios in this theorem have not been optimized. For example, in Section 4.3 we give a better deterministic algorithm. For the special case when $h = 2$ the problem can be reduced to All-or-One Paging, for which the ratio can be improved even further [20].

The proof of Theorem 8 is by a reduction of Slot-Laminar Paging to Page-Laminar Paging, in Lemma 9 (proved in Appendix A). The reduction uses a relaxation of Slot-Laminar Paging that relaxes the constraint that each slot hold at most one page (but still enforces the cache-capacity constraint), yielding an instance of Page-Laminar Paging. The reduction simulates the given Page-Laminar Paging algorithm on multiple instances of Page-Laminar Paging—one for each set $S \in \mathcal{S}$, obtained by relaxing the subsequence that contains just those requests contained in S — then aggregates the resulting Page-Laminar Paging solutions to obtain the global Slot-Laminar Paging solution. Lemma 9 and Theorem 6 (for Page-Laminar Paging) immediately imply Theorem 8.

▶ **Lemma 9.** *Every $f_h(k)$-approximation algorithm $\mathbb{A}$ for Page-Laminar Paging can be converted into a $3hf_h(k)$-approximation algorithm $\mathbb{B}$ for Slot-Laminar Paging, preserving the following properties: being polynomial-time, online, and/or deterministic.*

4.3 Improved upper bound for deterministic Slot-Laminar Paging

For Slot-Laminar Paging, this section presents a deterministic algorithm with competitive ratio $O(hk)$, improving upon the bound of $O(h^2k)$ from Theorem 8. The algorithm, RefSearch, refines ExhSearch. Like ExhSearch, it is phase-based and maintains a configuration that can satisfy all requests in a phase; however, in order to satisfy the next request in the current phase, the particular configuration is chosen by judiciously moving pages in certain slots that are serving requests along a path in the laminar hierarchy.

▶ **Theorem 10.** *For Slot-Laminar Paging, Algorithm* RefSearch *(Fig. 4) has competitive ratio at most $2 \cdot \mathsf{mass}(\mathcal{S}) - k \le (2h-1)k$.*

We begin by defining the terminology used in the algorithm and the proof, and establish some useful properties. Recall that a configuration D satisfies a request $r = \langle p, S\rangle$ if there exists a slot s in S such that s holds p in D; in this case, we also say that slot s satisfies r in D. A configuration D is said to *satisfy a set* R of requests if it satisfies every request in R. A set R of requests will be called *satisfiable* if there exists a configuration that satisfies R. To determine if a set R of requests is satisfied by a configuration, it is sufficient (and necessary) to examine the maximal subset of "deepest" requests in the laminar hierarchy. Formally, a request $\langle p, S\rangle$ is an *ancestor* (resp., *descendant*) of $\langle p, S'\rangle$ if $S \supseteq S'$ (resp., $S \subseteq S'$). For any set R of requests, define $\mathsf{rep}(R)$ as the set of requests in R that do not have any proper descendants in R. That is, $\mathsf{rep}(R) = \{\langle p, S\rangle \in R : \forall S' \subsetneq S, \langle p, S'\rangle \notin R\}$. For $r = \langle p, S\rangle$, define $\mathsf{anc}(r, R) = \{\langle p, S'\rangle \in R : S \subseteq S'\}$. Lemma 11 (proved in the full paper [23]) establishes some basic properties of $\mathsf{rep}(R)$.

▶ **Lemma 11.** *Let R be a set of requests. Then,*

- **(i)** *In any configuration, each slot can satisfy at most one request in $\mathsf{rep}(R)$.*
- **(ii)** *A configuration satisfies R if and only if it satisfies $\mathsf{rep}(R)$.*
- **(iii)** *R is satisfiable iff for any requestable set S, $\mathsf{rep}(R)$ has at most $|S|$ requests to subsets of S.*

Algorithm RefSearch is given in Figure 4. It consists of phases. The first phase starts in time Step 1, and each phase ends when adding the current request to the request set from this phase makes it unsatisfiable. Within a phase, redundant requests, that is those

input: Slot-Laminar Paging instance $(k, \mathcal{S}, \sigma = (\sigma_1, \ldots, \sigma_T))$
1. for $t \leftarrow 1, 2, \ldots, T$, respond to the current request $\sigma_t = \langle p, S \rangle$ as follows:
1.1. if $t = 1$ or $R_{t-1} \cup \{\sigma_t\}$ is not satisfiable: let $R_{t-1} = \emptyset$ and empty the cache — *start new phase*
1.2. let $R_t = R_{t-1} \cup \{\sigma_t\}$
1.3. if C_{t-1} satisfies $\sigma_t = \langle p, S \rangle$: let $C_t = C_{t-1}$ — *redundant request*
1.4. else: — *non-redundant request*
1.4.1. find sequences $\langle s_1, \ldots, s_m \rangle$, $\langle S_0 = S, S_1, \ldots, S_{m-1} \rangle$, and $\langle p_0 = p, p_1, \ldots, p_{m-1} \rangle$ s.t.
(i) $S_{i-1} \subsetneq S_i$ and slot $s_i \in S_{i-1}$ of C_{t-1} satisfies $\langle p_i, S_i \rangle \in \mathsf{rep}(R_{t-1})$, for $1 \leq i < m$, and
(ii) Slot $s_m \in S_{m-1}$ of C_{t-1} either
(ii.1) does not satisfy any requests in $\mathsf{rep}(R)$, or
(ii.2) satisfies a request $\langle p, S' \rangle \in \mathsf{rep}(R_{t-1})$ such that $S' \supsetneq S_{m-2}$
1.4.2. to obtain C_t and satisfy $\langle p_{i-1}, S_{i-1} \rangle$, place p_{i-1} in slot s_i, for $1 \leq i \leq m$

Figure 4 Deterministic online Slot-Laminar Paging algorithm RefSearch. Note that in Step 1.4.1 we have $m \leq k + 1 - |S|$, and that in (ii), if s_m satisfies $\langle p, S' \rangle \in \mathsf{rep}(R_{t-1})$ then $m \geq 2$ (because C_{t-1} does not satisfy σ_t); thus S_{m-2} is well-defined.

satisfied by the current configuration, are ignored (Step 1.3). To serve a non-redundant request $\sigma_t = \langle p, S \rangle$, the cache content is rearranged to free a slot in S. This rearrangement involves shifting the content of some slots that serve requests in $\mathsf{rep}(R)$ along the path from S to the root, to find a slot that is either unused or holds p (Step 1.4.2).

For technical reasons, in the analysis of Algorithm RefSearch it will be useful to introduce a slightly refined concept of configurations. Given a request set R, an *R-configuration* is a configuration D in which each request in $\mathsf{rep}(R)$ is served by exactly one slot. (By Lemma 11(i), each slot can serve only one request in $\mathsf{rep}(R)$, but in general in a configuration serving R there may be multiple slots that serve the same request in $\mathsf{rep}(R)$.) Slots in D that do not serve requests in $\mathsf{rep}(R)$ are called *free in D*. Observe that each configuration C_t of Algorithm RefSearch implicitly is an R_t-configuration – due to the assignment of slots in Step 1.4.2. Also, if the slot s_m chosen by the algorithm in Step 1.4.1 satisfies condition (ii.1) then s_m is a free slot of D, according to our definition.

The following helper claim, which characterizes when a particular request is not satisfied by a given configuration, follows directly from Lemma 11(iii).

▷ Claim 12. Let R be a set of requests and D be an R-configuration. Let also $r = \langle p, S \rangle$ be a request such that D does not satisfy r, yet $R \cup \{r\}$ is satisfiable. Then D has a slot s in S that is either free or satisfies a request $\langle p', S' \rangle \in \mathsf{rep}(R)$ where $S \subsetneq S'$.

The following lemma establishes the validity of Steps 1.4.1 and 1.4.2 of Algorithm RefSearch. We defer the proof of Lemma 13 and the full proof of Theorem 10 to Appendix A.

▶ **Lemma 13.** *Let R be a set of requests and D be an R-configuration. Let $r = \langle p_0, S_0 \rangle$ be a request such that r is not satisfied by D and $R \cup \{r\}$ is satisfiable. Then there exist sequences $\langle s_1, \ldots, s_m \rangle$, $\langle S_0, S_1, \ldots, S_{m-1} \rangle$, and $\langle p_0, p_1, \ldots, p_{m-1} \rangle$ such that (i) $S_{i-1} \subsetneq S_i$ and $s_i \in S_{i-1}$ is currently satisfying request $\langle p_i, S_i \rangle \in \mathsf{rep}(R)$, for $1 \leq i < m$, and (ii) $s_m \in S_{m-1}$ is either a free slot or is currently satisfying $\langle p_0, S' \rangle \in \mathsf{rep}(R)$ for some $S' \supsetneq S_{m-2}$. Furthermore, transforming D by moving page p_{i-1} to slot s_i (and modifying the slot assignment in D accordingly), for $1 \leq i \leq m$, yields an $(R \cup \{r\})$-configuration.*

input: Weighted All-Or-One Paging instance (k, σ), where $\sigma_t = \langle p_t, s_t \rangle$ for $t \in [T]$
1. initialize $\mathsf{cap}[t] \leftarrow \mathsf{credit}[t] \leftarrow 0$ for each $t \in [T]$
2. assume that $\langle p_t, s_t \rangle = \langle 0, t \rangle$ for $t \in [k]$ — *k specific requests to artificial weight-0 page in each slot*
3. for $t \leftarrow k+1, k+2, \ldots, T$:
3.1. if $\langle p_t, s_t \rangle$ is a specific request with no equivalent request t' (s.t. $\langle p_{t'}, s_{t'} \rangle = \langle p_t, s_t \rangle$) in the cache:
3.1.1. evict any cached general request to page p_t, and any cached request in slot s_t
3.1.2. put t in slot s_t — *note* $\mathsf{cap}[t] = \mathsf{credit}[t] = 0$
3.2. else if $\langle p_t, s_t \rangle$ is a general request not satisfied by any cached request t' (s.t. $p_{t'} = p_t$):
3.2.1. let
$\ell_t(s) \coloneqq \max\{t' \le t : s_{t'} = s\}$ for $s \in [k]$ — *most recent specific request to s*
$A \coloneqq \{s \in [k] : \mathsf{cap}[\ell_t(s)] \ge \frac{1}{2}\mathsf{wt}(p_t)$, *$s$ does not hold a specific request*$\}$
$B \coloneqq \{s \in [k] :$ *slot s holds a general request of weight at least* $\frac{1}{2}\mathsf{wt}(p_t)\}$
3.2.2. while $|A| \le |B|$:
3.2.2.1. continuously raise $\mathsf{cap}[\ell_t(s)]$ for $s \in [k]$ and $\mathsf{credit}[t']$ for each cached request t', at unit rate,
3.2.2.2. evicting each request t' such that $\mathsf{credit}[t'] = \mathsf{wt}(p_{t'})$, and updating A and B continuously
3.2.3. choose a slot $s \in A \setminus B$; evict the request t' currently in slot s (if any)
3.2.4. put t in slot s — *note* $\mathsf{credit}[t] = 0$
3.3. else: classify the (already satisfied) request as *redundant* and ignore it

Figure 5 An $O(k)$-competitive online algorithm for Weighted All-Or-One Paging. For technical convenience, we present the algorithm as caching request times rather than pages, with the understanding that request t represents page p_t.

5 Weighted All-Or-One Paging

This section initiates the study of Heterogenous k-Server in non-uniform metrics. Weighted All-Or-One Paging is the natural weighted extension of All-or-One Paging (allowing general and specific requests) in which the pages have weights and the cost of retrieving a page is its weight. (This is equivalent to Heterogenous k-Server in star metrics with requestable set family $\mathcal{S} = \{[k]\} \cup \{\{s\} : s \in [k]\}$.) This section proves the following theorem:

▶ **Theorem 14.** *Weighted All-Or-One Paging has a deterministic $O(k)$-competitive online algorithm.*

The bound is optimal up to a small constant factor, as the optimal ratio for standard Weighted Paging is k. Figure 5 shows the algorithm. It is implicitly a linear-programming primal-dual algorithm. Note that the standard linear program for standard Weighted Paging doesn't have constraints that force pages into specific slots — indeed, those constraints make even the unweighted problem an $\mathbb{NP}$-hard special case of Multicommodity Flow. As a small example that illustrates the challenge, consider a cache of size $k = 2$, and repeatedly make three requests: a general request to a weight-1 page, and specific requests to different weight-zero pages in slots 1 and 2. The weight-zero requests force the weight-1 page to be evicted with each round, so the optimal cost is the number of rounds. But the solution of the the classical linear-program relaxation will have value 1. Thus this linear program cannot be used to bound the competitive ratio.

Proof sketch of Theorem 14. Fix an optimal solution C, that is $\mathsf{opt}(\sigma) = \mathsf{cost}(C)$. For each $t \in [T]$, let $x_t \in \{0,1\}$ be an indicator variable for the event that C evicts request t before satisfying another request $t' > t$ with the same page/slot pair that satisfied t. Let $R \subseteq [T]$ be the set of all specific requests, and for each $t \in R$, let y_t be the amount C pays to retrieve pages into slot s_t before the next specific request to slot s_t (if any). Define the *pseudo-cost* of the optimal solution to be $\sum_{t=1}^{T} \mathsf{wt}(p_t)x_t + \sum_{t\in R} y_t$. The pseudo-cost is at most $2\,\mathsf{opt}(\sigma)$. As the algorithm proceeds, define the *residual cost* to be $\sum_{t=1}^{T} \max(0, \mathsf{wt}(p_t)x_t - \mathsf{credit}[t]) + \sum_{t\in R} \max(0, y_t - \mathsf{cap}[t])$. The residual cost is initially the pseudo-cost (at most $2\,\mathsf{opt}(\sigma)$), and remains non-negative throughout, so the total decrease in the residual cost is at most $2\,\mathsf{opt}(\sigma)$. In Appendix B, we show in Lemma 15 that whenever the algorithm is raising credits and capacities at time t, there is either a cached request t' with $x_{t'} = 1$ and $\mathsf{credit}[t'] < \mathsf{wt}(p_{t'})$, or there is a slot s with $y_{t'} > \mathsf{cap}[t']$, where $t' = \ell_t(s) \in R$. It follows that the residual cost is decreasing at least at unit rate in Step 3.2.2.1.

On the other hand, the algorithm is raising k capacities and at most k credits, so the value of $\phi = \sum_{t=1}^{T} \mathsf{credit}[t] + \sum_{t\in R} \mathsf{cap}[t]$ is increasing at rate at most $2k$. So, the final value of ϕ is at most $4k\,\mathsf{opt}(\sigma)$. To finish, we show by a charging argument that the algorithm's cost is at most $6\,\phi + 3\,\mathsf{opt}(\sigma) \le (24k+3)\,\mathsf{opt}(\sigma)$. We defer the full proof to Appendix B. ◀

6 Open Problems

The results here suggest many open problems and avenues for further research. Closing or tightening gaps left by our upper and lower bounds would be of interest. In particular:

- For Slot-Heterogenous Paging, is the upper bound in Theorem 5 tight for every $\mathcal{S} \subseteq 2^{[k]} \setminus \{\emptyset\}$, within $\mathrm{poly}(k)$ factors?
- For Page-Laminar Paging it is easy to show a lower bound of $\Omega(h)$, even for $k = 1$ and for randomized algorithms. But it still may be possible to eliminate or reduce the multiplicative dependence on h. For example, is it possible to achieve ratio $O(h + k)$ with a deterministic algorithm and $O(h + H_k)$ with a randomized algorithm? Similarly, does Slot-Laminar Paging (where $h \le k$) admit an $O(k)$ deterministic ratio and $O(\log k)$ randomized ratio?
- For deterministic All-or-One Paging, we conjecture that the optimal ratio is $2k - 1$. (For $k = 2$ we can show an upper bound of 3.) In the randomized case, can ratio $2H_k - 1$ be achieved?
- For Weighted All-Or-One Paging, is the optimal randomized ratio $O(\mathrm{polylog}(k))$?
- The status of Heterogenous k-Server in arbitrary metric spaces is widely open. Can ratio dependent only on k be achieved? This question, while challenging, could still be easier to resolve for Heterogenous k-Server than for Generalized k-Server.

References

1 Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000. doi:10.1016/S0304-3975(98)00116-9.

2 C. J. Argue, Anupam Gupta, Ziye Tang, and Guru Guruganesh. Chasing convex bodies with linear competitive ratio. *Journal of the ACM*, 68(5):32:1–32:10, August 2021. doi:10.1145/3450349.

3 Nikhil Ayyadevara and Ashish Chiplunkar. The randomized competitive ratio of weighted k-server is at least exponential. *CoRR*, abs/2102.11119, 2021. arXiv:2102.11119.

4 Nikhil Bansal, Niv Buchbinder, Aleksander Mądry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 267–276. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.63.

5 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 507–517. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.7.

6 Nikhil Bansal, Marek Eliás, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 493–504. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.52.

7 Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 992–1001, 2018. doi:10.1137/1.9781611975031.64.

8 Nikhil Bansal, Joseph (Seffi) Naor, and Ohad Talmon. Efficient online weighted multi-level paging. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 94–104. ACM, 2021. doi:10.1145/3409964.3461801.

9 Nathan Beckmann, Phillip B. Gibbons, Bernhard Haeupler, and Charles McGuffey. Writeback-aware caching. In Bruce M. Maggs, editor, *1st Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Salt Lake City, UT, USA, January 8, 2020*, pages 1–15. SIAM, 2020. doi:10.1137/1.9781611976021.1.

10 L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966. doi:10.1147/sj.52.0078.

11 Marcin Bienkowski, Łukasz Jeż, and Pawel Schmidt. Slaying Hydrae: Improved bounds for generalized k-server in uniform metrics. In Pinyan Lu and Guochuan Zhang, editors, *30th International Symposium on Algorithms and Computation, ISAAC 2019*, volume 149 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ISAAC.2019.14.

12 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

13 Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992. doi:10.1145/146585.146588.

14 Mark Brehob, Richard J. Enbody, Eric Torng, and Stephen Wagner. On-line restricted caching. *J. Sched.*, 6(2):149–166, 2003. doi:10.1023/A:1022989909868.

15 Mark Brehob, Stephen Wagner, Eric Torng, and Richard J. Enbody. Optimal replacement is NP-hard for nonstandard caches. *IEEE Trans. Computers*, 53(1):73–76, 2004. doi:10.1109/TC.2004.1255792.

16 Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Mądry. K-server via multiscale entropic regularization. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16. ACM, 2018. doi:10.1145/3188745.3188798.

17 Sébastien Bubeck, Yuval Rabani, and Mark Sellke. Online multiserver convex chasing and optimization. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2093–2104. SIAM, 2021.

18 Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive algorithms for restricted caching and matroid caching. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms – ESA 2014 – 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2014. doi:10.1007/978-3-662-44777-2_18.

19 Niv Buchbinder, Christian Coester, and Joseph (Seffi) Naor. Online k-taxi via double coverage and time-reverse primal-dual. In Mohit Singh and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization – 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings*, volume 12707 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2021. doi:10.1007/978-3-030-73879-2_2.

20 Jannik Castenow, Björn Feldkord, Till Knollmann, Manuel Malatyali, and Friedhelm Meyer auf der Heide. The k-server with preferences problem. In *SPAA '22: 34rd ACM Symposium on Parallelism in Algorithms and Architectures*, 2022. To appear. URL: https://arxiv.org/abs/2205.11102.

21 Ashish Chiplunkar and Sundar Vishwanathan. Metrical service systems with multiple servers. *Algorithmica*, 71(1):219–231, 2015. doi:10.1007/s00453-014-9903-7.

22 Ashish Chiplunkar and Sundar Vishwanathan. Randomized memoryless algorithms for the weighted and the generalized k-server problems. *ACM Trans. Algorithms*, 16(1), December 2019. doi:10.1145/3365002.

23 Marek Chrobak, Samuel Haney, Mehraneh Liaee, Debmalya Panigrahi, Rajmohan Rajaraman, Ravi Sundaram, and Neal E. Young. Online paging with heterogeneous cache slots, 2022. arXiv:2206.05579.

24 Marek Chrobak and John Noga. Competitive algorithms for relaxed list update and multilevel caching. *J. Algorithms*, 34(2):282–308, 2000. doi:10.1006/jagm.1999.1061.

25 Christian Coester and Elias Koutsoupias. The online k-taxi problem. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1136–1147. ACM, 2019. doi:10.1145/3313276.3316370.

26 Christian Coester and Elias Koutsoupias. Towards the k-server conjecture: A unifying potential, pushing the frontier to the circle. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 57:1–57:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.57.

27 Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Trans. Archit. Code Optim.*, 8(4), January 2012.

28 Esteban Feuerstein. Uniform service systems with k servers. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Cláudio L. Lucchesi, and Arnaldo V. Moura, editors, *LATIN'98: Theoretical Informatics*, volume 1380, pages 23–32, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. doi:10.1007/BFb0054307.

29 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991. doi:10.1016/0196-6774(91)90041-V.

30 Amos Fiat, Manor Mendel, and Steven S. Seiden. Online companion caching. In Rolf Möhring and Rajeev Raman, editors, *Algorithms — ESA 2002*, pages 499–511, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

31 Amos Fiat and Moty Ricklin. Competitive algorithms for the weighted server problem. *Theor. Comput. Sci.*, 130(1):85–99, 1994. doi:10.1016/0304-3975(94)90154-6.

32 Samuel Haney. *Algorithms for Networks With Uncertainty*. PhD thesis, Duke University, 2019. URL: https://dukespace.lib.duke.edu/dspace/handle/10161/18661.

33 Shahin Kamali and Helen Xu. Multicore paging algorithms cannot be competitive. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 547–549, 2020.

34 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988. doi:10.1007/BF01762111.

35 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

36 Elias Koutsoupias and David Scot Taylor. The CNN problem and other k-server variants. *Theor. Comput. Sci.*, 324(2-3):347–359, 2004. doi:10.1016/j.tcs.2004.06.002.

37 Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B. Lee. CATalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 406–418, 2016. doi:10.1109/HPCA.2016.7446082.

38 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

39 Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991. doi:10.1007/BF01759073.

40 M. Mendel and Steven S. Seiden. Online companion caching. *Theoretical Computer Science*, 324(2):183–200, 2004. Online Algorithms: In Memoriam, Steve Seiden. doi:10.1016/j.tcs.2004.05.015.

41 Jignesh Patel. Restricted k-server problem. Master's thesis, Michigan State University, 2004. URL: https://d.lib.msu.edu/etd/32678.

42 Mark Sellke. Chasing convex bodies optimally. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 1509–1518. Society for Industrial and Applied Mathematics, 2020. doi:10.1137/1.9781611975994.92.

43 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. doi:10.1145/2786.2793.

44 Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 494–505, New York, NY, USA, 2007. doi:10.1145/1250662.1250723.

45 Ying Ye, Richard West, Zhuoqun Cheng, and Ye Li. COLORIS: A dynamic cache partitioning system using page coloring. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 381–392, 2014. doi:10.1145/2628071.2628104.

46 Wei Zang and Ann Gordon-Ross. CaPPS: cache partitioning with partial sharing for multi-core embedded systems. *Des. Autom. Embed. Syst.*, 20(1):65–92, 2016. doi:10.1007/s10617-015-9168-7.

A Proofs for Slot-Laminar Paging

A.1 General upper bounds via reduction to page-laminar paging

Proof of Lemma 9. We first define the Page-Laminar Paging *relaxation* of a given Slot-Laminar Paging instance. The idea is to relax the constraint that each slot can hold at most one page, while keeping the capacity constraint. The relaxed problem is equivalent to a Page-Laminar Paging instance over "virtual" pages $v(p,s)$ corresponding to page/slot pairs (p,s). This virtual page can be placed in any slot, although it represents p being in slot s.

Formally, this relaxation is defined as follows. Fix any k-slot Slot-Heterogenous Paging instance $\sigma = (\sigma_1, \ldots, \sigma_T)$ with requestable slot-set family $\mathcal{S}$. For any page p and $S \in \mathcal{S}$, define $V(p,S) = \{v(p,s) : s \in S\}$, where $v(p,s)$ is a *virtual page* for the pair (p,s). Define the *relaxation* of σ to be the k-slot Page-Subset Paging instance $\pi = (P_1, \ldots, P_T)$ defined by $P_t = V(p_t, S_t)$ (where $\sigma_t = \langle p_t, S_t \rangle$, for $t \in [T]$). The requestable set family for π is $\mathcal{P} = \big\{V(p,S) : p \text{ is any page and } S \in \mathcal{S}\big\}$. Crucially, if $\mathcal{S}$ is slot-laminar with height h, then $\mathcal{P}$ is page-laminar with the same height h.

Instance π is a relaxation of σ in the sense that for any solution C for σ there is a solution D for π with $\mathsf{cost}(D) \le \mathsf{cost}(C)$. (Namely, have D keep in its cache the virtual pages $v(p,s)$ such that C has page p cached in slot s.) It follows that $\mathsf{opt}(\pi) \le \mathsf{opt}(\sigma)$.

Next we define the algorithm $\mathbb{B}$. Fix an $f_h(k)$-approximation algorithm $\mathbb{A}$ for Page-Laminar Paging. Fix the input σ with $\sigma_t = \langle p_t, S_t \rangle$ (for $t \in [T]$) to Slot-Laminar Paging algorithm $\mathbb{B}$. We assume for ease of presentation that Algorithm $\mathbb{A}$ is an online algorithm, and present Algorithm $\mathbb{B}$ as an online algorithm. If $\mathbb{A}$ is not online, $\mathbb{B}$ can easily be executed as an offline algorithm instead.

Assume that the family $\mathcal{S}$ has just one root R with $|R| \leq k$. (This is without loss of generality, as multiple roots, being disjoint, naturally decouple any Slot-Laminar Paging instance into independent problems, one for each root.)

For each $S \in \mathcal{S}$, define S's Slot-Laminar Paging *subinstance* σ_S to be obtained from σ by deleting all requests that are not subsets of S. Let π_S denote the (Page-Laminar Paging) relaxation of σ_S. Algorithm $\mathbb{B}$ on input σ executes, simultaneously, $\mathbb{A}(\pi_S)$ for every requestable set $S \in \mathcal{S}$, giving each execution $\mathbb{A}(\pi_S)$ its own independent cache of size $|S|$ composed of copies of the slots in S.

For each such S, Algorithm $\mathbb{B}$ will build its own solution, denoted $\mathbb{B}(\sigma_S)$, for σ_S, also using its own independent cache of size $|S|$ composed of copies of the slots in S. The desired solution to σ will then be $\mathbb{B}(\sigma_R)$ (note that $\sigma = \sigma_R$).

For internal bookkeeping purposes only, in presenting Algorithm $\mathbb{B}$, we consider each virtual page $v(p, s)$ (as defined for Page-Laminar Paging) to be a *copy* of page p, and we have $\mathbb{B}$ maintain cache configurations that place these virtual pages in specific slots, with the understanding that the actual cache configurations are obtained by replacing each virtual page $v(p, s)$ (in whatever slot it's in) by a copy of page p. This virtual copy $v(p, s)$ is functionally equivalently to p; for example, if placed in slot s', it will satisfy any request $\langle p, S' \rangle$ with $s' \in S'$. When we analyze the cost, we will consider two copies $v(p, s)$ and $v(p', s')$ to be distinct unless $(p', s') = (p, s)$. In particular, if $\mathbb{B}$ evicts $v(p, s)$ while retrieving $v(p, s')$ (with $s' \neq s$) in the same slot, this contributes 1 to the cost of $\mathbb{B}$. We will upper bound $\mathbb{B}$'s cost overestimated in this way.

Correctness. the following invariant over time. For each requestable set S, for each virtual page $v(p, s)$ currently cached by $\mathbb{A}(\pi_S)$:
1. the solution $\mathbb{B}(\sigma_S)$ caches $v(p, s)$ in some slot in S, and
2. if S has a child c with $s \in c$, and $\mathbb{B}(\sigma_c)$ has $v(p, s)$ in its cache c, then in $\mathbb{B}(\sigma_S)$ copy $v(p, s)$ is in the same slot as in $\mathbb{B}(\sigma_c)$.

The invariant suffices to guarantee correctness of the solution $\mathbb{B}(\sigma_S)$ for each instance σ_S. Indeed, when $\mathbb{B}(\sigma_S)$ receives a request $\langle p_t, S_t \rangle$, its relaxation $\mathbb{A}(\pi_S)$ has just received the request $\{v(p_t, s) : s \in S_t\}$, so $\mathbb{A}(\pi_S)$ is caching a virtual page $v(p_t, s)$ (for some $s \in S_t$) in S. By Condition 1, then, $\mathbb{B}(\sigma_S)$ also has $v(p_t, s)$ in some slot in S. In the case $S = S_t$, this suffices for $\mathbb{B}(\sigma_S)$ to satisfy the request. In the remaining case S has a child c with $S_t \subseteq c$, and $\mathbb{B}(\sigma_C)$ just received the same request, so (assuming inductively that $\mathbb{B}(\sigma_c)$ is correct for σ_c) $\mathbb{B}(\sigma_c)$ has $v(p_t, s)$ in some slot s' in S_t, so by Condition 2 of the invariant $\mathbb{B}(\sigma_S)$ has $v(p_t, s)$ in the same slot s' in S_t, as required. In particular, $\mathbb{B}(\sigma_R)$ will be correct for σ_R.

To maintain the invariant $\mathbb{B}$ does the following for each requestable set S. Whenever the relaxed solution $\mathbb{A}(\pi_S)$ evicts a page $v(p, s)$, the solution $\mathbb{B}(\sigma_S)$ also evicts $v(p, s)$. After this eviction both Conditions 1 and 2 will be preserved. Whenever $\mathbb{A}(\pi_S)$ retrieves a page $v(p, s)$, the solution $\mathbb{B}(\sigma_S)$ also retrieves $v(p, s)$, into any vacant slot in S (there must be one, because $\mathbb{A}(\sigma_S)$ caches at most $|S|$ pages). This retrieval can cause up to two violations of Condition 2 of the invariant: one at $\mathbb{B}(\sigma_S)$, because $v(p, s)$ is already cached by a child $\mathbb{B}(\sigma_c)$ but in some slot $s_1 \neq s'$; the other at the parent $\mathbb{B}(\sigma_P)$ of $\mathbb{B}(\sigma_S)$ (if any), because $v(p, s)$ is already cached by the parent, but in some slot $s_2 \neq s'$. In the case that the retrieval does

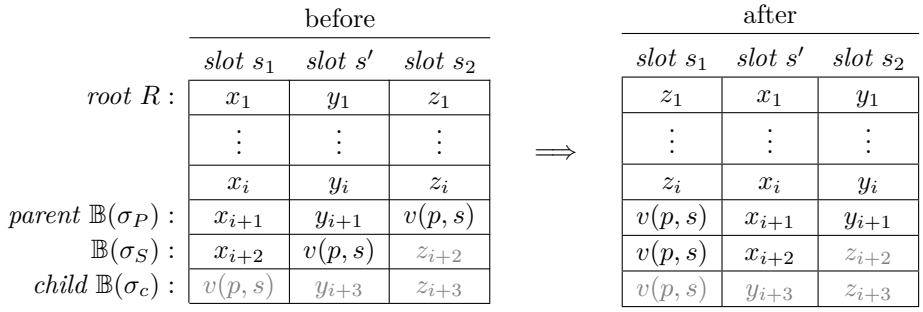


Figure 6 "Rotating" slots in $\mathbb{B}(\sigma_S)$ and ancestors to preserve the invariant. Pages in grey are not moved.

create two violations (and $s_1 \neq s_2$), $\mathbb{B}$ restores the invariant by "rotating" the contents of the slots s_1, s', and s_2 in $\mathbb{B}(\sigma_S)$ *and in each ancestor*, as shown in Figure 6. Note that y_{i+3} and z_{i+2} cannot be $v(p,s)$, so moving $v(p,s)$ out of slots s' and s_2 doesn't introduce a violation there. Thus this rotation indeed restores the invariant, at the expense of three retrievals at the root. (The retrievals at other nodes only modify the internal state of $\mathbb{B}$.) There are three other cases: two violations with $s_1 = s_2$, one violation at $\mathbb{B}(\sigma_S)$, or one violation at its parent, but all these three cases can be handled similarly, also with at most three retrievals (in fact at most two) at the root.

Total cost. Each retrieval by $\mathbb{A}(\pi_S)$ causes at most 3 retrievals in $\mathbb{B}(\sigma_R)$, so $\mathsf{cost}(\mathbb{B}(\sigma_R))$

$$\leq \sum_{S\in\mathcal{S}} 3\,\mathsf{cost}(\mathbb{A}(\pi_S)) \leq \sum_{S\in\mathcal{S}} 3\,f_h(|S|)\mathsf{opt}(\pi_S) \leq 3f_h(k)\sum_{S\in\mathcal{S}} \mathsf{opt}(\sigma_S) \leq 3hf_h(k)\,\mathsf{opt}(\sigma_R).$$

The second step uses that $\mathbb{A}(\pi_S)$ is $f_h(|S|)$-competitive for π_S. The third step uses that π_S is a relaxation of σ_S so $\mathsf{opt}(\pi_S) \leq \mathsf{opt}(\sigma_S)$, and that $|S| \leq k$ so $f_h(|S|) \leq f_h(k)$.[1] The last step uses that the sets within any given level $i \in \{1,2,\ldots,h\}$ of the laminar family are disjoint, so $\mathsf{opt}(\sigma_R)$ is at least the sum, over the sets S within level i, of $\mathsf{opt}(\sigma_S)$. This shows that $\mathbb{B}$ is a $3hf_h(k)$-approximation algorithm. To finish, we observe that $\mathbb{B}$ is polynomial-time, online, and/or deterministic if $\mathbb{A}$ is. ◀

A.2 Improved upper bound for deterministic case

Proof of Lemma 13. The proof is by induction on the depth of S_0 in the laminar hierarchy. For the induction base, consider $S_0 = [k]$. Since r is not satisfied by D, $R \cup \{r\}$ is satisfiable, and every requestable slot set is subset of $[k]$, we obtain from Claim 12 that there is a free slot $s_1 \in S_0$. The desired claim of the lemma holds with $m = 1$ and sequences $\langle s_1\rangle$, $\langle S_0\rangle$ and $\langle p_0\rangle$ which satisfy (i). Since s_1 is free, bringing page p_0 to slot s_1 yields a $(R \cup \{r\})$-configuration.

We now establish the induction step. Let R, D, and $r = \langle p_0, S_0\rangle$ be as given. By Claim 12 there are two cases. In the first case, there is a free slot $s_1 \in S_0$ in D. Then the desired claim holds with $m = 1$, and sequences $\langle s_1\rangle$, $\langle S_0\rangle$ and $\langle p_0\rangle$. Furthermore, as in the base case, since s_1 is free, bringing page p_0 to slot s_1 yields an $(R \cup \{r\})$-configuration.

[1] We assume here that $f_h(k') \leq f_h(k)$ for $k' \leq k$, which is without loss of generality as one can simulate a cache of size k' using a cache of size k by introducing artificial requests that force $k - k'$ slots to be continuously occupied.

The remainder of this proof concerns the second case, in which there is a slot $s_1 \in S_0$ currently satisfying a request $r' = (p_1, S_1)$ in $\mathsf{rep}(R)$ with $S_0 \subsetneq S_1$. Let D' denote the configuration that is identical to D except that D has p_0 in slot s_1. Since D is an R-configuration, no other slot satisfies r' in D; the same holds in D'. Hence, D' does not satisfy r'. Furthermore, D' satisfies every request in $\mathsf{rep}(R)$ other than r'. Let $R' = R \cup \{r\} \setminus \mathsf{anc}(r', R)$. In D', s_1 satisfies r. Consider any request x in $R \setminus \mathsf{anc}(r', R)$. By definition of $\mathsf{rep}(R)$, there exists a request x' in $\mathsf{rep}(R)$ that is a descendant of x. Since R' does not include any ancestors of r', x' is not r' and hence is satisfied by some slot in D'. We thus obtain that D' satisfies R' and, in fact D' is an R'-configuration. In D' slot s_1 is assigned to r, and if there is a request (p, S') in $\mathsf{rep}(R)$ then its assigned slot is designated as free in D'. At the same time, D' does not satisfy r'. Further, since $R' \cup \{r'\}$ is a subset of $R \cup \{r\}$, which is satisfiable, $R' \cup \{r'\}$ is also satisfiable. Since $S_1 \supsetneq S_0$, by the induction hypothesis, there are sequences $\langle s_2, \ldots, s_m\rangle$, $\langle S_1, S_2, \ldots S_{m-1}\rangle$ and $\langle p_1, p_2, \ldots, p_{m-1}\rangle$ such that (i) $S_{i-1} \subsetneq S_i$ and $s_i \in S_{i-1}$ is currently satisfying $(p_i, S_i) \in \mathsf{rep}(R')$, for $2 \leq i < m$; and either (ii.1) s_m is a free slot in D' or (ii.2) is currently satisfying a request $(p_1, S') \in \mathsf{rep}(R')$ for some $S' \supsetneq S_1$. Note, however, that that s_m has to be a free slot in D' since (ii.2) above cannot hold: any request (p_1, S') is in $\mathsf{anc}(r', R)$, all requests of which are excluded from R'. Furthermore, transforming D' to D'' by moving page p_{i-1} to s_i for $2 \leq i \leq m$, satisfies $R' \cup \{r'\}$.

We now establish the desired claim for D, R, and r. Consider sequences $\langle s_1, \ldots, s_m\rangle$, $\langle S_0, S_1, \ldots S_{m-1}\rangle$ and $\langle p_0, \ldots, p_{m-1}\rangle$. The desired condition (i) follows from (i) of the induction step above and the fact that in D, $s_1 \in S_0$ is currently satisfying a request (p_1, S_1) in $\mathsf{rep}(R)$ with $S_0 \subsetneq S_1$. For (ii), note that since s_m is a free slot in D', either s_m is a free slot in D or (p_0, S') is in $\mathsf{rep}(R)$ for some $S' \supsetneq S_{m-2}$, thus establishing (ii). Finally, transforming D to D'' by moving p_{i-1} to s_i for $1 \leq i \leq m$, satisfies $R' \cup \{r'\}$. Since any request satisfying r' also satisfies all ancestors of r', we have $\mathsf{rep}(R \cup \{r\}) = \mathsf{rep}(R' \cup \{r'\})$, implying that D'' also satisfies $R \cup \{r\}$. This completes the induction step and the proof of the lemma. ◀

Proof of Theorem 10. We first argue that at any time t, configuration C_t of RefSearch satisfies the set R_t of requests from the current phase of the algorithm. The proof is by induction on the number of steps within a phase. When the phase is about to start at time t then R_{t-1} is set to $\emptyset$, so the claim holds. For the induction step, consider a step t within a phase and assume that C_{t-1} satisfies R_{t-1}. If C_{t-1} satisfies new request σ_t, then by Step 3.3, C_t satisfies R_t. Otherwise, $R_{t-1} \cup \{\sigma_t\}$ is satisfiable but C_{t-1} does not satisfy σ_t. Then, by Lemma 13, Steps 1.4.1 and 1.4.2 derive a configuration C_t satisfying R_t, completing the induction step and the argument that at any time t, C_t satisfies R_t.

We next analyze the competitive ratio. We first show that the number of page retrievals during a phase of RefSearch is at most $2 \cdot \mathsf{mass}(\mathcal{S})$. Let R denote the set of requests in the current phase. We charge the cost in this phase to the depths of the requests in $\mathsf{rep}(R)$. The cost of Step 1.4.2 is m. If s_m satisfies condition (ii.1), then $\mathsf{rep}(R \cup \{\sigma_t\}) = \mathsf{rep}(R) \cup \{\sigma_t\}$ and the depth of S is at least m, so the charge per unit depth is at most 1. Otherwise, condition (ii.2) holds and $\mathsf{rep}(R \cup \{\sigma_t\}) = \mathsf{rep}(R) \cup \{\sigma_t\} \setminus \{\langle p, S'\rangle\}$. In this case we have σ_t inherit the charges to $\langle p, S'\rangle$, and we charge the cost of m to the difference in depths of S and S', which is at least $m - 1$ (because $S_{m-2} \subsetneq S'$), so the charge per unit of depth is at most $m/(m-1) \leq 2$. (Note that in this case $m \geq 2$.) When the phase ends, a request at depth d was charged at most d times, and these charges include at least a unit charge, so its total charge is most $2d-1$. So, the algorithm's cost per phase is at most $2 \cdot \mathsf{mass}(\mathcal{S}) - k \leq (2h-1)k$. The optimal cost in a phase is at least 1 as no configuration satisfies all requests in the phase and the request that starts the next phase. The theorem follows. ◀

B Proofs for Weighted All-Or-One Paging

Consider any execution of the algorithm on a k-slot instance σ. To ease notation and streamline the analysis, without loss of generality we will make the following assumptions:

- The first k requests are specific requests for an artificial 0-wt page in each of the k slots.
- Each request is not redundant (per Step 3.3).
- The last k requests are specific requests for an artificial 0-wt page in each of the k slots.

These assumptions can be made without loss of generality as the zero-weight requests do not have any cost, the algorithm ignores redundant requests, and removing redundant requests doesn't increase the optimum cost. For technical convenience, we think of the algorithm and the optimal solution as caching request *times* rather than pages, with the understanding that request t represents page p_t.

▶ **Lemma 15.** *Suppose that, while responding to a general request t, the algorithm is executing Step 3.2.2.1 (that is, the loop condition in Step 3.2.2 is satisfied). Then, in any solution C, just after C has responded to request t, either*
(i) C has evicted some request t' currently cached by the algorithm, or
(ii) for some slot $s \in [k]$, after the most recent specific request $\ell_t(s)$ to slot s solution C has incurred cost more than $\mathsf{cap}[\ell_t(s)]$ for retrievals into s.

Proof. If C satisfies property (i), we are done. If (i) doesn't hold then, just after responding to request t, in addition to the current general request p_t, solution C caches every request t' that is cached by the algorithm. This, together with the loop condition, implies that C has at least $|B| + 1 \geq |A| + 1$ generally requested pages of weight at least $\frac{1}{2}\,\mathsf{wt}(p_t)$ in its cache. Thus one of these pages, say $p_{t'}$, is in a slot $s \notin A$. The choice of $p_{t'}$ and the definition of A imply then that the cost of C for retrievals into s after time $\ell_t(s)$ is at least $\mathsf{wt}(p_{t'}) \geq \frac{1}{2}\,\mathsf{wt}(p_t) > \mathsf{cap}[\ell_t(s)]$, so property (ii) holds. ◀

Proof of Theorem 14. Fix an optimal solution C, that is $\mathsf{opt}(\sigma) = \mathsf{cost}(C)$. For each $t \in [T]$, let $x_t \in \{0, 1\}$ be an indicator variable for the event that C evicts request t before satisfying another request $t' > t$ with the same page/slot pair that satisfied t. Let $R \subseteq [T]$ be the set of all specific requests, and for each $t \in R$, let y_t be the amount C pays to retrieve pages into slot s_t before the next specific request to slot s_t (if any). Define the *pseudo-cost* of the optimal solution to be $\sum_{t=1}^{T} \mathsf{wt}(p_t)x_t + \sum_{t\in R} y_t$. The pseudo-cost is at most $2\,\mathsf{opt}(\sigma)$. As the algorithm proceeds, define the *residual cost* to be $\sum_{t=1}^{T} \max(0, \mathsf{wt}(p_t)x_t - \mathsf{credit}[t]) + \sum_{t\in R} \max(0, y_t - \mathsf{cap}[t])$. The residual cost is initially the pseudo-cost (at most $2\,\mathsf{opt}(\sigma)$), and remains non-negative throughout, so the total decrease in the residual cost is at most $2\,\mathsf{opt}(\sigma)$. By Lemma 15, whenever the algorithm is raising credits and capacities at time t, there is either a cached request t' with $x_{t'} = 1$ and $\mathsf{credit}[t'] < \mathsf{wt}(p_{t'})$, or there is a slot s with $y_{t'} > \mathsf{cap}[t']$, where $t' = \ell_t(s) \in R$. It follows that the residual cost is decreasing at least at unit rate in Step 3.2.2.1.

On the other hand, the algorithm is raising k capacities and at most k credits, so the value of $\phi = \sum_{t=1}^{T} \mathsf{credit}[t] + \sum_{t\in R} \mathsf{cap}[t]$ is increasing at rate at most $2k$. So, the final value of ϕ is at most $4k\,\mathsf{opt}(\sigma)$. To finish, we show that the algorithm's cost is at most $6\,\phi + 3\,\mathsf{opt}(\sigma) \leq (24k + 3)\,\mathsf{opt}(\sigma)$. Count the costs that the algorithm pays as follows:

1. *Requests remaining in the cache at the end (time T).* By the assumption on the last k requests, these cost nothing to bring in. All other requests are evicted.
2. *Requests evicted in Line 3.2.2.2.* Each such request t' is evicted only after $\mathsf{credit}[t']$ reaches $\mathsf{wt}(p_{t'})$. So these have total weight at most $\sum_{t'=1}^{T} \mathsf{credit}[t']$.

3. *Specific requests t' evicted from slot s_t in Line 3.1.1.* Throughout the time interval $[t', t-1]$, the algorithm has $p_{t'}$ in slot $s_{t'} = s_t$, and σ has neither an equivalent specific request nor a general request to p_t (by our non-redundancy assumption). The optimal solution C has $p_{t'}$ in slot $s_{t'}$ at time t', but not at time t, so evicts it during $[t'+1, t]$. So the total cost of such requests is at most the total weight of specific requests evicted by C, and thus at most $\mathsf{opt}(\sigma)$.
4. *General requests evicted from slot s_t in Line 3.1.1.* By Line 3.2.3, any general request in slot s_t at time t has weight at most $2\,\mathsf{cap}[\ell_{t-1}(s_t)]$. So the total weight of such requests is at most $2\sum_{t' \in R} \mathsf{cap}[t']$.
5. *General requests to page p_t evicted in Line 3.1.1.* The algorithm replaces each such general request t' by a specific request t (which it later evicts, unless the weight is zero) to the same page. Have general request t' *charge* its cost $\mathsf{wt}(p_{t'}) = \mathsf{wt}(p_t)$, and any amount charged to t' (in Item 6 below), to specific request t. (We analyze the charging scheme for Items 5 and 6 below.)
6. *General requests t' evicted in Line 3.2.3.* Have request t' charge the cost of its eviction, and any amount charged to t' to request t. Since the slot holding $p_{t'}$ is not in B, $\mathsf{wt}(p_{t'}) < \frac{1}{2}\,\mathsf{wt}(p_t)$.

Each general request t receives at most one charge in Item 6, from a request t' of at most half the weight of t; this general request t' may also receive such charges, forming a chain of charges, but since the weights of the requests in this chain decrease geometrically, t is charged at most its weight. In Item 5, each specific request t is charged by at most one general request t' of the same weight, that may also carry the chain charge not exceeding its weight. So this specific request is charged at most twice its weight. Overall, the charge of each request from Items 5 and 6 is at most twice its weight.

The total weight of evictions considered in Items 1, 2, 3, and 4 is at most $2\,\phi + \mathsf{opt}(\sigma)$. Adding also the charges to these items by evictions considered in Items 5 and 6, we obtain that the total cost of the algorithm is bounded by $3\,(2\,\phi + \mathsf{opt}(\sigma)) = 6\,\phi + 3\,\mathsf{opt}(\sigma)$. ◀

On Rational Recursive Sequences

Lorenzo Clemente
University of Warsaw, Poland

Maria Donten-Bury
University of Warsaw, Poland

Filip Mazowiecki
University of Warsaw, Poland

Michał Pilipczuk
University of Warsaw, Poland

Abstract

We study the class of rational recursive sequences (ratrec) over the rational numbers. A ratrec sequence is defined via a system of sequences using mutually recursive equations of depth 1, where the next values are computed as rational functions of the previous values. An alternative class is that of simple ratrec sequences, where one uses a single recursive equation, however of depth k: the next value is defined as a rational function of k previous values.

We conjecture that the classes ratrec and simple ratrec coincide. The main contribution of this paper is a proof of a variant of this conjecture where the initial conditions are treated symbolically, using a formal variable per sequence, while the sequences themselves consist of rational functions over those variables. While the initial conjecture does not follow from this variant, we hope that the introduced algebraic techniques may eventually be helpful in resolving the problem.

The class ratrec strictly generalises a well-known class of polynomial recursive sequences (polyrec). These are defined like ratrec, but using polynomial functions instead of rational ones. One can observe that if our conjecture is true and effective, then we can improve the complexities of the zeroness and the equivalence problems for polyrec sequences. Currently, the only known upper bound is Ackermanian, which follows from results on polynomial automata. We complement this observation by proving a PSPACE lower bound for both problems for polyrec. Our lower bound construction also implies that the Skolem problem is PSPACE-hard for the polyrec class.

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

Keywords and phrases recursive sequences, polynomial automata, zeroness problem, equivalence problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.24

Related Version *arXiv Version*: https://arxiv.org/abs/2210.01635

Funding This work is a result of research conducted within projects no. 2017/26/D/ST6/00201 (Lorenzo Clemente) and 2017/26/E/ST1/00231 (Maria Donten-Bury) financed by National Science Centre, Poland, as well as a part of projects INFSYS (Lorenzo Clemente and Filip Mazowiecki) and BOBR (Michał Pilipczuk) that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreements no. 950398 and 948057, respectively).

Acknowledgements We thank Szymon Toruńczyk for helpful discussions.

1 Introduction

The topic of this paper are recursively defined sequences of rational numbers $\mathbb{N} \to \mathbb{Q}$. There are two natural ways to define such sequences. In a *simple recursion of depth* k one fixes k initial values and defines the next value as a function of the previous k values. This is how the Fibonacci sequence is usually defined (with $k = 2$): $f_0 = 0$, $f_1 = 1$, and $f_{n+2} = f_{n+1} + f_n$. In a *mutual recursion of width* k one defines a system of k sequences such that every sequence

© Lorenzo Clemente, Maria Donten-Bury, Filip Mazowiecki, and Michał Pilipczuk;
licensed under Creative Commons License CC-BY 4.0

has its initial value and the update function can access the immediately previous value of all k sequences, but no older value. For example, we can define $a_n = n^2$ with an extra sequence $b_n = n$ as follows: $a_0 = b_0 = 0$ and $a_{n+1} = a_n + 2b_n + 1$, $b_{n+1} = b_n + 1$. Both styles allow to define various classes of sequences depending on what operations are allowed in the equations, and in general mutual recursion of width k can simulate simple recursion of depth k (by adding sufficiently many auxiliary sequences).

One of the most well-known classes of sequences is the class of *linear recursive sequences*, which is obtained by allowing the update function to use addition and multiplication with constants. These are usually defined with a simple recursion, like in the Fibonacci example, but in fact, as a consequence of the Cayley-Hamilton theorem, one obtains the same class when using mutual recursion [20, Lemma 1.1]. In particular, all the example sequences f_n, a_n and b_n are linear recursive.

Another natural class of sequences are the *polynomial recursive sequences* (*polyrec*), which are defined with mutual recursion and updates from the ring of polynomial functions $\mathbb{Q}[x_1, \ldots, x_k]$. An example sequence from this class is $c_n = n!$, where one can use the already defined sequence b_n and define $c_0 = 1$ and $c_{n+1} = c_n \cdot (b_n + 1)$. To see the polynomials behind this definition, let x and y be variables corresponding to b_n and c_n, respectively. The polynomial to define b_{n+1} is $P_b(x, y) = x + 1$, and the polynomial to define c_{n+1} is $P_c(x, y) = y(x + 1)$. The class of *simple polynomial recursive sequences* is obtained by using polynomial updates and a simple recursion (instead of mutual recursion) and it is known to be strictly included in the class of all polyrec sequences. In particular, the sequence c_n is polyrec but not simple polyrec [13, Theorem 3.1].

The definition via mutual recursion appears in the area of control theory (under the name *implicit representation* of the space of states), and, in computer science, in the context of *weighted automata* over $\mathbb{Q}$. Such automata output a rational number for every word over a finite alphabet Σ, and they are defined by linear updates [17]. Linear recursive sequences are thus equivalent to weighted automata with a 1-letter alphabet $\Sigma = \{a\}$ [3]. Similarly, polyrec sequences are equivalent to *polynomial automata* [4] (also known as *cost-register automata* [1]) with a 1-letter alphabet [13].

We are interested in two classical decision problems for such automata. *Equivalence*: Given two automata $\mathcal{A}$ and $\mathcal{B}$ do they output the same number for every word, and *zeroness*: Does the input automaton $\mathcal{A}$ output 0 for every word. These problems are well-known to be efficiently equivalent to each other: Zeroness is clearly a special case of equivalence (just take $\mathcal{B}$ to output zero for every word), and equivalence of $\mathcal{A}, \mathcal{B}$ reduces to zeroness of the difference automaton $\mathcal{A} - \mathcal{B}$ with the expected semantics. Therefore, we will consider only the zeroness problem. From the seminal work of Schützenberger on minimisation of weighted automata it follows that the zeroness problem for weighted automata is in PTIME [32] (in fact even in NC^2 [36]). For polynomial automata over a binary alphabet, zeroness is known to be Ackermann-complete [4]. Using the connection between sequences and automata one immediately obtains NC^2 and Ackermann upper bounds for the zeroness problem of linear recursive sequences, resp., polyrec sequences.

Let us take a closer look at the zeroness problem for recursive sequences, i.e., given a sequence u_n is it the case that $u_n = 0$ for all $n \in \mathbb{N}$? The zeroness problem is a fundamental problem for number sequences. It is a basic building block in computer algebra, e.g., in proving identities involving recursively defined sequences. It is also important from a theoretical point of view as a yardstick of the well-behavedness of classes of number sequences, i.e., interesting classes of sequences should at least have a decidable zeroness problem. The difficulty of solving the zeroness problem in general depends on how the sequence is presented. If the sequence is

defined with a simple recursion of depth k such as $u_{n+k} = f(u_{n+k-1}, \ldots, u_n)$, then zeroness trivially reduces to checking that the first k values are 0 and that the recursive update f is well-defined and needs to output 0 when the previous values are 0, i.e., $f(0, \ldots, 0) = 0$. However, this simple reasoning is flawed in the case of mutual recursion, because the auxiliary sequences employed in the mutual recursion need not be zero. However, for linear recursive sequences the zeroness problem is easily solved even in the case of mutual recursion, because the reduction to simple recursion [20, Lemma 1.1] implies that a_n is zero if, and only if, its first $k+1$ values $a_0 = \cdots = a_k$ are zero. For polyrec sequences we cannot apply this argument since mutual recursion cannot be simulated by simple recursion in the case of polynomial updates.

Our results. In this paper we introduce the class of *rational recursive sequences* (*ratrec*). This class is defined with mutual recursion and updates from the field of rational functions $\mathbb{Q}(x_1, \ldots, x_k)$. For example, the Catalan numbers $C_{n+1} = \frac{2(2n+1)}{n+2} C_n$ can be defined using b_n as an auxiliary sequence. Namely, $C_0 = 1$ and $C_{n+1} = \frac{2(2b_n+1)}{b_n+2} C_n$, where the rational function used to define C_{n+1} is $R(x, y) = \frac{2(2x+1)}{x+2} y$. By definition, the class of polyrec sequences is included in the class of ratrec sequences, and in fact the inclusion is strict as witnessed by the fact that the Catalan numbers C_n are not polynomialy recursive [13, Corollary 4.1]. Moreover, ratrec sequences also include the well-known and wide-spread *P-recursive sequences*[1] [22], which according to a 2005 estimate comprise at least 25% of the OEIS archive [31].

A natural question is whether the class of ratrec sequences semantically collapses to the class of *simple rational recursive sequences* obtained by adopting simple recursion. Unlike in the case of polynomial updates, we conjecture that for rational updates we do have such a collapse.

▶ **Conjecture 1.** *The class of rational recursive sequences coincides with the class of simple rational recursive sequences.*

To see the power of ratrec sequences recall that $c_n = n!$ is not a simple polyrec sequence. However, when in the recursion we allow rational functions, then c_n can be defined with a simple recursion, namely: $c_{n+2} = \frac{(c_{n+1})^2}{c_n} + c_{n+1}$. Thus c_n is simple ratrec.

We introduce a technique towards proving Conjecture 1, which comes from commutative algebra. Instead of looking at the elements of a ratrec sequence as numbers in the field of rationals $\mathbb{Q}$, we symbolically view them as elements of the field of rational functions $\mathbb{Q}(x_1, \ldots, x_k)$. More precisely, we assume that the sequences $\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(k)}$ are initialised by setting $F_0^{(i)} = x_i$ for all $i \in \{1, \ldots, k\}$; then, a system of recursive equations governed by rational functions defines further entries of the sequences. Thus, the recursive definition will output elements in $\mathbb{Q}(x_1, \ldots, x_k)$ rather than $\mathbb{Q}$. Intuitively, this corresponds to treating the initial conditions of a system of ratrec sequences symbolically, rather than instantiating them with actual rational values.

Informally speaking, we prove Conjecture 1 for symbolic ratrec sequences, as explained above. Here is a semi-formal statement of our main result, see Theorem 6 for a formalization.

▶ **Theorem 2.** *The class of rational recursive sequences over $\mathbb{Q}(x_1, \ldots, x_k)$, with the system initialised by $F_0^{(i)} = x_i$, coincides with the class of simple rational recursive sequences.*

[1] Sometimes P-recursive sequences are also called *holonomic sequences*, due to a connection with holonomic generating functions.

The proof proceeds as follows. From the functions defining the ratrec system we build a sequence of field extensions

$$\mathbb{Q} \subseteq \mathbb{F}_0 \subseteq \mathbb{F}_1 \subseteq \mathbb{F}_2 \subseteq \ldots \subseteq \mathbb{Q}(x_1, \ldots, x_k)$$

and translate the problem of belonging to the class of simple ratrec sequences to the question of whether this sequence of field extensions eventually stabilises. In order to estimate at which level the stabilisation occurs we use certain results on basic algorithms for rational function fields [25]. We believe that this technique could be extended to prove Conjecture 1, but we also show an example why our current results are not strong enough.

Note that if Conjecture 1 is moreover efficient, it gives a simple algorithm to check zeroness for polyrec. Indeed, since polyrec is a particular case of ratrec, then once a sequence is expressed as a simple ratrec it suffices to check whether the first elements of the sequence are 0. This would improve the Ackermann upper bound inherited from polynomial automata from [4]. This suggests that for polyrec sequences the natural object of study are rational function fields, which are of more algebraic nature and could provide better complexity bounds than the order-theoretic techniques based on sequences of polynomial ideals and Hilbert's finite basis theorem [4].

Our final result is a complexity lower bound for the zeroness problem of polyrec sequences.

▶ **Theorem 3.** *The zeroness problem for polynomial recursive sequences is PSPACE-hard.*

As far as we know, prior to this work nothing was known about the complexity of zeroness for polyrec sequences, except for the Ackermann upper bound following from polynomial automata [4]. The lower bound is proved by reducing from the QBF validity problem.

Given Conjecture 1 it seems natural to investigate the zeroness problem for ratrec sequences (not just for polyrec sequences). The issue is that it is not clear what would be the input for such a decision problem. Recall that to define ratrec sequences we allow for rational functions in the recursion, which means that we have to deal with division in order to compute the elements of the sequences. Then either one would require that the input sequence comes with a promise that all elements are well-defined and no division by 0 occurs; or one would need to verify whether division by 0 occurs in the input sequence. We find the former solution unnatural, and the latter is at least as hard as the so-called Skolem problem (cf. below), which is not known to be decidable even for linear recursive sequences.

Related work. The zeroness problem has been extensively studied. In the field of automata theory, we can mention applications to the equivalence problem of several classes of automata and grammars, starting from weighted finite automata [32] and polynomial automata [4] already mentioned above, and including context-free grammars [14], multiplicity equivalence of finite automata [36] and multitape finite automata [21, 37], unambiguous context-free grammars [30, Theorem 5.5] (cf. [19, 15] for a PSPACE upper bound), polynomial grammars (which generalise polynomial automata) [8, Chapter 11], deterministic top-down tree-to-string transducers [33], MSO transductions on unordered forests [6, 7], MSO transductions of bounded treewidth under a certain equivalence relation [9], Parikh automata [10], and unambiguous register automata [2]. By replacing (pointwise) multiplication with convolution in the definition of polyrec sequences we obtain the so-called *convolution recursive sequences*, for which the zeroness problem can be solved in PSPACE [15, Theorem 4].

The zeroness problem of D-finite [38] and, more generally, D-algebraic power series [16, 35] is known to be decidable, but its computational complexity has not been investigated. We remark that the class of numerical sequences with D-algebraic power series is incomparable

with ratrec. For instance, the sequence 2^{2^n} is ratrec (in fact, already polyrec) but it cannot be D-algebraic since any D-algebraic sequence has growth rate $n!^{O(1)}$ [24, Ch. 8, Theorem 16]. On the other hand, the exponential generating function of the sequence n^{n-1} counting labelled rooted trees with n nodes is known to be D-algebraic [11]. In other words, the ordinary generating function of $n^{n-1}/n!$ is D-algebraic. If the latter sequence were ratrec, so it would be n^n since $n!$ and n are ratrec and ratrec sequences are closed under Hadamard (i.e., pointwise) product. However n^n it is not ratrec [13, Theorem 5.3].

A natural problem related to the zeroness problem is the so-called *Skolem problem*, which asks whether a given sequence a_n has a zero, i.e., whether for some n we have $a_n = 0$. As a corollary of the constructions used to prove Theorem 3, it follows that the Skolem problem for polyrec sequences is PSPACE-hard. Only NP-hardness was formerly known, and already for linear recursive sequences [5, Corollary 2.1]. Decidability of the Skolem problem for linear recursive sequences is a long-standing open problem (cf. the survey paper [27]). It is interesting to notice that those lower bounds are obtained already on the fixed field with two elements $\{0, 1\}$, and are thus of a combinatorial rather than numerical nature. The Skolem problem for weighted automata over $\mathbb{Q}$ (that generalise linear recursive sequences) is undecidable [29].

2 Preliminaries

By $\mathbb{N}$ we denote the set of nonnegative integers. We denote an arbitrary field by $\mathbb{F}$, and we use 0 and 1 to denote the zero, resp., one elements thereof. Example fields of interest in this paper are: rationals $\mathbb{Q}$; and the two-element field $\mathbb{F}_2$. A *sequence* over a *domain* $\mathbb{D}$ is a function $u\colon \mathbb{N} \to \mathbb{D}$. The sequences considered in this work are over domains that have a field structure, like rationals $\mathbb{Q}$. We use bold-face letters as a short-hand for sequences, e.g., $\mathbf{u} = \langle u_n \rangle_{n\in\mathbb{N}}$.

In this paper we work with multivariate polynomials and rational functions. The *(combined) degree* of a monomial $x_1^{d_1} \cdots x_k^{d_k}$ is $d_1 + \cdots + d_k$ and the degree of a polynomial $P \in \mathbb{Q}(x_1, \ldots, x_k)$, written $\deg P$, is the maximum degree of monomials appearing in it. A *rational function* is a formal fraction of two polynomials, where the denominator is required to be non-zero. The degree of a rational function is the maximum of the degrees of the numerator and the denominator. Recall that for any field $\mathbb{F}$ and a set of variables $x_1, \ldots, x_n$, polynomials over $x_1, \ldots, x_n$ form the ring $\mathbb{F}[x_1, \ldots, x_n]$, while rational functions over $x_1, \ldots, x_n$ form the field $\mathbb{F}(x_1, \ldots, x_n)$. We also write $\mathbb{F}[\mathbf{x}]$ and $\mathbb{F}(\mathbf{x})$, where $\mathbf{x} = (x_1, \ldots, x_n)$.

The computational aspects of multivariate polynomials, in particular their representation on input to algorithms, are explained in Appendix A, as they will be of no concern in Sections 3 and 4.

3 Rational recursive sequences

We start with the central definitions, which were already discussed in Section 1.

▶ **Definition 4.** *A sequence* $\mathbf{u}^{(1)}$ *over a field* $\mathbb{F}$ *is* rationally recursive *(or* ratrec *for short) of* dimension k *and* degree D *if there exist auxiliary sequences* $\mathbf{u}^{(2)}, \ldots, \mathbf{u}^{(k)}$ *over* $\mathbb{F}$ *and rational functions* $P_1, \ldots, P_k \in \mathbb{F}(x_1, \ldots, x_k)$ *of degree at most* D *such that for all* $n \in \mathbb{N}$*, we have*

$$\begin{cases} u^{(1)}_{n+1} &= P_1(u^{(1)}_n, \ldots, u^{(k)}_n), \\ &\vdots \\ u^{(k)}_{n+1} &= P_k(u^{(1)}_n, \ldots, u^{(k)}_n). \end{cases} \tag{1}$$

A sequence $\mathbf{u}$ *over a field* $\mathbb{F}$ *is* polynomially recursive *(or* polyrec *for short) if it satisfies the same definition above, where* $P_1, \ldots, P_k$ *are taken as polynomials in* $\mathbb{F}[x_1, \ldots, x_k]$*. We refer to* $(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(k)})$ *as the system defining* $\mathbf{u}^{(1)}$*.*

In what follows we assume that whenever $\mathbf{u}$ is a ratrec sequence, say defined by a system $(\mathbf{u} = \mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(k)})$, for all $n \in \mathbb{N}$ all the right hand sides of equations (1) are well-defined, that is, denominators on the right hand side never evaluate to zero.[2]

For instance, the sequence of *Catalan numbers* $C_n = \frac{1}{n+1} \cdot \binom{2n}{n}$ is ratrec. This can be seen in several ways. For example, they satisfy the recurrence $C_{n+1} = \frac{2(2n+1)}{n+2} \cdot C_n$, giving rise to the following ratrec system:

$$\begin{cases} u_{n+1} &= \frac{2(2v_n+1)}{v_n+2} \cdot u_n, \\ v_{n+1} &= v_n + 1. \end{cases}$$

More generally, any P-recursive sequence a_n is ratrec. A sequence a_n is *P-recursive* [34, Sec. 6.4] if it satisfies a single recursion of the form

$$P_0(n) \cdot a_n + P_1(n) \cdot a_{n+1} + \cdots + P_d(n) \cdot a_{n+d} = 0, \tag{2}$$

for every n large enough, where $P_0, \ldots, P_d \in \mathbb{Q}[n]$ are polynomials of the index variable n with $P_d \neq 0$. This is essentially transformed into the ratrec system

$$\begin{cases} u_{n+1}^{(d)} &= -\frac{P_0(v_n)}{P_d(v_n)} \cdot u_n^{(0)} - \cdots - \frac{P_{d-1}(v_n)}{P_d(v_n)} \cdot u_n^{(d-1)}, \\ u_{n+1}^{(d-1)}, &= u_n^{(d)}, \\ &\vdots \\ u_{n+1}^{(0)} &= u_n^{(1)}, \\ v_{n+1} &= v_n + 1, \end{cases} \tag{3}$$

However, (3) works only assuming that $P_d(n) \neq 0$. Since P_d is a fixed polynomial, there exists n_d such that $\forall_{n \geqslant n_d} P_d(n) \neq 0$. Thus in the final version of (3) we add n_d more sequences in the system to include the first n_d elements of the P-recursive sequence a_n.

Assuming $v_0 = 0$ and $u_0^{(0)} = a_0, \ldots, u_0^{(d)} = a_d$, it is immediate to verify $v_n = n$ and $u_n^{(0)} = a_n, \ldots, u_n^{(d)} = a_{n+d}$ for every $n \in \mathbb{N}$.

The family of ratrec sequences strictly includes both P-recursive sequences and polyrec sequences. As an example consider the sequence $u_n = 2^{2^n} + C_n$. On the one hand, this sequence is certainly ratrec because it is the sum of a polyrec and a P-recursive sequence (which are ratrec) and ratrec sequences are closed under sum. On the other hand, u_n is not P-recursive since it grows asymptotically faster than any P-recursive sequence (every P-recursive sequence is in $O((n!)^\gamma)$ for some constant $\gamma \in \mathbb{R}$ [23, Proposition 3.11]). Further, recall that 2^{2^n} is polyrec, and C_n is not polyrec [13, Corollary 4.1]. Since polyrec sequences are closed under sum and subtraction, we conclude that u_n is not polyrec.

In [13, Theorem 7.1], the following property of ratrec sequences is proved: if $\mathbf{u}$ is ratrec, then there exists $m \in \mathbb{N}$ and a *cancelling polynomial* $P \in \mathbb{Q}[y_0, \ldots, y_m]$, that is, a non-zero polynomial such that

$$P(u_n, u_{n+1}, \ldots, u_{n+m}) = 0 \qquad \text{for all } n \in \mathbb{N}.$$

[2] As mentioned in the introduction, checking whether a ratrec sequence is well-defined is at least Skolem hard, which means it is not known to be decidable. Thus for decision problems one should restrict to classes like polyrec, where sequences are always well-defined.

▶ **Theorem 5** (Theorem 7.1 in [13]). *Every ratrec sequence admits a cancelling polynomial.*

In [13, Theorem 5.3] it is shown that the sequence $u_n = n^n$ has no cancelling polynomial, and hence is not polyrec and not ratrec.

4 Extension degrees

In this section we consider ratrec sequences as in Definition 4 over the field $\mathbb{Q}(\mathbf{x})$. Let $(\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(k)})$ be a system defining $\mathbf{F}^{(1)}$. In this section we will consider sequences with the following fixed initial conditions: $F_0^{(i)} = x_i$. Note that this technical assumption is important, in particular we cannot initialise $F_0^{(i)}$ with elements in $\mathbb{Q}$. (If we could, this class would generalise ratrec over the field $\mathbb{Q}$.)

Theorem 6 below formalises Theorem 2 and is the main result of this paper. In essence, we show that a ratrec definition over $\mathbb{Q}(\mathbf{x})$ can be translated to a simple ratrec over $\mathbb{Q}(\mathbf{x})$ with a polynomial recursion depth. We hope that this insight might lead towards a resolution of Conjecture 1.

▶ **Theorem 6.** *Let $\mathbf{F}^{(1)}$ be a ratrec sequence over the field $\mathbb{Q}(\mathbf{x})$, defined by a system $(\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(k)})$, with the initial conditions: $F_0^{(i)} = x_i$ for $i = 1, \ldots, k$. Then there exists a rational function $R \in \mathbb{Q}(y_0, \ldots, y_m)$ such that*

$$F^{(1)}_{n+m+1} = R(F^{(1)}_n, F^{(1)}_{n+1}, \ldots, F^{(1)}_{n+m}), \text{ for all } n \in \mathbb{N}.$$

Moreover, if $F_n^{(1)}$ is of dimension k and degree D, then m can be bounded from above by $k + k^3 \log(kD)$.

Before we proceed to the proof, let us note that if we write $R(y_0, \ldots, y_m) = \frac{A(y_0,\ldots,y_m)}{B(y_0,\ldots,y_m)}$, where $A, B \in \mathbb{Q}[y_0, \ldots, y_m]$, then Theorem 6 shows that the following polynomial is cancelling for $\mathbf{u}^{(1)}$:

$$P(y_0, \ldots, y_m, y_{m+1}) = y_{m+1} \cdot B(y_0, \ldots, y_m) - A(y_0, \ldots, y_m).$$

Thus, Theorem 6 shows (and in fact, is equivalent to) that every ratrec sequence over $\mathbb{Q}(\mathbf{x})$ admits a cancelling polynomial that is linear in the last variable (here y_{m+1}), improving upon Theorem 5.

The remainder of this section is devoted to the proof of Theorem 6 and to a discussion related to it. In particular, the first part of the theorem (existence) will be proved in Section 4.1 and the concrete bound on the depth m will be proved in Section 4.2.

Let us make a few observations about the sequences $F_n^{(1)}, \ldots, F_n^{(k)}$. First, a straightforward estimation shows that the degrees of functions $F_n^{(1)}, \ldots, F_n^{(k)}$ grow at most single-exponentially in n.

▶ **Lemma 7.** *For $n \in \mathbb{N}$, let d_n be the maximum degree of $F_n^{(1)}, \ldots, F_n^{(k)}$. Then $d_n \leqslant (k \cdot D)^n$.*

Proof. We proceed by induction on n. Initially we have $d_0 = 1$ by definition. By Definition 4 $F_{n+1}^{(i)}$ is obtained by substituting rational functions $F_n^{(1)}, \ldots, F_n^{(k)}$ of degree at most d_n into a rational function P_i of degree at most D. Let $P_i = \frac{A}{B}$ be the ratio of two polynomials $A, B \in \mathbb{Q}[\mathbf{x}]$ of degree at most D. Let $C \in \mathbb{Q}[\mathbf{x}]$ be the least common multiple of all denominators of $F_n^{(1)}, \ldots, F_n^{(k)}$, and thus of degree at most $k \cdot d_n$. We can then write $F_n^{(1)} = \frac{G^{(1)}}{C}, \ldots, F_n^{(k)} = \frac{G^{(k)}}{C}$, where the numerators $G^{(1)}, \ldots, G^{(k)} \in \mathbb{Q}[\mathbf{x}]$ are polynomials of degree also at most $k \cdot d_n$. It follows that both $A(F_n^{(1)}, \ldots, F_n^{(k)})$ and $B(F_n^{(1)}, \ldots, F_n^{(k)})$

can be written as rational functions of the form $\frac{\hat{A}}{C^D}$, resp., $\frac{\hat{B}}{C^D}$, where the numerators are polynomials $\hat{A}, \hat{B} \in \mathbb{Q}[\mathbf{x}]$ of degree at most $D \cdot k \cdot d_n$ and the same holds for the common denominator $C^D \in \mathbb{Q}[\mathbf{x}]$. It follows that $F^{(i)}_{n+1}$ is a rational function of degree $d_{n+1} \leqslant k \cdot D \cdot d_n$, as required. ◀

The next lemma is a key property implied by the recurrence: if several consecutive elements of the sequence $F^{(i)}_n$ satisfy some algebraic constraint, then this constraint is also satisfied at every step later in the sequence.

▶ **Lemma 8** (Substitution lemma). *Suppose $Z(y_0, \ldots, y_m) \in \mathbb{Q}[y_0, \ldots, y_m]$ is a polynomial such that $Z(F^{(i)}_0(\mathbf{x}), \ldots, F^{(i)}_m(\mathbf{x})) = 0$. Then $Z(F^{(i)}_n(\mathbf{x}), \ldots, F^{(i)}_{n+m}(\mathbf{x})) = 0$ for all $n \in \mathbb{N}$.*

Proof. By assumption we have

$$Z(F^{(i)}_0(\mathbf{x}), \ldots, F^{(i)}_m(\mathbf{x})) = 0. \tag{4}$$

Consider the ring homomorphism $h\colon \mathbb{Q}[\mathbf{x}] \to \mathbb{Q}(\mathbf{x})$ that maps the variables $x_1, \ldots, x_k$ to rational functions $F^{(1)}_1, \ldots, F^{(k)}_1$, respectively. For a rational function P/Q such that $h(Q) \neq 0$, by $h(P/Q)$ we understand the rational function $h(P)/h(Q)$. (Note that such an extension of h to $\mathbb{Q}(\mathbf{x})$ does not have to be a field homomorphism.) From the definition of the sequence $F^{(i)}_n$ it readily follows by induction that

$$h(F^{(i)}_n) = F^{(i)}_{n+1}, \quad \text{for all } n \in \mathbb{N}.$$

Thus, by applying h to both sides of (4), we infer that

$$Z(F^{(i)}_1(\mathbf{x}), \ldots, F^{(i)}_{m+1}(\mathbf{x})) = 0.$$

We conclude by repeating this reasoning n times. ◀

In the following we introduce some basic terminology about (commutative) fields (cf. [26, Sec. II.1], [12, Sec. V.3], or [18, Sec. 13.1 and 13.2] for more details). Let $\mathbb{E}, \mathbb{F}$ be two fields. When $\mathbb{E} \subseteq \mathbb{F}$ we say that $\mathbb{F}$ is a *field extension* of $\mathbb{E}$, which is called the *base field*. Given a field extension $\mathbb{F}$ over $\mathbb{E}$ and elements $f_1, \ldots, f_n \in \mathbb{F}$, let $\mathbb{E}(f_1, \ldots, f_n)$ be the smallest field extension over $\mathbb{E}$ containing $f_1, \ldots, f_n$. If $\mathbb{F} = \mathbb{E}(f_1, \ldots, f_n)$, then we say that $\mathbb{F}$ is *finitely generated over* $\mathbb{E}$ (with *generators* $f_1, \ldots, f_n$).

The *degree* of $\mathbb{F}$ over $\mathbb{E}$, written $\deg_{\mathbb{E}} \mathbb{F}$, is the dimension of $\mathbb{F}$ as a vector space over the base field $\mathbb{E}$. For instance, $\mathbb{Q}(\sqrt{2})$ has degree 2 over $\mathbb{Q}$ (its elements can be put in the form $a + b \cdot \sqrt{2}$) and $\mathbb{Q}(\sqrt[3]{2})$ has degree 3 (its elements can be put in the form $a + b \cdot \sqrt[3]{2} + c \cdot (\sqrt[3]{2})^2$). Field extensions need not have finite degree. For instance, $\mathbb{Q}(\pi)$ and $\mathbb{Q}(x)$ are two field extensions of $\mathbb{Q}$ of infinite degree. The degree is multiplicative:

▶ **Lemma 9** (cf. [18, Theorem 14]). *Consider field extensions $\mathbb{E} \subseteq \mathbb{F} \subseteq \mathbb{G}$. Then, $\deg_{\mathbb{E}} \mathbb{G} = \deg_{\mathbb{E}} \mathbb{F} \cdot \deg_{\mathbb{F}} \mathbb{G}$ (even for infinite degrees).*

An element $f \in \mathbb{F}$ is *algebraic* over the base field $\mathbb{E}$ if there is a nonzero polynomial $P(x) \in \mathbb{E}[x]$ s.t. $P(f) = 0$. The field extension $\mathbb{F}$ is *algebraic* over the base field $\mathbb{E}$ if every element in $\mathbb{F}$ is algebraic over $\mathbb{E}$.

Let $\mathbb{F}$ be a field extension of $\mathbb{E}$. A subset $\{f_1, \ldots, f_n\} \subseteq \mathbb{F}$ of elements of $\mathbb{F}$ is *algebraically independent* over $\mathbb{E}$ if there is no nonzero polynomial $P(x_1, \ldots, x_n) \in \mathbb{E}[x_1, \ldots, x_n]$ such that $P(f_1, \ldots, f_n) = 0$. The *transcendence degree* of $\mathbb{F}$ over $\mathbb{E}$, denoted $\operatorname{tr}\deg_{\mathbb{E}} \mathbb{F}$, is the largest cardinality of a subset of elements of $\mathbb{F}$ which are algebraically independent over $\mathbb{E}$. Note that $\mathbb{F}$ is algebraic over $\mathbb{E}$ if and only if $\operatorname{tr}\deg_{\mathbb{E}} \mathbb{F} = 0$. Like the algebraic degree is multiplicative, the transcendence degree is additive:

▶ **Lemma 10** (cf. [12, Corollary to Theorem 4, A.5.111]). *Consider field extensions* $\mathbb{E} \subseteq \mathbb{F} \subseteq \mathbb{G}$. *Then,* $\operatorname{tr\,deg}_{\mathbb{E}} \mathbb{G} = \operatorname{tr\,deg}_{\mathbb{E}} \mathbb{F} + \operatorname{tr\,deg}_{\mathbb{F}} \mathbb{G}$.

In the following we will always take as the base field $\mathbb{E} = \mathbb{Q}$, in which case we will write just $\operatorname{tr\,deg} \mathbb{F}$ instead of $\operatorname{tr\,deg}_{\mathbb{Q}} \mathbb{F}$. For example, $\operatorname{tr\,deg} \mathbb{Q}(\sqrt{2}) = 0$ because $\sqrt{2}$ is an algebraic number over $\mathbb{Q}$, $\operatorname{tr\,deg} \mathbb{Q}(\sqrt{2}, \pi) = 1$ because π is a transcendental number, and $\operatorname{tr\,deg} \mathbb{Q}(x_1, \ldots, x_n) = n$.

The motivation to look at field extensions is that a ratrec system naturally defines the following sequence of field extensions

$$\mathbb{Q} \subseteq \mathbb{F}_0 \subseteq \mathbb{F}_1 \subseteq \ldots \subseteq \mathbb{Q}(\mathbf{x}), \tag{5}$$

where $\mathbb{F}_0 = \mathbb{Q}(x_1)$ and $\mathbb{F}_{n+1} = \mathbb{F}_n(F^{(1)}_{n+1}(\mathbf{x}))$ for $n \in \mathbb{N}$.

4.1 Ascending sequences of field extensions

In this section we prove the following Noether-like result.

▶ **Theorem 11.** *Consider any ascending sequence of field extensions of the form*

$$\mathbb{Q} \subseteq \mathbb{F}_0 \subseteq \mathbb{F}_1 \subseteq \ldots \subseteq \mathbb{Q}(x_1, \ldots, x_k).$$

Then the sequence eventually stabilises: there exists n_0 *such that* $\mathbb{F}_{n_0} = \mathbb{F}_{n_0+1} = \mathbb{F}_{n_0+2} = \ldots$.

The crucial reason for the result above is that the number k of variables is fixed. In the proof of Theorem 11 we use the following result on finitely generated extensions.

▶ **Lemma 12** (cf. [12, A.5.118, Cor. 3]). *If* $\mathbb{G}$ *is a finitely generated extension over* $\mathbb{E}$, *then every subextension* $\mathbb{E} \subseteq \mathbb{F} \subseteq \mathbb{G}$ *of* $\mathbb{G}$ *over* $\mathbb{E}$ *is also finitely generated.*

Proof of Theorem 11. We lead the proof along the lines that will be used also in the proof of Lemma 16 to give bounds for the stabilisation point. First of all, observe that

$$\operatorname{tr\,deg} \mathbb{F}_n \leqslant \operatorname{tr\,deg} \mathbb{Q}(x_1, \ldots, x_k) = k, \quad \text{for all } n.$$

Hence, there is n_1 such that $\operatorname{tr\,deg} \mathbb{F}_{n_1} = \operatorname{tr\,deg} \mathbb{F}_{n_1+1} = \cdots$. Let $\mathbb{F}_\infty := \bigcup_{n=0}^{\infty} \mathbb{F}_n$ and consider the ascending sequence

$$\mathbb{F}_{n_1} \subseteq \mathbb{F}_{n_1+1} \subseteq \cdots \subseteq \mathbb{F}_\infty. \tag{6}$$

We have $\operatorname{tr\,deg} \mathbb{F}_{n_1} = \operatorname{tr\,deg} \mathbb{F}_{n_1+i}$ for all $i > 0$, and, by Lemma 10, $\operatorname{tr\,deg}_{\mathbb{F}_{n_1}} \mathbb{F}_{n_1+i} = 0$, i.e., $\mathbb{F}_{n_1+i}$ is algebraic over $\mathbb{F}_{n_1}$. Moreover, $\mathbb{F}_\infty$ is also algebraic over $\mathbb{F}_{n_1}$ because any element of $\mathbb{F}_\infty$ belongs to some $\mathbb{F}_{n_1+i}$. Since $\mathbb{Q}(x_1, \ldots, x_k) \supseteq \mathbb{F}_{n_1}$ is a finitely generated extension of $\mathbb{F}_{n_1}$ and $\mathbb{F}_{n_1} \subseteq \mathbb{F}_\infty \subseteq \mathbb{Q}(x_1, \ldots, x_k)$ is a subextension, by Lemma 12 we have that $\mathbb{F}_\infty$ is also a finitely generated extension of $\mathbb{F}_{n_1}$. In other words, there are generators $f_1, \ldots, f_m \in \mathbb{F}_\infty$ such that

$$\mathbb{F}_\infty = \mathbb{F}_{n_1}(f_1, \ldots, f_m).$$

Since the generators $f_1, \ldots, f_m$ are algebraic over $\mathbb{F}_{n_1}$, $\mathbb{F}_\infty$ is an algebraic extension of finite degree over $\mathbb{F}_{n_1}$ by Lemma 9. (Concretely, an upper bound for the degree is the product of the degrees of minimal polynomials of the generators $f_1, \ldots, f_m$.) It follows that the sequence in (6) is an ascending sequence of vector subspaces of $\mathbb{F}_\infty$, where we treat $\mathbb{F}_\infty$ as a vector space over $\mathbb{F}_{n_1}$. Since the dimension of $\mathbb{F}_\infty$ as a vector space over $\mathbb{F}_{n_1}$ is finite, this sequence must eventually stabilise at $\mathbb{F}_{n_0}$ for some $n_0 \geqslant n_1$. ◀

We now prove the existence part of Theorem 6 using Theorem 11.

Proof (of the first part of Theorem 6). By Theorem 11, the sequence in (5) stabilises at some $\mathbb{F}_m$, that is,

$$\mathbb{F}_m = \mathbb{F}_{m+1} = \mathbb{F}_m(F^{(1)}_{m+1}(\mathbf{x})).$$

Therefore, we have $F^{(1)}_{m+1}(\mathbf{x}) \in \mathbb{F}_m$. Noting that $\mathbb{F}_m = \mathbb{Q}(F^{(1)}_0(\mathbf{x}), \dots, F^{(1)}_m(\mathbf{x}))$, we see that $F^{(1)}_{m+1}(\mathbf{x})$ can be expressed as a rational function of the generators: there exists a rational function $R \in \mathbb{Q}(y_0, \dots, y_m)$ such that

$$F^{(1)}_{m+1}(\mathbf{x}) = R(F^{(1)}_0(\mathbf{x}), \dots, F^{(1)}_m(\mathbf{x})).$$

We may now apply Lemma 8 to the numerator of the rational function $R(y_0, \dots, y_m) - y_{m+1}$, thus obtaining that

$$F^{(1)}_{n+m+1}(\mathbf{x}) = R(F^{(1)}_n(\mathbf{x}), \dots, F^{(1)}_{n+m}(\mathbf{x})), \quad \text{for every } n \in \mathbb{N}. \qquad \blacktriangleleft$$

4.2 Upper bound

We now move to the second, quantitative part of the proof of Theorem 6: we need to prove that m is bounded from above by $k + k^3 \log(kD)$. For this, we inspect the proof of Theorem 11 in the special case of the chain of extensions (5) given by a ratrec system. The first observation is that the sequence of transcendence degrees stabilises very quickly.

▶ **Lemma 13.** *The transcendence degrees* $\operatorname{tr\,deg} \mathbb{F}_n$ *of the sequence* (5) *stabilise after at most* k *steps.*

Proof. As argued, $\operatorname{tr\,deg} \mathbb{F}_n \leqslant \operatorname{tr\,deg} \mathbb{Q}(x_1, \dots, x_k) = k$ for all n. The next extension $\mathbb{F}_{n+1}$ is obtained by adding a new rational function $F^{(1)}_{n+1}(x_1, \dots, x_k)$ to the previous extension $\mathbb{F}_n$. This immediately shows that $\operatorname{tr\,deg} \mathbb{F}_n \leqslant \operatorname{tr\,deg} \mathbb{F}_{n+1} \leqslant \operatorname{tr\,deg} \mathbb{F}_n + 1$. We argue that if $\operatorname{tr\,deg} \mathbb{F}_{d+1} = \operatorname{tr\,deg} \mathbb{F}_d$ for some d, then the transcendence degree cannot change anymore: $\operatorname{tr\,deg} \mathbb{F}_d = \operatorname{tr\,deg} \mathbb{F}_{d+1} = \operatorname{tr\,deg} \mathbb{F}_{d+2} = \cdots$. Note that this will conclude the proof, because then the transcendence degree can increase at most k times before eventually stabilising.

Since $\operatorname{tr\,deg} \mathbb{F}_{d+1} = \operatorname{tr\,deg} \mathbb{F}_d$, it follows that $F^{(1)}_{d+1}(\mathbf{x})$ is algebraic over $\mathbb{F}_d$, which means that it satisfies $P(F^{(1)}_{d+1}(\mathbf{x})) = 0$ for some nonzero polynomial $P(x) \in \mathbb{F}_d[x]$. By clearing out denominators, there is a nonzero polynomial $Z(y_0, \dots, y_{d+1}) \in \mathbb{Q}[y_0, \dots, y_{d+1}]$ such that

$$Z(F^{(1)}_0(\mathbf{x}), \dots, F^{(1)}_{d+1}(\mathbf{x})) = 0. \tag{7}$$

By Lemma 8 we have

$$Z(F^{(1)}_n(\mathbf{x}), \dots, F^{(1)}_{n+d+1}(\mathbf{x})) = 0 \quad \text{for every } n \in \mathbb{N}.$$

This means that $F^{(1)}_{n+d+1}(\mathbf{x})$ is algebraic over $\mathbb{F}_{n+d}$, implying

$$\operatorname{tr\,deg} \mathbb{F}_{n+d+1} = \operatorname{tr\,deg} \mathbb{F}_{n+d}(F^{(1)}_{n+d+1}(\mathbf{x})) = \operatorname{tr\,deg} \mathbb{F}_{n+d}.$$

This concludes the proof. $\blacktriangleleft$

Note that even when the transcendence degrees of the fields in (5) stabilise, it may still take several further steps until the fields themselves eventually stabilise. We will later give an example that this may indeed happen.

We are left with estimating the degrees of field extensions after the transcendence degree in the chain (5) stabilises. For this, we use the following two results.

▶ **Lemma 14** (cf. [25, Lemma 3.4]). *Let $f \in \mathbb{Q}(x_1, \ldots, x_k)$ be algebraic over*

$$\mathbb{F}_n = \mathbb{Q}(F_0(x_1, \ldots, x_k), \ldots, F_n(x_1, \ldots, x_k)).$$

Then there is a polynomial $Z(x) \in \mathbb{F}_n[x]$ of degree at most $\deg F_0 \cdots \deg F_n$ s.t. $Z(f) = 0$.

▶ **Lemma 15** (cf. [26, Exercise III.A.2]). *Let $\mathbb{Q} \subseteq \mathbb{F} \subseteq \mathbb{Q}(x_1, \ldots, x_k)$ be a subextension of $\mathbb{Q}(x_1, \ldots, x_k)$ over $\mathbb{Q}$ of transcendence degree $r := \operatorname{tr.deg}_{\mathbb{Q}} \mathbb{F}$. Then there are (algebraically independent) rational functions $f_1, \ldots, f_r \in \mathbb{Q}(x_1, \ldots, x_k)$ such that $\mathbb{F} \subseteq \mathbb{Q}(f_1, \ldots, f_r)$.*

▶ **Lemma 16.** *The sequence* (5) *eventually stabilises after at most $k + k^3 \log(kD)$ steps.*

Proof. By Lemma 13, there exists $j_1 \leqslant k$ such that

$$r := \operatorname{tr.deg} \mathbb{F}_{j_1} = \operatorname{tr.deg} \mathbb{F}_j \quad \text{for all } j \geqslant j_1.$$

In particular, all field extensions $\mathbb{F}_j$ for $j \geqslant j_1$ are algebraic over $\mathbb{F}_{j_1}$. As in the proof of Theorem 11, consider the field extension $\mathbb{F}_\infty = \bigcup_{j=0}^{\infty} \mathbb{F}_j$ over $\mathbb{F}_{j_1}$, which is algebraic. In particular, $\operatorname{tr.deg} \mathbb{F}_\infty = r$. By Lemma 15, there are rational functions $f_1, \ldots, f_r \in \mathbb{Q}(x_1, \ldots, x_k)$ s.t. $\mathbb{F}_\infty \subseteq \mathbb{Q}(f_1, \ldots, f_r)$. Since $\mathbb{Q}(f_1, \ldots, f_r)$ has the same transcendence degree as F_{j_1}, i.e. $\operatorname{tr.deg} \mathbb{Q}(f_1, \ldots, f_r) = r$, it follows that the f_i's are algebraic over $\mathbb{F}_{j_1}$. By Lemma 14, the degree of each $F_{j_1}[f_i]$ is at most $\deg F_{j_1}[f_i] \leqslant \deg F_0 \cdots \deg F_{j_1}$ over $\mathbb{F}_{j_1}$. It follows that $\mathbb{Q}(f_1, \ldots, f_r)$ is an algebraic extension of degree at most $d = (\deg F_0 \cdots \deg F_{j_1})^r$ over $\mathbb{F}_{j_1}$. Thus the chain

$$\mathbb{F}_{j_1} \subseteq \mathbb{F}_{j_1+1} \subseteq \mathbb{F}_{j_1+2} \subseteq \cdots$$

is such that the degree of any $\mathbb{F}_{j_1+t}$ over $\mathbb{F}_{j_1}$ is at most d. Recall that all extensions in this chain are algebraic. We show that it stabilises after at most $\log d$ steps. Assume that for some $t \geqslant 0$ we have $\mathbb{F}_{j_1+t} = \mathbb{F}_{j_1+t+1}$, that is $F^{(1)}_{j_1+t+1}(\mathbf{x}) \in \mathbb{F}_{j_1+t}$. Thus, there is a rational function $R \in \mathbb{F}_{j_1}(y_1, \ldots, y_t)$ such that $F^{(1)}_{j_1+t+1}(\mathbf{x}) = R(F^{(1)}_{j_1+1}(\mathbf{x}), \ldots, F^{(1)}_{j_1+t}(\mathbf{x}))$. Then by applying Lemma 8 to the numerator of $R(y_1, \ldots, y_t) - y_{t+1}$, we may express $F^{(1)}_{j_1+n+t+1}(\mathbf{x})$ as a rational function of $F^{(1)}_{j_1+n+1}(\mathbf{x}), \ldots, F^{(1)}_{j_1+n+t}(\mathbf{x})$, i.e., elements of $\mathbb{F}_{j_1+n+t}$. Hence an equality in the field chain implies stabilisation at this point.

Since the degree grows at each step before stabilisation and the degree is multiplicative, the chain stabilises after at most $\log d$ steps. Indeed, in every step before stabilisation the degree is multiplied by at least two. Thus the chain stabilises at $\mathbb{F}_{j_0}$ for some $j_0 \leqslant j_1 + \log((\deg F_0 \cdots \deg F_{j_1})^r)$. By Lemma 7 and since $j_1 \leqslant k$ and $r \leqslant k$, we have, as required,

$$\begin{aligned} j_0 &\leqslant j_1 + r \log(\deg F_0 \cdots \deg F_{j_1}) \\ &\leqslant k + k \log((kD)^0 \cdots (kD)^k) \\ &\leqslant k + k^3 \log(kD). \end{aligned}$$

◀

We are ready to provide the proof of the quantitative bound promised in Theorem 6.

Proof (of the second part of Theorem 6). It suffices to observe that m in the proof of the first part of Theorem 6 can be bounded by $k + k^3 \log(kD)$ thanks to Lemma 16. ◀

We finish this section by giving an example that shows that in the proof of Lemma 16, it may happen that $j_1 < j_0$, that is, after the stabilisation of the transcendence degree, there can be several non-trivial algebraic extensions until the fields themselves stabilise. Consider the polyrec system

$$\begin{cases} u^{(1)}_{n+1} &= (u^{(1)}_n)^2 + (u^{(2)}_n)^2, \\ u^{(2)}_{n+1} &= u^{(1)}_n + u^{(2)}_n. \end{cases}$$

We have $F^{(1)}_0 = x_1, F^{(2)}_0 = x_2$, then $F^{(1)}_1 = x_1^2 + x_2^2, F^{(2)}_1 = x_1 + x_2$ and $F^{(1)}_2 = (x_1^2 + x_2^2)^2 + (x_1 + x_2)^2$. The chain (5) starts with

$$\mathbb{Q} \subseteq \mathbb{F}_0 = \mathbb{Q}(x_1) \subseteq \mathbb{F}_1 = \mathbb{F}_0(x_1^2 + x_2^2) = \mathbb{Q}(x_1, x_2^2) \subseteq \mathbb{F}_2.$$

Note that $\operatorname{tr\,deg} \mathbb{F}_0 = 1$ and $\operatorname{tr\,deg} \mathbb{F}_1 = 2$, which is the maximum value. However, the next extension $\mathbb{F}_1 \subseteq \mathbb{F}_2 = \mathbb{F}_1(F^{(1)}_2) = \mathbb{F}_1(x_1x_2)$ is non-trivial, because x_1x_2 does not belong to $\mathbb{Q}(x_1, x_2^2)$. In fact, it is algebraic of degree 2.

4.3 Obstacles towards the zeroness problem for polyrec sequences

Theorem 6 suggests the following algorithm for deciding zeroness of a polyrec sequence $\mathbf{u}$. Suppose the dimension of $\mathbf{u}$ is k and the degree is D. We compute the first $p + 2$ entries of $\mathbf{u}$, where $p = k + k^3\lceil\log(kD)\rceil$, and we verify whether all of them are zero. Obviously, if one of them is non-zero, then $\mathbf{u}$ is non-zero. Otherwise, by Theorem 6, we expect that there is a rational function $R(y_0, \ldots, y_m)$ for some $m \leqslant p$ such that

$$u_{n+m+1} = R(u_n, u_{n+1}, \ldots, u_{n+m}) \qquad \text{for all } n \in \mathbb{N}. \tag{8}$$

In particular,

$$0 = u_{m+1} = R(u_0, \ldots, u_m) = R(0, \ldots, 0).$$

Consequently,

$$u_{m+2} = R(u_1, \ldots, u_{m+1}) = R(0, \ldots, 0) = 0,$$

and a straightforward induction shows that $u_n = 0$ for all $n \in \mathbb{N}$. So we can declare that $\mathbf{u}$ is the zero sequence.

The reasoning above is incorrect for the following reason. By Theorem 6, there is a rational function $R \in \mathbb{Q}(y_0, \ldots, y_m)$ such that (8) holds when both sides are treated symbolically, as rational functions over a set of k variables $\mathbf{x}$ that denote the vector of initial entries of the polyrec system defining $\mathbf{u}$. However, R is a rational function, hence when the variables are substituted with actual entries of the sequence $\mathbf{u}$, we may get an accidental 0 in the denominator of the right hand side. In other words, assertion (8) may be incorrect due to the right hand side being ill-defined, which renders the remainder of the reasoning flawed. To exemplify the problem we now present a case where this situation actually occurs.

Fix some $d \in \mathbb{N}$, and let

$$P(x) = x(x-1)\ldots(x-d+1).$$

Define the sequence $\mathbf{u}$ by setting

$$u_n = P(n) \qquad \text{for all } n \in \mathbb{N}.$$

It is straightforward to see that $\mathbf{u}$ is polyrec of dimension 2 and degree d: one can simply use one auxiliary sequence $\mathbf{v}$ with $v_n = n$.

Observe that if instead of setting $v_0 = 0$, we set $v_0 = x$ for a formal variable x, the same polyrec system defines a sequence of polynomials $\widehat{\mathbf{u}}$ over x defined as

$$\widehat{u}_n = P(x+n) \qquad \text{for all } n \in \mathbb{N}.$$

(Here, we also set initial condition $\widehat{u}_0 = P(x)$.) Now, we may apply the reasoning behind Theorem 6 to find the rational function $R(y_0, y_1) \in \mathbb{Q}(y_0, y_1)$, defined as

$$R(y_0, y_1) = y_1 \cdot \frac{(d+1) \cdot y_1 - y_0}{y_1 + (d-1) \cdot y_0},$$

such that

$$\widehat{u}_{n+2} = R(\widehat{u}_n, \widehat{u}_{n+1}) \qquad \text{for all } n \in \mathbb{N}.$$

This, however, should be regarded as an equality of two rational functions over the variable x, which means that we cannot infer that

$$u_{n+2} = R(u_n, u_{n+1}) \qquad \text{for all } n \in \mathbb{N},$$

because the right hand side can be undefined for specific values; and indeed, $R(0,0)$ is undefined. The flawed reasoning from the beginning of this section would suggest that in order to verify the zeroness of $\mathbf{u}$, it suffices to check that the first three entries of $\mathbf{u}$ are zero. However, we have $u_0 = u_1 = \cdots = u_{d-1} = 0$ and $u_d = d! \neq 0$, so the algorithm would provide an incorrect answer.

Notice that if we had a promise that we never encounter a division by zero when recursively applying (8) from the given initial conditions, then the naïve zeroness algorithm presented at the beginning of the section would be sound. (The naïve algorithm is complete even without the promise.) However, deciding whether no division by zero occurs is essentially the Skolem problem for polyrec sequences, which, as mentioned in the introduction, is a long-standing open problem.

We are hopeful that the problem with accidentally hitting a singularity of R when starting from a polyrec sequence, as present in the example above, can somehow be circumvented, hence we state the following conjecture.

▶ **Conjecture 17.** *There is an elementary function $g\colon \mathbb{N} \to \mathbb{N}$ such that the following holds. Suppose $\mathbf{u}$ is a polyrec sequence of dimension at most N and degree at most N such that $u_n = 0$ for all $n \leqslant g(N)$. Then $u_n = 0$ for all $n \in \mathbb{N}$.*

Note that a positive resolution to Conjecture 17 would immediately imply that the complexity of the zeroness problem for polyrec sequences is elementary.

5 Zeroness for polyrec is PSPACE-hard

In this section we discuss the following lower bound. The technical part of this section can be found in the appendix.

▶ **Theorem 18.** *For every fixed field $\mathbb{F}$, the zeroness problem for polyrec sequences over $\mathbb{F}$ is PSPACE-hard.*

The lower bound claimed in the introduction follows from the theorem above by taking $\mathbb{F} = \mathbb{Q}$. Note also that together with Theorem 19 below, we can conclude that the problem is actually PSPACE-complete for every fixed *finite* field $\mathbb{F}$.

▶ **Theorem 19.** *For every fixed finite field $\mathbb{F}$, the zeroness problem for polyrec sequences over $\mathbb{F}$ is in PSPACE.*

Proof. Let m be the cardinality of $\mathbb{F}$; note that m is a fixed constant. A standard periodicity argument, e.g. as in the proof of [13, Theorem 4.1], shows that if $\mathbf{u}$ is a polyrec sequence of dimension k, then it is zero if, and only if, it is zero for the first m^k steps. We can check the latter condition by storing in memory a k-tuple of values and computing the first m^k values of the sequence, which takes an amount of space which is polynomial in k. ◀

6 Conclusion

We believe that ratrec is a natural class of sequences with various promising questions deserving further investigation. Questions about decision problems are more natural for polyrec sequences due to their connection to polynomial automata and the issues with division by 0 in ratrec discussed in the introduction. Nevertheless, as discussed in this paper, understanding the properties of ratrec might lead to concrete complexity results for polyrec. The most natural problem for future work is to overcome the obstacles discussed in Section 4.3.

References

1 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 13–22, 2013. doi:10.1109/LICS.2013.65.

2 Corentin Barloy and Lorenzo Clemente. Bidimensional linear recursive sequences and universality of unambiguous register automata. In Markus Bläser and Benjamin Monmege, editors, *Proc. of STACS'21*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

3 Corentin Barloy, Nathanaël Fijalkow, Nathan Lhote, and Filip Mazowiecki. A robust class of linear recurrence sequences. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, pages 9:1–9:16, 2020. doi:10.4230/LIPIcs.CSL.2020.9.

4 M. Benedikt, T. Duff, A. Sharad, and J. Worrell. Polynomial automata: Zeroness and applications. In *Proc. of LICS'17*, pages 1–12, June 2017. doi:10.1109/LICS.2017.8005101.

5 Vincent D. Blondel and Natacha Portier. The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. *Linear Algebra and its Applications*, 351–352:91–98, 2002. Fourth Special Issue on Linear Systems and Control. doi:10.1016/S0024-3795(01)00466-9.

6 Adrien Boiret, Radoslaw Piórkowski, and Janusz Schmude. Reducing transducer equivalence to register automata problems solved by "Hilbert Method". In Sumit Ganguly and Paritosh Pandya, editors, *Proc. of FSTTCS'18*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2018.48.

7 Mikołaj Bojańczyk. The Hilbert method for transducer equivalence. *ACM SIGLOG News*, 6(1):5–17, February 2019. doi:10.1145/3313909.3313911.

8 Mikołaj Bojańczyk and Wojciech Czerwiński. An automata toolbox, February 2018. URL: https://www.mimuw.edu.pl/~bojan/paper/automata-toolbox-book.

9 Mikołaj Bojańczyk and Janusz Schmude. Some remarks on deciding equivalence for graph-to-graph transducers. In Javier Esparza and Daniel Kráľ, editors, *Proc. of MFCS'20*, volume 170 of *LIPIcs*, pages 19:1–19:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2020.19.

10 Alin Bostan, Arnaud Carayol, Florent Koechlin, and Cyril Nicaud. Weakly-unambiguous parikh automata and their link to holonomic series. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. of ICALP'20*, volume 168 of *LIPIcs*, pages 114:1–114:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.114.

11 Alin Bostan and Antonio Jiménez-Pastor. On the exponential generating function of labelled trees. *Comptes Rendus. Mathématique*, 358(9-10):1005–1009, 2020.

12 N. Bourbaki. *Algebra II*. Elements of Mathematics. Springer Verlag Berlin Heidelberg, 2003.

13 Michaël Cadilhac, Filip Mazowiecki, Charles Paperman, Michał Pilipczuk, and Géraud Sénizergues. On polynomial recursive sequences. *Theory of Computing Systems*, pages 1–22, 2021.

14 N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier, 1963. doi:10.1016/S0049-237X(08)72023-8.

15 Lorenzo Clemente. On the complexity of the universality and inclusion problems for unambiguous context-free grammars. In Laurent Fribourg and Matthias Heizmann, editors, Proceedings 8th International Workshop on *Verification and Program Transformation* and 7th Workshop on *Horn Clauses for Verification and Synthesis,* Dublin, Ireland, 25-26th April 2020, volume 320 of *EPTCS*, pages 29–43. Open Publishing Association, 2020. doi:10.4204/EPTCS.320.2.

16 J. Denef and L. Lipshitz. Decision problems for differential equations. *Journal of Symbolic Logic*, 54(3):941–950, 1989. doi:10.2307/2274755.

17 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

18 David S. Dummit and Richard M. Foote. *Abstract Algebra*. Wiley, 3rd edition, 2003. URL: http://gen.lib.rus.ec/book/index.php?md5=36e6532b72807b9ef6b27e52e8c62ccc.

19 Vojtěch Forejt, Petr Jančar, Stefan Kiefer, and James Worrell. Language equivalence of probabilistic pushdown automata. *Information and Computation*, 237:1–11, 2014.

20 Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumäki. Skolem's problem - on the border between decidability and undecidability, 2005.

21 T. Harju and J. Karhumäki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78(2):347–355, 1991. doi:10.1016/0304-3975(91)90356-7.

22 Manuel Kauers and Peter Paule. *The Concrete Tetrahedron - Symbolic Sums, Recurrence Equations, Generating Functions, Asymptotic Estimates*. Texts & Monographs in Symbolic Computation. Springer, 2011. doi:10.1007/978-3-7091-0445-3.

23 Leonard Lipshitz. D-finite power series. *Journal of Algebra*, 122(2):353–373, 1989. doi:10.1016/0021-8693(89)90222-6.

24 Kurt Mahler. *Lectures on Transcendental Numbers*. Lecture Notes in Mathematics 546. Springer-Verlag Berlin Heidelberg, 1st edition, 1976.

25 J. Müller-Quade and R. Steinwandt. Basic algorithms for rational function fields. *Journal of Symbolic Computation*, 27(2):143–170, 1999. doi:10.1006/jsco.1998.0246.

26 Masayoshi Nagata. *Theory of Commutative Fields*. Translations of Mathematical Monographs, Vol. 125. American Mathematical Society, 1993. URL: http://gen.lib.rus.ec/book/index.php?md5=249c3cba331671e0fd3c692d01b54b94.

27 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *ACM SIGLOG News*, 2(2):4–13, April 2015. doi:10.1145/2766189.2766191.

28 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

29 Azaria Paz. *Introduction to probabilistic automata*. Academic Press, 1971.

30 Arto Salomaa and Marti Soittola. *Automata-theoretic aspects of formal power series*. Texts and Monographs in Computer Science. Springer, 1978. doi:10.1007/978-1-4612-6264-0.

31 Bruno Salvy. D-finiteness: Algorithms and applications. In *Proc. of ISAAC'05*, pages 2–3, New York, NY, USA, 2005. ACM. doi:10.1145/1073884.1073886.

32 Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2–3):245–270, 1961.

33 Helmut Seidl, Sebastian Maneth, and Gregor Kemper. Equivalence of deterministic top-down tree-to-string transducers is decidable. *J. ACM*, 65(4):21:1–21:30, April 2018. doi:10.1145/3182653.

34 Richard P. Stanley and Sergey Fomin. *Enumerative combinatorics*, volume 2 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 1 edition, 2001.

35 Joris van der Hoeven. Computing with d-algebraic power series. *Applicable Algebra in Engineering, Communication and Computing*, 30(1):17–49, 2019. doi:10.1007/s00200-018-0358-y.

36 Tzeng Wen-Guey. On path equivalence of nondeterministic finite automata. *Information Processing Letters*, 58(1):43–46, 1996. doi:10.1016/0020-0190(96)00039-7.

37 James Worrell. Revisiting the equivalence problem for finite multitape automata. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Proc. of ICALP'13*, pages 422–433, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

38 Doron Zeilberger. A holonomic systems approach to special functions identities. *Journal of Computational and Applied Mathematics*, 32(3):321–368, 1990. doi:10.1016/0377-0427(90)90042-X.

A Zeroness for polyrec is PSPACE-hard

In order to speak about computational aspects of poly-rec sequences, we need to fix how they are encoded on input. For robustness, we choose to use arithmetic circuits. Formally, for a fixed field $\mathbb{F}$, a polynomial $P \in \mathbb{F}[x_1, \ldots, x_k]$ is encoded by a circuit C that may use the following gates:

- binary addition and multiplication gates;
- nullary input gates, bijectively labelled with variables $x_1, \ldots, x_k$; and
- nullary constant gates, each labelled with an element of $\mathbb{F}$.

Note that subtraction can be emulated using addition and multiplication by the constant -1. One of the gates is designated as the output gate. Given a valuation of variables with elements of $\mathbb{F}$, the values of the gates can be computed as expected, and the value yielded by the circuit C is the one computed for the output gate.

A.1 Extended polyrec sequences

In the reductions leading to the lower bound of Theorem 18 it is convenient to construct polyrec sequences according to a definition slightly more general than what we allowed in Definition 4. Namely, the definition of an *extended polyrec system* is the same as before, except that we generalize the format of the ith equation $u^{(i)}_{n+1} = P_i(\cdots)$ by allowing $u^{(i)}_{n+1}$ to additionally depend on $u^{(1)}_{n+1}, \ldots, u^{(i-1)}_{n+1}$. Thus, the ith equation takes the form:

$$u^{(i)}_{n+1} = P_i(u^{(1)}_n, \ldots, u^{(k)}_n, u^{(1)}_{n+1}, \ldots, u^{(i-1)}_{n+1}), \tag{9}$$

where now P_i is a polynomial in $k + i - 1$ variables. This more relaxed definition will help focus on the important aspects of the reduction presented in the rest of this section. The following lemma shows that the modification does not affect the complexity of the zeroness problem.

▶ **Lemma 20.** *Suppose* $\mathbf{u}$ *is a sequence defined by an extended polyrec system* S *of dimension* k*, where each polynomial* P_i *is represented by circuit* C_i*. Then given the circuits* C_i*, one can in polynomial time construct a circuit* C *that represents a polyrec system* S' *of dimension* k *that also defines* $\mathbf{u}$ *(with the same initial condition as* S*).*

Proof. Let the input gates of circuit C_i be labelled with $x_1, \ldots, x_k, z_1, \ldots, z_{i-1}$, where variables $z_1, \ldots, z_{i-1}$ respectively correspond to the values $u_{n+1}^{(1)}, \ldots, u_{n+1}^{(i-1)}$ in (9). Construct the circuit C from the union of circuits $C_1, \ldots, C_k$ by performing the following operations for each $i \in \{1, \ldots, k\}$:

- Fuse all input gates labelled x_i in circuits $C_1, \ldots, C_k$ into a single input gate labelled x_i.
- Fuse the output gate of C_i with all input gates labelled z_i in circuits $C_{i+1}, \ldots, C_k$.

The output gates of C are the output gates of $C_1, \ldots, C_k$. (Formally, we assumed that output gates must have fan-out 0, but this can be easily obtained by making a copy of each output gate.) It is straightforward to verify that the polyrec system S' that C represents defines the same k-tuple of sequences as S under the same initial condition. ◀

A.2 Reduction

We now proceed to the proof of Theorem 18. Let us fix the field $\mathbb{F}$; in the reduction we will use only two constants from $\mathbb{F}$, namely 0 and 1. We reduce from the validity problem for Quantified Boolean Formulas (QBF), which is known to be PSPACE-complete (see, e.g., [28, Theorem 19.1]). Recall that the QBF validity problem amounts to determine whether a given QBF of the form

$$\psi = \exists x_1 \; \forall x_2 \; \ldots \; Q_k x_k \; \varphi(x_1, \ldots, x_k) \tag{10}$$

is true, where $\varphi(x_1, \ldots, x_k)$ is quantifier-free, the variables with odd indices are quantified existentially, the remaining variables are quantified universally, and Q_k is either $\exists$ or $\forall$ depending on the parity of k. Hence, we are given a QBF ψ and we wish to construct, in polynomial time, a polyrec system S and its initial condition that define a sequence $\mathbf{u}$ over $\mathbb{F}$ such that the zeroness of $\mathbf{u}$ is equivalent to the invalidity of ψ. By Lemma 20, it suffices to construct an extended polyrec system S with this property, where each polynomial P_i involved is represented by a separate circuit. In the following, the *size* of an extended polyrec system is the total size of its representation through circuits, which is constructed implicitly.

In the reduction it will be convenient to consider formulas obtained by fixing the truth values of a subset of the bound variables of ψ. For every $i \in \{0, \ldots, k\}$ and $c_{i+1}, \ldots, c_k \in \{0, 1\}$ we define the formula

$$\psi|_{c_{i+1},\ldots,c_k} = \exists x_1 \forall x_2 \cdots Q_i x_i \; \varphi(x_1, \ldots, x_i, c_{i+1}, \ldots, c_k),$$

where Q_i is either $\exists$ or $\forall$ depending on the parity of i. In particular, for $i = k$ we get back ψ, and for $i = 0$ the formula $\psi|_{c_1,\ldots,c_k}$ reduces to the truth value of $\varphi(c_1, \ldots, c_k)$. We encode a quantifier Boolean formula φ into a polynomial P_φ using the following simulation of Boolean operators $\neg$, $\wedge$ and $\vee$ by arithmetic operations:

$$\begin{aligned} P_x &= x, \\ P_{\neg\varphi} &= 1 - P_\varphi, \\ P_{\varphi\wedge\psi} &= P_\varphi \cdot P_\psi, \\ P_{\varphi\vee\psi} &= P_{\neg(\neg\varphi\wedge\neg\psi)}. \end{aligned} \tag{11}$$

For example, $\tau(x, y) = (x \wedge y) \vee \neg y$ is encoded as $P_\tau(x, y) = 1 - (1 - xy)y$. The following straightforward claim shows that with the standard interpretation of 1 and 0 representing true, resp., false, such polynomials evaluate as expected. Note that this claims holds in any fixed field.

▷ Claim 21. Let $\tau(x_1, \ldots, x_k)$ be a Boolean formula and $P_\tau(x_1, \ldots, x_k)$ its corresponding polynomial. For every $c_1, \ldots, c_k \in \{0, 1\}$ we have $P_\tau(c_1, \ldots, c_k) \in \{0, 1\}$ and

$$(c_1, \ldots, c_k) \models \tau \iff P_\tau(c_1, \ldots, c_k) = 1.$$

To ease the notation we will directly write formulas as polynomials; for instance, $P_\tau(x, y) = (x \wedge y) \vee \neg y$. All sequences in this section will be over $\{0, 1\}$ and the involved polynomials will be of the form P_τ.

Sequences $\mathbf{c^1}, \ldots, \mathbf{c^k}$. The truth valuations of variables $x_1, \ldots, x_k$ will be encoded by sequences $\mathbf{c}^1, \ldots, \mathbf{c}^k$, where for every i and n we have

$$c^i_n = \begin{cases} 0 & \text{if } n \bmod 2^i \text{ is less than } 2^{i-1}, \\ 1 & \text{otherwise.} \end{cases} \tag{12}$$

For example, the first eight values of $\mathbf{c}^1, \mathbf{c}^2, \mathbf{c}^3$ are

$$\begin{array}{lllllllll} \mathbf{c}^1 & = & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \mathbf{c}^2 & = & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \mathbf{c}^3 & = & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array} .$$

▷ Claim 22. For every $i \geqslant 1$, the sequence $\mathbf{c}^i$ is definable by an extended polyrec system over $\mathbb{F}$ of size polynomial in i.

Proof. We proceed by induction on i. For $i = 1$, by definition we have $c^1_n = 1 - c^1_{n-1}$, and thus we let

$$c^1_n = P(c^1_{n-1}), \text{ with } P(x) = \neg x. \tag{13}$$

Now, suppose $i > 1$ and we have defined $\mathbf{c}^{i-1}$. We start by proving the following equality for every $n > 1$

$$c^i_n = \begin{cases} 1 - c^i_{n-1} & \text{if } c^{i-1}_{n-1} = 1 \text{ and } c^{i-1}_n = 0; \\ c^i_{n-1} & \text{otherwise.} \end{cases} \tag{14}$$

Notice that $\mathbf{c}^i$ is periodic with period 2^i, i.e., $c^i_n = c^i_{n+2^i}$ for all n. Thus it suffices to prove (14) for $n \in \{1, \ldots, 2^i\}$. By definition, $c^{i-1}_{n-1} = 1$ and $c^{i-1}_n = 0$ hold precisely for two values of $n \in \{1, \ldots, 2^i\}$, namely for $n = 2^{i-1}$ and $n = 2^i$. Thus (14) is proved since $c^i_n = 0$ for $0 \leqslant n < 2^{i-1}$; $c^i_n = 1$ for $2^{i-1} \leqslant n < 2^i$; and $c^i_{2^i} = 0$.

Using (14) one can determine c^i_n given c^i_{n-1}, c^{i-1}_{n-1}, and c^{i-1}_n:

$$c^i_n = Q(c^i_{n-1}, c^{i-1}_{n-1}, c^{i-1}_n), \tag{15}$$

where $Q(x, y, z) = (\neg x \wedge (y \wedge \neg z)) \vee (x \wedge (\neg y \vee z))$. This follows from (14) and from the fact that $y \wedge \neg z$ and $\neg y \vee z$ are mutually exclusive formulas encoding the "if" condition in (14). It is clear that the constructed extended polyrec system is of size polynomial in i. ◁

Sequences $\mathbf{d^0}, \ldots, \mathbf{d^k}$. We define sequences $\mathbf{d}^0, \ldots, \mathbf{d}^k$, where for any $i \geqslant 0$ we have:

$$d^i_0 = 0, \quad d^i_n = \begin{cases} 0 & \text{if } 2^i \nmid n \\ \left[\!\left[\psi|_{c^{i+1}_{n-1}, \ldots, c^k_{n-1}} \right]\!\right] & \text{otherwise,} \end{cases} \tag{16}$$

where for a closed formula ξ (i.e., with no free variables) $[\![\xi]\!]$ is 1 if ξ is true and 0 otherwise. Notice that the formula depends on $\mathbf{c}^1, \ldots, \mathbf{c}^k$. Since $\mathbf{d}^k$ is the zero sequence if, and only if, ψ is false, it suffices to show that each $\mathbf{d}^i$ can be defined by an extended polyrec system of polynomial size.

We proceed by induction on i. In the base case $i = 0$,

$$d_n^0 = P_\varphi(c_{n-1}^1, \ldots, c_{n-1}^k), \tag{17}$$

where P_φ is the polynomial obtained from the quantifier-free formula φ according to the rules in (11). (Notice that P_φ can be represented by an arithmetic circuit of size polynomial in the size of φ – this is where we use the conciseness of representation using circuits.) This fulfills the conditions in (16) since, for $n > 0$, $d_n^0 = 1$ if $(c_{n-1}^1, \ldots, c_{n-1}^k) \models \varphi$ and $d_n^0 = 0$ otherwise.

Now, fix $i \geqslant 1$ and suppose that $\mathbf{d}^{i-1}$ is defined. The goal is to define $\mathbf{d}^i$. Recall that if i is odd then x_i is quantified existentially, and otherwise x_i is quantified universally.

▷ Claim 23. Let $\circledast_i = \vee$ if i is odd and $\circledast_i = \wedge$ if i is even. For every $n > 0$ and $0 < i \leqslant k$, we have

$$d_n^i = \begin{cases} d_n^{i-1} \circledast_i d_{n-2^{i-1}}^{i-1} & \text{if } 2^i \mid n \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

Proof. We may focus only on the case $2^i \mid n$. Since x_i is quantified according to the parity of i, we have

$$\psi|_{c_{n-1}^{i+1},\ldots,c_{n-1}^k} = \psi|_{0,c_{n-1}^{i+1},\ldots,c_{n-1}^k} \circledast_i \psi|_{1,c_{n-1}^{i+1},\ldots,c_{n-1}^k}.$$

We claim that

$$d_n^{i-1} = \psi|_{1,c_{n-1}^{i+1},\ldots,c_{n-1}^k} \quad \text{and} \quad d_{n-2^{i-1}}^{i-1} = \psi|_{0,c_{n-1}^{i+1},\ldots,c_{n-1}^k}.$$

By (12) and the fact that $2^i \mid n$, we get $c_{n-1}^i = c_{2^i-1}^i = 1$, which proves the first equation. For the second equation, we observe that $c_{(n-1)-2^{i-1}}^i = c_{2^{i-1}-1}^i = 0$ and that $c_{(n-1)-2^{i-1}}^j = c_{n-1}^j$ for all $j > i$. The latter assertion readily follows from $2^i \mid n$ and (12). ◁

As an immediate consequence of Claim 23, we can write

$$d_n^i = S(d_n^{i-1}, d_{n-2^{i-1}}^{i-1}, c_{n-1}^{i-1}, c_n^{i-1}), \tag{19}$$

where $S(x, y, z, t) = (x \circledast_i y) \wedge (z \wedge \neg t)$ (by recalling that $c_{n-1}^{i-1} = 1, c_n^{i-1} = 0$ holds if, and only if, $2^i \mid n$, where $n > 0$). The issue with this recursive definition is that it requires access to the value $d_{n-2^{i-1}}^{i-1}$, which in general is not allowed in a polyrec system for $i \geqslant 2$ (not even in the extended variant). This will be addressed in the next section by introducing the last family of recursive sequences.

Sequences $\mathbf{f^0}, \ldots, \mathbf{f^{k-1}}$. For every $1 \leqslant i \leqslant k$, the sequence $\mathbf{f}^{i-1}$ is defined as

$$f_n^{i-1} = \begin{cases} 0 & \text{if } n \bmod 2^i \text{ is less than } 2^{i-1} \\ d_m^{i-1} & \text{otherwise,} \end{cases}$$

where $m \leqslant n$ is the unique number such that $n - m < 2^{i-1}$ and $2^{i-1} \mid m$. Thus, $\mathbf{f}$ is divided into blocks of length 2^{i-1} of equal elements, where every other block is either filled with zeros, or its value is determined by the value of an appropriate entry d_m^{i-1}. Observe that in particular, if $2^i \mid n$ then $f_{n-1}^{i-1} = d_{n-2^{i-1}}^{i-1}$. Thus, intuitively, the sequence $\mathbf{f}^{i-1}$ is a "memory" that allows us to store the relevant value of $\mathbf{d}^{i-1}$ from $2^{i-1} - 1$ steps back.

We now proceed to defining sequences $\mathbf{f}^0, \ldots, \mathbf{f}^{k-1}$ using polyrec systems. Observe that $f_0^{i-1} = 0$ and for $n > 0$, we can write

$$f_n^{i-1} = \begin{cases} d_n^{i-1} & \text{if } c_{n-1}^{i-1} = 0 \text{ and } c_n^{i-1} = 1; \\ 0 & \text{if } c_{n-1}^{i-1} = 1 \text{ and } c_n^{i-1} = 0; \\ f_{n-1}^{i-1} & \text{otherwise.} \end{cases} \tag{20}$$

Notice that the value of f_n^{i-1} is copied from f_{n-1}^{i-1} unless c_{n-1}^{i-1}, c_n^{i-1} differ. To conclude, recall from (12) that this happens if, and only if, $2^{i-1} \mid n$.

▷ Claim 24. For every $i \geqslant 1$, the sequences $\mathbf{d}^i$ and $\mathbf{f}^{i-1}$ are definable by extended polyrec systems over $\mathbb{F}$ of size polynomial in i and the size of φ.

Proof. Using (20), we may write $\mathbf{f}^{i-1}$ as an extended polyrec sequence $f_0^{i-1} = 0$ and, for $n > 0$,

$$f_n^{i-1} = R(c_{n-1}^{i-1}, c_n^{i-1}, d_n^{i-1}, f_{n-1}^{i-1}), \tag{21}$$

where

$$R(x, y, z, t) = (z \wedge (\neg x \wedge y)) \vee (t \wedge ((x \wedge y) \vee (\neg x \wedge \neg y))).$$

In turn, this allows us to rewrite (19) as

$$d_n^i = S(d_n^{i-1}, f_{n-1}^{i-1}, c_{n-1}^{i-1}, c_n^{i-1}), \tag{22}$$

where S was defined in (19). Note that (21) and (22) are in the extended polyrec format provided that we write the equations for the $\mathbf{f}^i$'s after the equations for the $\mathbf{d}^i$'s, and the latter after the equations for $\mathbf{c}^i$'s (in order to avoid creating a cyclic dependency). In other words, the final extended polyrec system consists of equations (15), followed by (22), and followed by (21), where each set of equations is numbered naturally according to the indices of sequences.

The involved polynomials P_φ, R and S are all of size polynomial in the input size when represented as arithmetic circuits (R and S are even of constant size), and we have a polynomial number of equations. Thus, the definition above is an extended polyrec system of polynomial size. ◁

As discussed, Claim 24 finishes the proof of Theorem 18.

In the end, we discuss the Skolem problem: given a sequence $\mathbf{u}$ to determine whether there is n such that $u_n = 0$. This problem was extensively studied for the class of linear recursive sequences (see e.g. [27]). For linear recursive sequences it is open whether the Skolem problem is decidable, but only NP-hardness is known [5, Corollary 2.1]. For polyrec sequences, decidability of the Skolem problem is also open, but we can improve the lower bound.

▶ **Corollary 25.** *The Skolem problem is PSPACE-hard for polyrec sequences.*

Proof. Notice that in the proof of Theorem 18 we define a system of sequences over $\{0, 1\}$. It remains to observe that for such sequences the zeroness problem and the Skolem problem reduce to each other. Indeed, the nonzeroness problem of a sequence $\mathbf{u}$ over $\{0, 1\}$ is equivalent to the Skolem problem of $\mathbf{v}$ defined as $v_n = 1 - u_n$. ◀

We conclude this section by noting that the reduction from QBF that we have presented produces a polyrec sequence which is identically zero if and only if the first exponentially many initial values thereof are zero. We are not aware of examples requiring longer witnesses of zeroness for polyrec sequences.

Semigroup Intersection Problems in the Heisenberg Groups

Ruiwen Dong ✉
Department of Computer Science, University of Oxford, UK

Abstract

We consider two algorithmic problems concerning sub-semigroups of Heisenberg groups and, more generally, two-step nilpotent groups. The first problem is *Intersection Emptiness*, which asks whether a finite number of given finitely generated semigroups have empty intersection. This problem was first studied by Markov in the 1940s. We show that Intersection Emptiness is PTIME decidable in the Heisenberg groups $\mathrm{H}_n(\mathbb{K})$ over any algebraic number field $\mathbb{K}$, as well as in direct products of Heisenberg groups. We also extend our decidability result to arbitrary finitely generated 2-step nilpotent groups.

The second problem is *Orbit Intersection*, which asks whether the orbits of two matrices under multiplication by two semigroups intersect with each other. This problem was first studied by Babai et al. (1996), who showed its decidability within commutative matrix groups. We show that Orbit Intersection is decidable within the Heisenberg group $\mathrm{H}_3(\mathbb{Q})$.

2012 ACM Subject Classification Computing methodologies → Symbolic and algebraic manipulation

Keywords and phrases semigroup intersection, orbit intersection, matrix semigroups, Heisenberg group, nilpotent groups

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.25

Funding The author acknowledges support from UKRI Frontier Research Grant EP/X033813/1.

1 Introduction

The computational theory of matrix groups and semigroups is one of the oldest and most well-developed parts of computational algebra. Dating back to the work of Markov [31] in the 1940s, the area plays an essential role in analysing system dynamics, with notable applications in automata theory and program analysis [8, 10, 13, 21]. See [24] for an all-encompassing survey on this topic. While many computational problems are undecidable even for matrix groups of dimension three and four [6, 32, 34], various non-trivial algorithms have been developed for matrix groups satisfying additional constraints, such as commutativity [1], nilpotency [14], solvability [28], and having dimension two [5, 35].

As most algorithmic problems for commutative groups are well-understood due to their relatively simple structure, much effort has focused on problems concerning relaxations of the commutativity requirement, such as nilpotency and solvability. Prominent examples of widely studied groups include the *Heisenberg groups*, as well as the more general *2-step nilpotent groups*. The Heisenberg groups $\mathrm{H}_n(\mathbb{K})$ play an important role in many branches of mathematics, physics and computer science. They first arose in the description of one-dimensional quantum mechanical systems [33, 37], and have now become an important mathematical object connecting domains like representation theory, theta functions, Fourier analysis and quantum algorithms [20, 22, 25, 29, 38]. From a computational point of view, Heisenberg groups are interesting because they are the simplest non-commutative Lie groups. Heisenberg groups are included in the class of 2-step nilpotent groups: these are groups whose quotient by their centre is abelian. Despite being the simplest class of non-commutative groups, 2-step nilpotent groups admit highly non-trivial or even undecidable algorithmic problems, notably due to their ability to encode quadratic equations [27]. For example, decades of research has focused on finding a polynomial-time group isomorphism algorithm for 2-step nilpotent groups, with little success [2, 16].


© Ruiwen Dong;
licensed under Creative Commons License CC-BY 4.0


For a set $\mathcal{G}$ of matrices in some matrix group G, denote by $\langle\mathcal{G}\rangle$ the *semi*group generated by the set $\mathcal{G}$. In this paper, we consider the following two decision problems for the Heisenberg groups and 2-step nilpotent groups.

i. *(Intersection Emptiness)* Given M finite sets of matrices $\mathcal{G}_1, \ldots, \mathcal{G}_M$, decide whether $\langle\mathcal{G}_1\rangle \cap \cdots \cap \langle\mathcal{G}_M\rangle = \emptyset$.
ii. *(Orbit Intersection)* Given two finite sets of matrices $\mathcal{G}, \mathcal{H}$ and matrices S, T, decide whether $T \cdot \langle\mathcal{G}\rangle \cap S \cdot \langle\mathcal{H}\rangle = \emptyset$.

Intersection Emptiness was one of the first problems studied in algorithmic semigroup theory. In the seminal work of Markov [31], the undecidability of Intersection Emptiness was shown for two sets of 4×4 integer matrices. More recently, by encoding the *Post Correspondence Problem*, Halava and Harju showed its undecidability for two sets of 3×3 upper triangular integer matrices [17]. For 2×2 integer matrices, the problem is only known to be NP-hard [7]. In this paper, we show that Intersection Emptiness is decidable in polynomial time for the Heisenberg groups $\mathrm{H}_n(\mathbb{K})$ over an arbitrary algebraic number field $\mathbb{K}$, as well as for any direct product of such Heisenberg groups. In fact, we will prove the decidability result in the more general case of (finitely generated) 2-step nilpotent groups.

The Orbit Intersection problem was first considered by Babai *et al.* [1], who proved its decidability in *commutative* matrix groups over an algebraic number field. In this paper, we prove the decidability of Orbit Intersection for matrices in the Heisenberg group $\mathrm{H}_3(\mathbb{Q})$.

Let us mention some previous work for semigroup algorithmic problems in the Heisenberg groups and 2-step nilpotent groups. These have seen significant advance in research in recent years. Various results have been shown for the following decision problems.

iii. *(Identity Problem)* Given a finite set of matrices $\mathcal{G}$, decide whether the identity matrix $I \in \langle\mathcal{G}\rangle$.
iv. *(Membership Problem)* Given a finite set of matrices $\mathcal{G}$ and a matrix A, decide whether $A \in \langle\mathcal{G}\rangle$.
v. *(Knapsack Problem)* Given matrices $A_1, A_2, \ldots, A_K$ and a matrix A, decide whether there exist $(n_1, n_2 \ldots, n_K) \in \mathbb{N}^K$ such that $A = A_1^{n_1} A_2^{n_2} \cdots A_K^{n_K}$.

The Identity Problem in $\mathrm{H}_n(\mathbb{Q})$ was shown to be decidable by Ko, Niskanen and Potapov [26]. Dong [14] then introduced tools from Lie algebra and strengthened this result to PTIME decidability in $\mathrm{H}_n(\mathbb{K})$ for algebraic number fields $\mathbb{K}$. The Membership Problem in $\mathrm{H}_n(\mathbb{Q})$ was shown to be decidable by Colcombet, Ouaknine, Semukhin and Worrell. Their main idea is to use the *Baker-Campbell-Hausdorff (BCH) formula* as well as to incorporate the Membership Problem in a Parikh automaton. It was left as an open problem whether the Membership Problem in $\mathrm{H}_n(\mathbb{K})$ for larger fields $\mathbb{K}$ remains decidable. On the other hand, it is known that there exist 2-step nilpotent groups with undecidable Membership Problem [30]. As for the Knapsack Problem, König, Lohrey and Zetzsche showed its decidability in $\mathrm{H}_n(\mathbb{Z})$ by reducing it to solving a single quadratic equation over the natural numbers [27]. They also constructed a 2-step nilpotent group (namely, a direct product of $\mathrm{H}_3(\mathbb{Z})$) where the Knapsack Problem is undecidable, using an embedding of Hilbert's Tenth Problem.

We point out that by taking $\mathcal{G}_1 = \mathcal{G}$, $\mathcal{G}_2 = \{I\}$, Intersection Emptiness subsumes the Identity Problem. Whereas by taking $T = I, S = A, \mathcal{H} = \{I\}$, the Orbit Intersection problem subsumes the Membership Problem. Hence, the tools developed in this paper provide a more general approach to semigroup problems in 2-step nilpotent groups. Our proofs are based on the logarithm of matrices and the BCH formula, whose usage in studying matrix semigroup problems has been introduced in [12] and [14]. However, our approach goes much deeper in analysing the non-commutative terms of the BCH formula. We show that these terms are

connected with a word combinatorics problem concerning subwords of length two, and show a critical result characterizing the behaviour of these terms. This will allow us to reduce equations containing word combinatorial terms to pure linear Diophantine equations.

2 Main results

In this section we state our main results. Denote by $\mathsf{UT}(n,\mathbb{Q})$ the group of $n \times n$ upper triangular rational matrices with ones along the diagonal. Our main result on Intersection Emptiness is the following. For the formal definition of 2-step nilpotency, see Section 3.

▶ **Theorem 1.** *Let G be a 2-step nilpotent subgroup of $\mathsf{UT}(n,\mathbb{Q})$ for some n. Given finite subsets $\mathcal{G}_1,\dots,\mathcal{G}_M$ of G, it is decidable in polynomial time whether $\langle\mathcal{G}_1\rangle \cap \cdots \cap \langle\mathcal{G}_M\rangle = \emptyset$.*

For $n \geq 3$, the *Heisenberg group* $\mathrm{H}_n(\mathbb{K})$ over a field or commutative ring $\mathbb{K}$ is defined as

$$\mathrm{H}_n(\mathbb{K}) \coloneqq \left\{ \begin{pmatrix} 1 & \boldsymbol{a}^\top & c \\ 0 & I_{n-2} & \boldsymbol{b} \\ 0 & 0 & 1 \end{pmatrix}, \text{ where } \boldsymbol{a},\boldsymbol{b} \in \mathbb{K}^{n-2},\, c \in \mathbb{K} \right\},$$

where we use the notation I_d for the identity matrix of dimension d. Decidability results for the Heisenberg groups and for 2-step nilpotent groups follow as a corollary of Theorem 1.

▶ **Corollary 2.** *Intersection Emptiness is decidable:*

(i) *in PTIME, for the Heisenberg groups $\mathrm{H}_n(\mathbb{K})$ over any algebraic number field $\mathbb{K}$, and for any direct product of Heisenberg groups.*

(ii) *for finitely generated 2-step nilpotent groups*[1].

Fix a group G. Given an element $T \in G$ and a subset $\mathcal{G}$ of G, denote by $T \cdot \langle\mathcal{G}\rangle$ the orbit of T under right multiplication by the semigroup $\langle\mathcal{G}\rangle$. That is, $T \cdot \langle\mathcal{G}\rangle \coloneqq \{T \cdot s \mid s \in \langle\mathcal{G}\rangle\}$. Our main result concerning Orbit Intersection is the following.

▶ **Theorem 3.** *Given elements $T,S \in \mathrm{H}_3(\mathbb{Q})$ and two finite subsets $\mathcal{G},\mathcal{H}$ of $\mathrm{H}_3(\mathbb{Q})$, it is decidable whether $T \cdot \langle\mathcal{G}\rangle \cap S \cdot \langle\mathcal{H}\rangle = \emptyset$.*

3 Preliminaries

Convex geometry

Let V be a $\mathbb{Q}$-linear space. A subset $\mathcal{C} \subseteq V$ is called a *cone* if $a \in \mathcal{C}$ implies $a\mathbb{Q}_{\geq 0} \subseteq \mathcal{C}$, and $a,b \in \mathcal{C}$ implies $a+b \in \mathcal{C}$. Given a set of vectors $\mathcal{S} \subseteq V$, denote by $\langle\mathcal{S}\rangle_{\mathbb{Q}_{\geq 0}}$ the *cone generated by* $\mathcal{S}$, that is, the smallest cone of V containing $\mathcal{S}$. The *dimension* of a cone $\mathcal{C}$ is the dimension of the smallest linear space containing $\mathcal{C}$.

The *support* of a vector $\boldsymbol{\ell} = (\ell_1,\dots,\ell_K) \in \mathbb{Z}^K_{\geq 0}$ is defined as the set of indices where the entry of $\boldsymbol{\ell}$ is non-zero:

$$\mathrm{supp}(\boldsymbol{\ell}) \coloneqq \{i \in \{1,\dots,K\} \mid \ell_i > 0\}.$$

The *support* of a subset Λ of $\mathbb{Z}^K_{\geq 0}$ is defined as the union of supports of all vectors in Λ:

$$\mathrm{supp}(\Lambda) \coloneqq \bigcup_{\boldsymbol{\ell}\in\Lambda} \mathrm{supp}(\boldsymbol{\ell}) = \{i \mid \exists (\ell_1,\dots,\ell_K) \in \Lambda, \ell_i > 0\}.$$

In this paper, we will need to compute the support of sets of the form $\Lambda = \mathbb{Z}^K_{\geq 0} \cap V$, where V is a $\mathbb{Q}$-linear subspace of $\mathbb{Q}^K$.

[1] We suppose that the structure of the group is given by a finite presentation or a consistent polycyclic presentation (see [19, Chapter 8]).

▶ **Lemma 4** ([14, Lemma 2.4]). *Given V a $\mathbb{Q}$-linear subspace of $\mathbb{Q}^K$, represented as the solution set of linear homogeneous equations, one can compute the support of $\Lambda = \mathbb{Z}^K_{\geq 0} \cap V$ in polynomial time.*

The group $\mathsf{UT}(n, \mathbb{Q})$ and 2-step nilpotent groups

Denote by $\mathsf{UT}(n, \mathbb{Q})$ the group of $n \times n$ upper triangular rational matrices with ones along the diagonal. Let $\mathbb{K}$ be an algebraic number field. $\mathbb{K}$ can be considered as a linear space over $\mathbb{Q}$ of dimension $d \coloneqq [\mathbb{K} : \mathbb{Q}]$. Let $k_1, \ldots, k_d \in \mathbb{K}$ be a $\mathbb{Q}$-basis of this linear space. Throughout this paper, an element k of $\mathbb{K}$ is represented as a tuple $(a_1, \ldots, a_d) \in \mathbb{Q}^d$ such that $k = a_1k_1 + \cdots a_dk_d$. An element k of $\mathbb{K}$ acts on $\mathbb{K}$ by multiplication, and can therefore be considered as an endomorphism of the $\mathbb{Q}$-linear space $\mathbb{K}$. Associate k with the matrix that represents this endomorphism, then we have an (injective) embedding $\iota : \mathbb{K} \hookrightarrow \mathbb{Q}^{d\times d}$. In particular, $\iota(1) = I_d$. This embedding is effectively computable in polynomial time [11].

The embedding ι extends to an embedding $\mathrm{H}_n(\mathbb{K}) \hookrightarrow \mathsf{UT}(nd, \mathbb{Q})$, which we also denote by ι. Note that for any matrix $A \in \mathrm{H}_n(\mathbb{K})$, the total bit size of entries in $\iota(A)$ is at most quadratic in the total bit size of entries in A. Therefore, throughout this paper, we will work with matrices in $\iota(\mathrm{H}_n(\mathbb{K})) \subseteq \mathsf{UT}(nd, \mathbb{Q})$, knowing that any polynomial time algorithm in $\mathsf{UT}(nd, \mathbb{Q})$ will translate to a polynomial time algorithm in $\mathrm{H}_n(\mathbb{K})$.

Let G be an arbitrary group. The *centre* of G is the normal subgroup $Z(G) \trianglelefteq G$ consisting of elements that commute with every element of G (see [15]). We say that G is *2-step nilpotent* if the quotient $G/Z(G)$ is abelian. In particular, the Heisenberg groups $\mathrm{H}_n(\mathbb{K})$, as well as their direct products, are 2-step nilpotent [15, Examples 13.36]. Every finitely generated 2-step nilpotent group can be embedded as a subgroup of the direct product $A \times G_0$, where A is finite and G_0 is a 2-step nilpotent subgroup of $\mathsf{UT}(n, \mathbb{Q})$ for some n [4, Theorem 2.1] [23, Theorem 17.2.5].

Logarithm of matrices and Lie algebra

The *Lie algebra* $\mathfrak{u}(n)$ is defined as the $\mathbb{Q}$-linear space of $n \times n$ upper triangular rational matrices with *zeros* on the diagonal. There exist the *logarithm* map

$$\log : \mathsf{UT}(n, \mathbb{Q}) \to \mathfrak{u}(n), \quad A \mapsto \sum_{k=1}^{n} \frac{(-1)^{k-1}}{k}(A - I)^k$$

and the *exponential* map

$$\exp : \mathfrak{u}(n) \to \mathsf{UT}(n, \mathbb{Q}), \quad X \mapsto \sum_{k=0}^{n} \frac{1}{k!}X^k$$

which are inverse of one another. In particular, $\log I = 0$ and $\exp(0) = I$.

The Lie algebra $\mathfrak{u}(n)$ is equipped with the *Lie bracket* $[\cdot, \cdot] : \mathfrak{u}(n) \times \mathfrak{u}(n) \to \mathfrak{u}(n)$ given by $[X, Y] = XY - YX$. For a subset or subsemigroup $\mathcal{A}$ of $\mathsf{UT}(n, \mathbb{Q})$, we naturally denote by $\log \mathcal{A} \coloneqq \{\log a \mid a \in \mathcal{A}\}$ the set of logarithm of matrices in $\mathcal{A}$.

Parikh Image and length two subwords

Given a finite alphabet $\mathcal{G} = \{A_1, \ldots, A_K\}$, the *Parikh Image* of a word $w = M_1 \cdots M_m$ over the alphabet $\mathcal{G}$ is the vector $\mathrm{PI}^{\mathcal{G}}(w) \coloneqq (\mathrm{PI}^{\mathcal{G}}_1(w), \ldots, \mathrm{PI}^{\mathcal{G}}_K(w)) \in \mathbb{Z}^K_{\geq 0}$, where $\mathrm{PI}^{\mathcal{G}}_i(w)$ is the number of times A_i appears in w. That is, $\mathrm{PI}^{\mathcal{G}}_i(w) \coloneqq \mathrm{card}(\{j \mid M_j = A_i\})$. When the alphabet $\mathcal{G}$ is clear from the context, we sometimes write $\mathrm{PI}(w), \mathrm{PI}_i(w)$ instead of $\mathrm{PI}^{\mathcal{G}}(w), \mathrm{PI}^{\mathcal{G}}_i(w)$.

For $1 \le i < j \le K$, let w be a word over the alphabet $\mathcal{G}$, denote by $\delta_{ij}^{\mathcal{G}}(w)$ the number of occurrences of the subword $\cdots A_i \cdots A_j \cdots$ minus the number of occurrences of the subword $\cdots A_j \cdots A_i \cdots$ in w. That is, writing $w = M_1 M_2 \cdots M_s$, we have

$$\delta_{ij}^{\mathcal{G}}(w) \coloneqq \delta_{ij}^{\mathcal{G},+}(w) - \delta_{ij}^{\mathcal{G},-}(w),$$

where

$$\delta_{ij}^{\mathcal{G},+}(w) \coloneqq \{(u,v) \mid 1 \le u < v \le s, M_u = A_i, M_v = A_j\},$$
$$\delta_{ij}^{\mathcal{G},-}(w) \coloneqq \{(u,v) \mid 1 \le u < v \le s, M_u = A_j, M_v = A_i\}$$

Again, if the alphabet $\mathcal{G}$ is clear from the context, we write $\delta_{ij}(w)$ instead of $\delta_{ij}^{\mathcal{G}}(w)$. Obviously, we have the parity constraint

$$\delta_{ij}(w) \equiv \delta_{ij}^{+}(w) + \delta_{ij}^{-}(w) = \mathrm{PI}_i(w) \cdot \mathrm{PI}_j(w) \mod 2. \tag{1}$$

The Baker-Campbell-Hausdorff formula

Let G be a 2-step nilpotent subgroup of $\mathsf{UT}(n, \mathbb{Q})$. The *Baker-Campbell-Hausdorff (BCH) formula* [3, 9, 18] states that, given a sequence of matrices $B_1, B_2, \ldots, B_s$ in G, we have

$$\log(B_1 B_2 \cdots B_m) = \sum_{i=1}^{m} \log B_i + \frac{1}{2} \sum_{1 \le i < j \le s} [\log B_i, \log B_j]. \tag{2}$$

Fix a finite alphabet $\mathcal{G} = \{A_1, \ldots, A_K\}$ in G. For an arbitrary word w with Parikh Image $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_K)$, applying Equation (2) to the sequence of matrices in w yields

$$\log w = \sum_{i=1}^{K} \ell_i \log A_i + \frac{1}{2} \sum_{1 \le i < j \le K} \delta_{ij}(w) [\log A_i, \log A_j]. \tag{3}$$

Here, $\log w$ is understood to be the result of multiplying all matrices appearing in w in order, then taking the logarithm. We will adopt this notation throughout this paper.

4 A combinatorial problem for length two subwords

First let us describe the general strategy for solving intersection-type decision problems. Consider a simple example: given two alphabets $\mathcal{G} = \{A_1, \ldots, A_K\}$, $\mathcal{H} = \{B_1, \ldots, B_M\}$ in a 2-step nilpotent subgroup of $\mathsf{UT}(n, \mathbb{Q})$, we want to decide whether $\langle \mathcal{G} \rangle \cap \langle \mathcal{H} \rangle \neq \emptyset$. This boils down to finding two words v, w respectively in the alphabet $\mathcal{G}$ and $\mathcal{H}$, such that $\log v = \log w$. Denote by $\boldsymbol{x} = (x_1, \ldots, x_K)$ the Parikh Image of v, and by $\boldsymbol{y} = (y_1, \ldots, y_M)$ the Parikh Image of w, then the BCH formula (3) yields the equivalence between $\log v = \log w$ and

$$\sum_{i=1}^{K} x_i \log A_i + \sum_{i<j} \frac{\delta_{ij}^{\mathcal{G}}(v)}{2} [\log A_i, \log A_j] = \sum_{i=1}^{M} y_i \log B_i + \sum_{i<j} \frac{\delta_{ij}^{\mathcal{H}}(w)}{2} [\log B_i, \log B_j],$$
$$\boldsymbol{x} \in \mathbb{Z}_{\ge 0}^{K},\ \boldsymbol{y} \in \mathbb{Z}_{\ge 0}^{M}, \quad \mathrm{PI}^{\mathcal{G}}(v) = \boldsymbol{x},\ \mathrm{PI}^{\mathcal{H}}(w) = \boldsymbol{y}. \tag{4}$$

Hence, deciding whether $\langle \mathcal{G} \rangle \cap \langle \mathcal{H} \rangle \neq \emptyset$ boils down to solving Equation (4) in the numerical variables $\boldsymbol{x}, \boldsymbol{y}$ and the word variables v, w over alphabets $\mathcal{G}, \mathcal{H}$.

Consider a "relaxed" version of this problem. That is, we replace $\delta_{ij}^{\mathcal{G}}(v)$ and $\delta_{ij}^{\mathcal{H}}(w)$ by new variables c_{ij}, d_{ij} over integers, without imposing any constraint. This gives the equation

$$\sum_{i=1}^{K} x_i \log A_i + \sum_{1\le i<j\le K} \frac{c_{ij}}{2}[\log A_i, \log A_j] = \sum_{i=1}^{M} y_i \log B_i + \sum_{1\le i<j\le M} \frac{d_{ij}}{2}[\log B_i, \log B_j],$$
$$\boldsymbol{x} \in \mathbb{Z}_{\ge 0}^{K},\ \boldsymbol{y} \in \mathbb{Z}_{\ge 0}^{M}, \quad c_{ij}, d_{ij} \in \mathbb{Z} \text{ for all } i, j. \tag{5}$$

Obviously, if Equation (4) has a solution, then the relaxed version (5) will also admit a solution. The converse is not necessarily true. The implicit constraints imposed by the word combinatorial variables $\delta_{ij}^{\mathcal{G}}(v), \delta_{ij}^{\mathcal{H}}(w)$ in Equation (4) are highly non-trivial. (For example, one should at least have $|\delta_{ij}^{\mathcal{G}}(v)| \le x_i x_j$ for all i, j). However, these constraints are not reflected by the numerical variables c_{ij} and d_{ij} in Equation (5).

The key idea of this paper is the following surprising fact. For the two problems we consider (Semigroup Intersection and Orbit Intersection), it is sufficient to solve the relaxed version of the equation, plus several simple constraints (such as the modulo 2 constraint in Equation (1)). In particular, given a "suitable" solution to the relaxed Equation (5), we can always construct a solution to Equation (4). *A priori*, the values of $\delta_{ij}^{\mathcal{G}}(v)$ cannot reach all integers like the free variables c_{ij}; nevertheless, when $x_1, \ldots, x_K$ tend towards infinity, the vector $\left(\delta_{ij}^{\mathcal{G}}(v)\right)_{1\le i<j\le K}$ can in fact reach every value within a ball of radius size $O(|\boldsymbol{x}|^2)$, satisfying modulo 2 constraints. This will suffice to construct a suitable word v, as the quadratic radius will eventually dominate the linear term $\sum_{i=1}^{K} x_i \log A_i$.

This section aims to formalize this idea. The main result of this section will be Proposition 6. First, we prove a simple case where the alphabet consists of two letters.

▶ **Lemma 5.** *Given an alphabet $\mathcal{G} = \{A_i, A_j\}$ and non-negative integers $s_i, s_j \in \mathbb{Z}_{\ge 0}$, then for every $C \in \mathbb{Z}$ satisfying*

$$|C| \le s_i s_j \quad \text{and} \quad C \equiv s_i s_j \mod 2, \tag{6}$$

there exists a permutation w of the word $A_i^{s_i} A_j^{s_j}$ such that $\delta_{ij}(w) = C$.

Proof. For an illustration of the proof, see Figure 1. We start with the word $w = A_i^{s_i} A_j^{s_j}$, which satisfies $\delta_{ij}(w) = s_i s_j$. We gradually swap pairs of consecutive letters in w: each time we replace an occurrence of consecutive $A_i A_j$ with $A_j A_i$. An occurrence of $A_i A_j$ can always be found unless we have reached the "final" permutation $A_j^{s_j} A_i^{s_i}$. It is easy to see that each swap reduces the value of $\delta_{ij}(w)$ by 2. Therefore, by swapping consecutive $A_i A_j$ one by one, $\delta_{ij}(w)$ can reach every value between $\delta_{ij}(A_i^{s_i} A_j^{s_j}) = s_i s_j$ and $\delta_{ij}(A_j^{s_j} A_i^{s_i}) = -s_i s_j$ that has the same parity with $s_i s_j$. This proves the lemma. ◀

We then prove the main result of this section, which generalizes Lemma 5 to alphabets of more than two letters.

▶ **Proposition 6.** *Fix a finite alphabet $\mathcal{G}$ of size $K \ge 2$. Then for any tuples $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_K) \in \mathbb{Z}_{\ge 0}^{K}$ and $\{C_{ij}\}_{1\le i<j\le K} \in \mathbb{Z}_{\ge 0}^{K(K-1)/2}$ satisfying*

$$|C_{ij}| \le \frac{\ell_i \ell_j}{4K^2} - 2K(\ell_i + \ell_j) - 4K^2, \quad \text{for all } 1 \le i < j \le K, \tag{7}$$

and

$$C_{ij} \equiv \ell_i \ell_j \mod 2, \quad \text{for all } 1 \le i < j \le K, \tag{8}$$

there exists a word w with Parikh Image $\boldsymbol{\ell}$ such that

$$\delta_{ij}(w) = C_{ij}, \quad \text{for all } 1 \le i < j \le K. \tag{9}$$

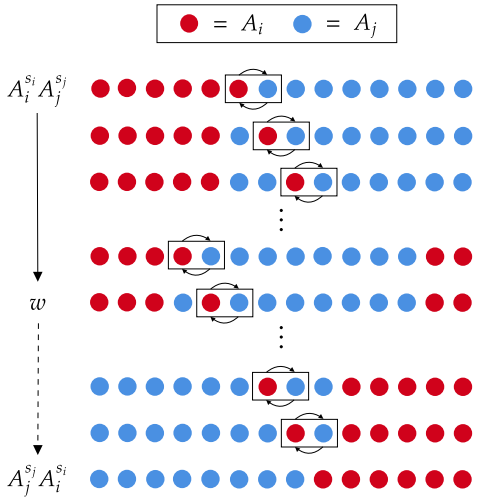


Figure 1 Illustration for the proof of Lemma 5.

Proof. For an illustration of the proof, see Figure 2. For all i, write $\ell_i = 2(K-1)s_i + r_i$ with $0 \le r_i < 2(K-1)$. Consider the word $W_{init} \coloneqq W_{res} \cdot W \cdot W_{rev}$, where

$$\begin{aligned} W_{res} &\coloneqq A_1^{r_1} A_2^{r_2} \cdots A_K^{r_K}, \\ W &\coloneqq (A_1^{s_1} A_2^{s_2})\,(A_1^{s_1} A_3^{s_3}) \cdots (A_1^{s_1} A_K^{s_K})\,(A_2^{s_2} A_3^{s_3}) \cdots (A_2^{s_2} A_K^{s_K})\,(A_3^{s_3} A_4^{s_4}) \cdots \left(A_{K-1}^{s_{K-1}} A_K^{s_K}\right), \\ W_{rev} &\coloneqq \left(A_K^{s_K} A_{K-1}^{s_{K-1}}\right)\left(A_K^{s_K} A_{K-2}^{s_{K-2}}\right) \cdots (A_K^{s_K} A_1^{s_1})\left(A_{K-1}^{s_{K-1}} A_{K-2}^{s_{K-2}}\right)\left(A_{K-1}^{s_{K-1}} A_{K-3}^{s_{K-3}}\right) \cdots (A_2^{s_2} A_1^{s_1}). \end{aligned}$$

In particular, W is the concatenation of all words of the form $A_i^{s_i} A_j^{s_j}$ where $i < j$, and W_{rev} is the reverse of W. It is easy to verify that W_{init} contains ℓ_i occurrences of the letter A_i, so its Parikh Image is exactly $\boldsymbol{\ell}$.

We now compute $\delta_{ij}(W_{init})$ for $i < j$. Since $W \cdot W_{rev}$ is a palindrome, we have $\delta_{ij}(W \cdot W_{rev}) = 0$, so

$$\begin{aligned} \delta_{ij}(W_{init}) &= \delta_{ij}(W_{res}) + \mathrm{PI}_i(W_{res})\,\mathrm{PI}_j(W \cdot W_{rev}) - \mathrm{PI}_j(W_{res})\,\mathrm{PI}_i(W \cdot W_{rev}) \\ &= r_i r_j + r_i \cdot 2(K-1)s_j - r_j \cdot 2(K-1)s_i. \end{aligned} \tag{10}$$

In particular, since $0 \le r_i < 2(K-1)$, we have

$$|\delta_{ij}(W_{init})| \le 4(K-1)^2 + 2(K-1)^2(s_j + s_i) < 4K^2 + 2(K-1)(\ell_i + \ell_j) \tag{11}$$

By Condition 7, we have

$$\begin{aligned} |\delta_{ij}(W_{init}) - C_{ij}| &\le |\delta_{ij}(W_{init})| + |C_{ij}| \\ &< 4K^2 + 2(K-1)(\ell_i + \ell_j) + \frac{\ell_i \ell_j}{4K^2} - 2K(\ell_i + \ell_j) - 4K^2 \\ &< \left(\frac{\ell_i}{2(K-1)} - 1\right)\left(\frac{\ell_j}{2(K-1)} - 1\right) \\ &< s_i s_j. \end{aligned} \tag{12}$$

Since Equation (10) yields $\delta_{ij}(W_{init}) \equiv \ell_i \ell_j \mod 2$, Condition (8) then gives

$$\delta_{ij}(W_{init}) \equiv C_{ij} \mod 2. \tag{13}$$

We now show how to construct the word w. Starting with the word W_{init}, for every pair $i < j$ perform the following:

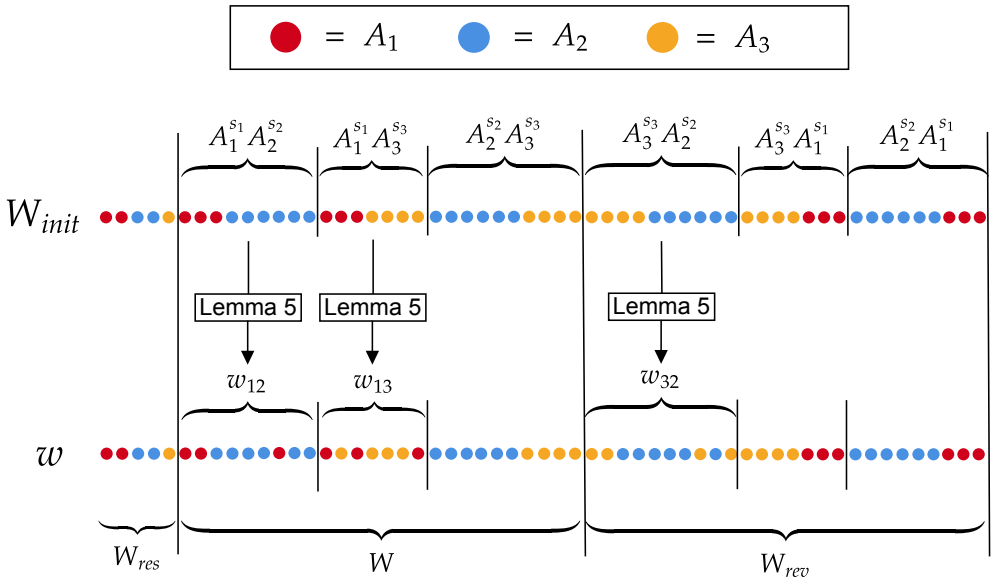


Figure 2 Illustration for the proof of Proposition 6.

1. If $\delta_{ij}(W_{init}) > C_{ij}$. By Lemma 5 there exists a permutation w_{ij} of the word $A_i^{s_i}A_j^{s_j}$ such that $\delta_{ij}(w_{ij}) = s_is_j + C_{ij} - \delta_{ij}(W_{init})$. Indeed, Equations (12) and (13) guarantee that the conditions (6) in Lemma 5 are satisfied. We then replace the subword $A_i^{s_i}A_j^{s_j}$ in the W-part of W_{init} with the word w_{ij}. The resulting new word W' will satisfy

$$\delta_{ij}(W') = \delta_{ij}(W_{init}) - \delta_{ij}(A_i^{s_i}A_j^{s_j}) + \delta_{ij}(w_{ij}) = C_{ij}.$$

   This replacement does not change $\delta_{uv}(W_{init})$ for $(u,v) \neq (i,j)$.
2. If $\delta_{ij}(W_{init}) < C_{ij}$. By Lemma 5 there exists a permutation w_{ji} of the word $A_j^{s_j}A_i^{s_i}$ such that $\delta_{ij}(w_{ji}) = -s_is_j + C_{ij} - \delta_{ij}(W_{init})$. Again, Equations (12) and (13) guarantee that the conditions (6) in Lemma 5 are satisfied. We then replace the subword $A_j^{s_j}A_i^{s_i}$ in the W_{rev}-part of W_{init} with the word w_{ji}. The resulting new word W' will satisfy

$$\delta_{ij}(W') = \delta_{ij}(W_{init}) - \delta_{ij}(A_j^{s_j}A_i^{s_i}) + \delta_{ij}(w_{ji}) = C_{ij}.$$

   This replacement does not change $\delta_{uv}(W_{init})$ for $(u,v) \neq (i,j)$.
3. If $\delta_{ij}(W_{init}) = C_{ij}$, do not perform any change.

Performing all these replacements on W_{init} for all pairs $i < j$ simultaneously, the resulting word w then satisfies $\delta_{ij}(w) = C_{ij}$ for all $1 \leq i < j \leq K$. ◀

5 A polynomial time algorithm for Intersection Emptiness

We prove Theorem 1 in this section. Let G be a 2-step nilpotent subgroup of $\mathsf{UT}(n,\mathbb{Q})$. Let

$$\mathcal{G}_1 = \{A_{11}, A_{12}, \ldots, A_{1K_1}\}, \ldots, \mathcal{G}_M = \{A_{M1}, A_{M2}, \ldots, A_{MK_M}\}$$

be M sets of matrices in G. The following proposition shows that Intersection Emptiness can be reduced to solving linear Diophantine equations with extra constraints on supports. The key to obtaining a PTIME algorithm is the fact that these equations are all *homogeneous*. Hence one can actually solve them over $\mathbb{Q}$, then scale them to obtain integer solutions.

▶ **Proposition 7.** *We have $\langle \mathcal{G}_1 \rangle \cap \cdots \cap \langle \mathcal{G}_M \rangle \neq \emptyset$ if and only if there exist non-zero vectors $\boldsymbol{\ell}_1 \in \mathbb{Z}_{\geq 0}^{K_1} \setminus \{\mathbf{0}\}, \cdots, \boldsymbol{\ell}_M \in \mathbb{Z}_{\geq 0}^{K_m} \setminus \{\mathbf{0}\}$ as well as rational numbers c_{mij} for $1 \leq m \leq M, i, j \in \operatorname{supp}(\boldsymbol{\ell}_m)$, such that*

$$\sum_{j=1}^{K_1} \ell_{1j} \log A_{1j} + \sum_{\substack{i<j \\ i,j \in \operatorname{supp}(\boldsymbol{\ell}_1)}} c_{1ij} [\log A_{1i}, \log A_{1j}] =$$

$$\sum_{j=1}^{K_2} \ell_{2j} \log A_{2j} + \sum_{\substack{i<j \\ i,j \in \operatorname{supp}(\boldsymbol{\ell}_2)}} c_{2ij} [\log A_{2i}, \log A_{2j}] = \cdots$$

$$= \sum_{j=1}^{K_M} \ell_{Mj} \log A_{Mj} + \sum_{\substack{i<j \\ i,j \in \operatorname{supp}(\boldsymbol{\ell}_m)}} c_{Mij} [\log A_{Mi}, \log A_{Mj}] \quad (14)$$

Proof. If $\langle \mathcal{G}_1 \rangle \cap \cdots \cap \langle \mathcal{G}_M \rangle \neq \emptyset$, let g be an element in the intersection. There exist non-empty words $w_1, \ldots, w_m$ over the alphabets $\mathcal{G}_1, \ldots, \mathcal{G}_M$ such that $\log g = \log w_1 = \cdots = \log w_m$. By the BCH formula (3),

$$\log g = \sum_{j=1}^{K_i} \mathrm{PI}_i(w_m) \log A_{mj} + \sum_{\substack{i<j \\ i,j \in \operatorname{supp}(\boldsymbol{\ell}_m)}} \frac{\delta_{ij}(w_m)}{2} [\log A_{mi}, \log A_{mj}], \quad \text{for } m = 1, \ldots, M.$$

This shows that (14) is satisfied by $\boldsymbol{\ell}_m \coloneqq \mathrm{PI}^{\mathcal{G}_m}(w_m)$ and $c_{mij} \coloneqq \delta_{ij}(w_m)/2$ for $1 \leq m \leq M, i, j \in \operatorname{supp}(\boldsymbol{\ell}_m)$.

For the other implication, suppose such non-zero vectors $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_M$ and the rational numbers c_{mij} exist. Then there exists $g \in G$ such that

$$\sum_{j=1}^{K_1} \ell_{mj} \log A_{mj} + \sum_{\substack{i<j \\ i,j \in \operatorname{supp}(\boldsymbol{\ell}_m)}} \frac{2c_{mij}}{2} [\log A_{mi}, \log A_{mj}] = \log g, \quad \text{for } m = 1, \ldots, M. \quad (15)$$

Note that if $i, j \in \operatorname{supp}(\boldsymbol{\ell}_m)$ then $\ell_{mi}\ell_{mj} \neq 0$.

By homogeneity, for any $N \in \mathbb{Z}_{>0}$, the vectors $N\boldsymbol{\ell}_1, \ldots, N\boldsymbol{\ell}_M$ and Nc_{mij} also satisfy Condition (14). Hence, multiplying all ℓ_{ij}, c_{mij} and $\log g$ by a common denominator, we can suppose all ℓ_{ij} and c_{mij} to be integers. Denote $K = \max_{1 \leq m \leq M} K_m$, then there exists a large enough *even* integer $N \in \mathbb{Z}_{>0}$ such that

$$|N \cdot 2c_{mij}| \leq \frac{N^2 \ell_{mi}\ell_{mj}}{4K^2} - 2NK(\ell_i + \ell_j) - 4K^2 \quad (16)$$

for $1 \leq m \leq M, i, j \in \operatorname{supp}(\boldsymbol{\ell}_m)$. This is because $\ell_{mi}\ell_{mj} > 0$, so the right hand side of (16) is quadratic and dominates the linear term on the left for large enough N. Replace all ℓ_{ij} with $N\ell_{ij}$, all c_{mij} with Nc_{mij}, and $\log g$ with $N \log g$, then the new variables satisfy $2c_{mij} \equiv 0 \equiv \ell_{mi}\ell_{mj} \mod 2$, and

$$|2c_{mij}| \leq \frac{\ell_{mi}\ell_{mj}}{4K^2} - 2K(\ell_i + \ell_j) - 4K^2 \leq \frac{\ell_{mi}\ell_{mj}}{4K_m^2} - 2K_m(\ell_i + \ell_j) - 4K_m^2 \quad (17)$$

for all i, j, m. Equation (15) is still satisfied after the variable replacements. Therefore, by Proposition 6, there exist words $w_1, \ldots, w_M$ over the alphabets $\mathcal{G}_1, \ldots, \mathcal{G}_M$ such that $\mathrm{PI}(w_m) = \boldsymbol{\ell}_m$ and $\delta_{ij}(w_m) = 2c_{mij}$ for all $1 \leq m \leq M, i, j \in \operatorname{supp}(\boldsymbol{\ell}_m)$. These words are non-empty since $\boldsymbol{\ell}_m \neq \mathbf{0}$. Plugging into the BCH formula (3), we have

$$\log w_m = \sum_{j=1}^{K_m} \ell_{mj} \log A_{mj} + \sum_{\substack{i<j \\ i,j\in \mathrm{supp}(\boldsymbol{\ell}_m)}} \frac{2c_{mij}}{2}[\log A_{mi}, \log A_{mj}] = \log g \quad \text{for } m = 1, \dots, M.$$

This shows that $\log g \in \bigcap_{i=1}^{M} \log\langle \mathcal{G}_i \rangle \neq \emptyset$. ◀

Using Proposition 7, we devise Algorithm 1 that decides Intersection Emptiness.

Algorithm 1 Algorithm for Intersection Emptiness.

Input: M finite sets of matrices $\mathcal{G}_1 = \{A_{11}, A_{12}, \dots, A_{1K_1}\}, \dots, \mathcal{G}_M = \{A_{M1}, A_{M2}, \dots, A_{MK_M}\}$ in the group G.

Output: **True** (intersection is empty) or **False** (intersection is not empty).

Step 1: **Initialization.** Set $S_1 \coloneqq \{1, 2, \dots, K_1\}, \dots, S_M \coloneqq \{1, 2, \dots, K_M\}$.

Step 2: **Main loop.** Repeat the following

a. Represent the $\mathbb{Q}$-linear subspace of $V \coloneqq \mathbb{Q}^{\sum_{m=1}^{M} K_m + \sum_{m=1}^{M} \mathrm{card}(S_m)(\mathrm{card}(S_m)-1)/2}$:

$$W \coloneqq \Bigg\{ \big((\ell_{mj})_{1\le m\le M, 1\le j\le K_m}, (c_{mij})_{1\le m\le M, i,j\in S_m}\big) \in V \;\Bigg|\; \sum_{j=1}^{K_1} \ell_{1j} \log A_{1j} + \sum_{\substack{i<j \\ i,j\in S_1}} c_{1ij}[\log A_{1i}, \log A_{1j}] = \cdots = \sum_{j=1}^{K_M} \ell_{Mj} \log A_{Mj} + \sum_{\substack{i<j \\ i,j\in S_M}} c_{Mij}[\log A_{Mi}, \log A_{Mj}] \Bigg\} \quad (18)$$

as the solution set of homogeneous linear equations.

b. Compute the projection of W onto the coordinates $(\ell_{mj})_{1\le m\le M, 1\le j\le K_m}$:

$$\pi_{\boldsymbol{\ell}}(W) \coloneqq \Bigg\{ (\ell_{mj})_{1\le m\le M, 1\le j\le K_m} \in \mathbb{Q}^{\sum_{m=1}^{M} K_m} \;\Bigg|\; \exists (c_{mij})_{1\le m\le M, i,j\in S_m}, \big((\ell_{mj})_{1\le m\le M, 1\le j\le K_m}, (c_{mij})_{1\le m\le M, i,j\in S_m}\big) \in W \Bigg\} \quad (19)$$

represented as the solution set of homogeneous linear equations.

c. Define $\Lambda \coloneqq \mathbb{Z}_{\ge 0}^{\sum_{m=1}^{M} K_m} \cap \pi_{\boldsymbol{\ell}}(W)$ and compute $\mathrm{supp}(\Lambda)$ using Lemma 4.

d. If $\mathrm{supp}(\Lambda) \cap S_m = S_m$ for all $1 \le m \le M$, terminate the loop and go to Step 3. Otherwise, let $S_m \coloneqq \mathrm{supp}(\Lambda) \cap S_m$ for every m, and continue with Step 2.

Step 3: **Output.**

a. If $S_m = \emptyset$ for any $1 \le m \le M$, return **True**.

b. Otherwise return **False**.

▶ **Theorem 1.** *Let G be a 2-step nilpotent subgroup of $\mathsf{UT}(n, \mathbb{Q})$ for some n. Given finite subsets $\mathcal{G}_1, \dots, \mathcal{G}_M$ of G, it is decidable in polynomial time whether $\langle \mathcal{G}_1 \rangle \cap \cdots \cap \langle \mathcal{G}_M \rangle = \emptyset$.*

Proof. Theorem 1 follows from the correctness and polynomial time complexity of Algorithm 1. Their proofs are given in Appendix A, Proposition 10. ◀

6 Decidability of Orbit Intersection in $\mathrm{H}_3(\mathbb{Q})$

We prove Theorem 3 in this section. Let $\mathcal{G}$ and $\mathcal{H}$ be finite sets of matrices in the group $\mathrm{H}_3(\mathbb{Q})$, and T, S be matrices in $\mathrm{H}_3(\mathbb{Q})$. Our goal is to decide whether $T \cdot \langle \mathcal{G} \rangle \cap S \cdot \langle \mathcal{H} \rangle = \emptyset$. Multiplying both $T \cdot \langle \mathcal{G} \rangle$ and $S \cdot \langle \mathcal{H} \rangle$ on the left by T^{-1}, one can without loss of generality suppose $T = I$. That is, it suffices to consider the problem of deciding whether $\langle \mathcal{G} \rangle \cap S \cdot \langle \mathcal{H} \rangle = \emptyset$. Denote by $\varphi\colon \log \mathrm{H}_3(\mathbb{Q}) \to \mathbb{Q}^2$ the projection onto the superdiagonal, and by $\pi\colon \log \mathrm{H}_3(\mathbb{Q}) \to \mathbb{Q}$ the projection onto the upper right entry:

$$\varphi\colon \begin{pmatrix} 0 & a & c \\ 0 & 0 & b \\ 0 & 0 & 0 \end{pmatrix} \mapsto (a, b); \qquad \pi\colon \begin{pmatrix} 0 & a & c \\ 0 & 0 & b \\ 0 & 0 & 0 \end{pmatrix} \mapsto c.$$

One easily verifies that for matrices $X, Y \in \mathrm{H}_3(\mathbb{Q})$, we have $[\log X, \log Y] = 0$ if and only if $\varphi(\log X)$ and $\varphi(\log Y)$ are linearly dependent. Define the cones

$$\mathcal{C}_{\mathcal{G}} \coloneqq \langle \varphi(\log \mathcal{G}) \rangle_{\mathbb{Q}_{\geq 0}}, \quad \mathcal{C}_{\mathcal{H}} \coloneqq \langle \varphi(\log \mathcal{H}) \rangle_{\mathbb{Q}_{\geq 0}}.$$

6.1 Easy case: The cone $\mathcal{C}_{\mathcal{G}} \cap \mathcal{C}_{\mathcal{H}}$ has dimension zero or one

The situation in this case is similar to the one discussed in [12, Section 3, Case I].

▶ **Proposition 8.** *Suppose the cone $\mathcal{C}_{\mathcal{G}} \cap \mathcal{C}_{\mathcal{H}}$ has dimension zero or one. Deciding whether $\langle \mathcal{G} \rangle \cap S \cdot \langle \mathcal{H} \rangle \neq \emptyset$ can be done by solving finitely many linear Diophantine equations.*

6.2 Hard case: The cone $\mathcal{C}_{\mathcal{G}} \cap \mathcal{C}_{\mathcal{H}}$ has dimension two

We have $\langle \mathcal{G} \rangle \cap S \cdot \langle \mathcal{H} \rangle \neq \emptyset$ if and only if there exist words v in the alphabet $\mathcal{G}$ and w in the alphabet $\mathcal{H}$ such that $\log v = \log Sw$. Let $\boldsymbol{x} = (x_1, \ldots, x_K)$ be the Parikh Image of v, and $\boldsymbol{y} = (y_1, \ldots, y_M)$ be the Parikh Image of w. By the BCH formula (2) and (3), $\log v = \log Sw$ is equivalent to

$$\sum_{i=1}^{K} x_i \log A_i + \frac{1}{2} \sum_{1 \leq i < j \leq K} \delta_{ij}^{\mathcal{G}}(v)[\log A_i, \log A_j] = \log S + \sum_{i=1}^{M} y_i (\log B_i + \frac{1}{2}[\log S, \log B_i]) + \frac{1}{2} \sum_{1 \leq i < j \leq M} \delta_{ij}^{\mathcal{H}}(w)[\log B_i, \log B_j] \quad (20)$$

The following proposition shows that it suffices to solve a relaxed version of Equation (20).

▶ **Proposition 9.** *Suppose the cone $\mathcal{C}_{\mathcal{G}} \cap \mathcal{C}_{\mathcal{H}}$ has dimension two. We have $\langle \mathcal{G} \rangle \cap S \cdot \langle \mathcal{H} \rangle \neq \emptyset$ if and only if there exists integers $x_i, 1 \leq i \leq K$ and $y_j, 1 \leq j \leq M$ and $c_{ij}, 1 \leq i < j \leq K$ and $d_{ij}, 1 \leq i < j \leq M$, satisfying*

$$\sum_{i=1}^{K} x_i \varphi(\log A_i) = \varphi(\log S) + \sum_{i=1}^{M} y_i \varphi(\log B_i), \quad (21)$$

$$\sum_{i=1}^{K} x_i \pi(\log A_i) + \frac{1}{2} \sum_{1 \leq i < j \leq K} c_{ij} \pi([\log A_i, \log A_j]) = \pi(\log S) + \sum_{i=1}^{M} y_i \pi(\log B_i + \frac{1}{2}[\log S, \log B_i]) + \frac{1}{2} \sum_{1 \leq i < j \leq M} d_{ij} \pi([\log B_i, \log B_j]) \quad (22)$$

and

$$c_{ij} \equiv x_i x_j \mod 2, \quad 1 \leq i < j \leq K; \qquad d_{ij} \equiv y_i y_j \mod 2, \quad 1 \leq i < j \leq M. \tag{23}$$

Proof. If $\langle \mathcal{G} \rangle \cap s \cdot \langle \mathcal{H} \rangle \neq \emptyset$, then let v, w be non-empty words over the respectively alphabets $\mathcal{G}$ and $\mathcal{H}$, such that $\log v = \log Sw$. Let $c_{ij} \coloneqq \delta_{ij}^{\mathcal{G}}(v)$ and $d_{ij} \coloneqq \delta_{ij}^{\mathcal{H}}(w)$ for all i, j. Since Equation (20) is satisfied, projecting it under φ and π gives respectively (21) and (22). The parity condition (23) is obviously due to Equation (1). Hence we have found the integers x_i, y_j, c_{ij}, d_{ij} satisfying Equations (21), (22) and (23).

On the other hand, let x_i, y_j, c_{ij}, d_{ij} be integers that satisfy Equations (21), (22) and (23). Since $\mathcal{C}_{\mathcal{G}}$ and $\mathcal{C}_{\mathcal{H}}$ have dimension two, the commutators $[\log A_i, \log A_j]$ and $[\log B_i, \log B_j]$ are not all zero (since $\varphi(A_i)$ are not all linearly dependant, same for $\varphi(B_i)$). Hence, there exist integers C_{ij}, D_{ij} such that

$$D \coloneqq \sum_{1 \leq i < j \leq K} C_{ij} \pi([\log A_i, \log A_j]) + \sum_{1 \leq i < j \leq M} D_{ij} \pi([\log B_i, \log B_j]) \in \mathbb{Q}_{>0}.$$

Denote by E the common denominator of all the entries of the matrices $\log A_i$, $\log B_i$, $\log S$, $\frac{1}{2}[\log S, \log B_i]$, $\frac{1}{2}[\log A_i, \log A_j]$ and $\frac{1}{2}[\log B_i, \log B_j]$. In particular, DE is an integer.

Since the cone $\mathcal{C}_{\mathcal{G}} \cap \mathcal{C}_{\mathcal{H}}$ has dimension two, there exist *strictly positive* integers $X_1, \ldots, X_K$ and $Y_1, \ldots, Y_M$, such that

$$\sum_{i=1}^{K} X_i \varphi(\log A_i) = \sum_{i=1}^{M} Y_i \varphi(\log B_i). \tag{24}$$

This is because, taking $\boldsymbol{v}$ to be a vector in the *interior* of $\mathcal{C}_{\mathcal{G}} \cap \mathcal{C}_{\mathcal{H}}$ (i.e. $\boldsymbol{v}$ admits an open neighbourhood contained in $\mathcal{C}_{\mathcal{G}} \cap \mathcal{C}_{\mathcal{H}}$), then $\boldsymbol{v}$ is in the interior of both $\mathcal{C}_{\mathcal{G}}$ and $\mathcal{C}_{\mathcal{H}}$. Hence, there exist strictly positive rational numbers $X'_1, \ldots, X'_K$ and $Y'_1, \ldots, Y'_M$, such that

$$\sum_{i=1}^{K} X'_i \varphi(\log A_i) = \boldsymbol{v} = \sum_{i=1}^{M} Y'_i \varphi(\log B_i).$$

Multiplying $X'_1, \ldots, X'_K$ and $Y'_1, \ldots, Y'_M$ by their common denominator gives positive integers satisfying Equation (24).

For any $N \in \mathbb{Z}_{>0}$, the integers x_i, y_i, c_{ij}, d_{ij} can be replaced by the integers

$$\begin{aligned}
x'_i &\coloneqq x_i + 2NDEX_i \\
y'_i &\coloneqq y_i + 2NDEY_i \\
c'_{ij} &\coloneqq c_{ij} - 4NEC_{ij} \left(\sum_{k=1}^{K} X_k \pi(\log A_k) - \sum_{k=1}^{M} Y_k \pi(\log B_k + \frac{1}{2}[\log S, \log B_k]) \right) \\
d'_{ij} &\coloneqq d_{ij} + 4NED_{ij} \left(\sum_{k=1}^{K} X_k \pi(\log A_k) - \sum_{k=1}^{M} Y_k \pi(\log B_k + \frac{1}{2}[\log S, \log B_k]) \right)
\end{aligned}$$

for all i, j, while still satisfying Equations (21), (22) and (23). Furthermore, when N is large enough, we have

$$x'_i > 0, y'_j > 0, \quad 1 \leq i \leq K, 1 \leq j \leq M, \tag{25}$$

$$|c'_{ij}| \leq \frac{x'_i x'_j}{4K^2} - 2K(x'_i + x'_j) - 4K^2, \quad 1 \leq i < j \leq K, \tag{26}$$

and

$$|d'_{ij}| \leq \frac{y'_i y'_j}{4M^2} - 2M(y'_i + y'_j) - 4M^2, \quad 1 \leq i < j \leq M. \tag{27}$$

This is because the right hand sides of the inequalities (26) and (27) are quadratic in N, whereas the left hand sides grow linearly in N.

Fix an N such that the inequalities (25), (26) and (27) are satisfied. Then, by Proposition 6, there exist non-empty words v, w over the alphabets $\mathcal{G}$ and $\mathcal{H}$, such that

$$\begin{aligned} \mathrm{PI}^{\mathcal{G}}(v) &= (x'_1, \ldots, x'_K), \quad \delta^{\mathcal{G}}_{ij}(v) = c'_{ij}, \quad \text{for } 1 \leq i < j \leq K, \\ \mathrm{PI}^{\mathcal{H}}(w) &= (y'_1, \ldots, y'_K), \quad \delta^{\mathcal{H}}_{ij}(v) = d'_{ij}, \quad \text{for } 1 \leq i < j \leq M. \end{aligned}$$

(Note that Condition (8) is guaranteed by Equation (23).) For these words v, w, we have

$$\varphi(\log v) = \sum_{i=1}^{K} x'_i \varphi(\log A_i) = \varphi(\log S) + \sum_{i=1}^{M} y'_i \varphi(\log B_i) = \varphi(\log Sw),$$

as well as

$$\pi(\log v) = \sum_{i=1}^{K} x'_i \pi(\log A_i) + \frac{1}{2} \sum_{1 \leq i < j \leq K} c'_{ij} \pi([\log A_i, \log A_j]) =$$

$$\pi(\log S) + \sum_{i=1}^{M} y'_i \pi(\log B_i + \frac{1}{2}[\log S, \log B_i]) + \frac{1}{2} \sum_{1 \leq i < j \leq M} d'_{ij} \pi([\log B_i, \log B_j]) = \pi(\log Sw).$$

This shows $\log v = \log Sw$, hence $\langle \mathcal{G} \rangle \cap s \cdot \langle \mathcal{H} \rangle \neq \emptyset$. ◀

Combining the two cases in Subsections 6.1 and 6.2, we are able to solve the Orbit Intersection problem for $\mathrm{H}_3(\mathbb{Q})$.

▶ **Theorem 3.** *Given elements $T, S \in \mathrm{H}_3(\mathbb{Q})$ and two finite subsets $\mathcal{G}, \mathcal{H}$ of $\mathrm{H}_3(\mathbb{Q})$, it is decidable whether $T \cdot \langle \mathcal{G} \rangle \cap S \cdot \langle \mathcal{H} \rangle = \emptyset$.*

Proof. See Appendix A. ◀

References

1. László Babai, Robert Beals, Jin-yi Cai, Gábor Ivanyos, and Eugene M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 498–507, 1996.
2. László Babai, Paolo Codenotti, Joshua A. Grochow, and Youming Qiao. Code equivalence and group isomorphism. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1395–1408. SIAM, 2011.
3. Henry Frederick Baker. Alternants and continuous groups. *Proceedings of the London Mathematical Society*, 2(1):24–47, 1905.
4. Gilbert Baumslag. *Lecture notes on nilpotent groups*. American Mathematical Society, 2007.
5. Paul C Bell, Mika Hirvensalo, and Igor Potapov. The identity problem for matrix semigroups in SL2(Z) is NP-complete. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 187–206. SIAM, 2017.
6. Paul C. Bell and Igor Potapov. On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups. *International Journal of Foundations of Computer Science*, 21(06):963–978, 2010.

7 Paul C. Bell and Igor Potapov. On the computational complexity of matrix semigroup problems. *Fundamenta Informaticae*, 116(1-4):1–13, 2012.

8 Vincent D. Blondel, Emmanuel Jeandel, Pascal Koiran, and Natacha Portier. Decidable and undecidable problems about quantum automata. *SIAM Journal on Computing*, 34(6):1464–1473, 2005.

9 John Edward Campbell. On a law of combination of operators (second paper). *Proceedings of the London Mathematical Society*, 1(1):14–32, 1897.

10 Christian Choffrut and Juhani Karhumäki. Some decision problems on integer matrices. *RAIRO-Theoretical Informatics and Applications-Informatique Théorique et Applications*, 39(1):125–131, 2005.

11 Henri Cohen. *A course in computational algebraic number theory*, volume 8. Springer-Verlag Berlin, 1993.

12 Thomas Colcombet, Joël Ouaknine, Pavel Semukhin, and James Worrell. On reachability problems for low-dimensional matrix semigroups. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 44:1–44:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.44.

13 Harm Derksen, Emmanuel Jeandel, and Pascal Koiran. Quantum automata and algebraic groups. *Journal of Symbolic Computation*, 39(3-4):357–371, 2005.

14 Ruiwen Dong. On the identity problem and the group problem for nilpotent groups, 2022. Submitted. doi:10.48550/arXiv.2208.02164.

15 Cornelia Druţu and Michael Kapovich. *Geometric group theory*, volume 63. American Mathematical Soc., 2018.

16 Max Garzon and Yechezkel Zalcstein. On isomorphism testing of a class of 2-nilpotent groups. *Journal of Computer and System Sciences*, 42(2):237–248, 1991. doi:10.1016/0022-0000(91)90012-T.

17 Vesa Halava and Tero Harju. On markov's undecidability theorem for integer matrices. In *Semigroup Forum*, volume 75, pages 173–180. Springer, 2007.

18 Felix Hausdorff. Die symbolische Exponentialformel in der Gruppentheorie. *Berichte über die Verhandlungen der Königlich-Sächsischen Gesellschaft der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse*, 58:19–48, 1906.

19 Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of Computational Group Theory*. Chapman and Hall/CRC, 2005.

20 Roger Howe. On the role of the heisenberg group in harmonic analysis. *Bulletin (New Series) of the American Mathematical Society*, 3(2):821–843, 1980.

21 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. Polynomial invariants for affine programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 530–539, 2018.

22 Jun-ichi Igusa. *Theta functions*, volume 194. Springer Science & Business Media, 2012.

23 Mikhail Ivanovich Kargapolov and Jurij Ivanovič Merzljakov. *Fundamentals of the Theory of Groups*, volume 62. Springer, 1979.

24 Olga G. Kharlampovich and Mark V. Sapir. Algorithmic problems in varieties. *International Journal of Algebra and Computation*, 5(04n05):379–602, 1995.

25 Aleksandr Aleksandrovich Kirillov. *Lectures on the orbit method*, volume 64. American Mathematical Soc., 2004.

26 Sang-Ki Ko, Reino Niskanen, and Igor Potapov. On the identity problem for the special linear group and the heisenberg group. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 132:1–132:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.132.

27 Daniel König, Markus Lohrey, and Georg Zetzsche. Knapsack and subset sum problems in nilpotent, polycyclic, and co-context-free groups. *Algebra and Computer Science*, 677:138–153, 2016.

28 V. M. Kopytov. Solvability of the problem of occurrence in finitely generated soluble groups of matrices over the field of algebraic numbers. *Algebra and Logic*, 7(6):388–393, 1968.

29 Hari Krovi and Martin Rötteler. An efficient quantum algorithm for the hidden subgroup problem over Weyl-Heisenberg groups. In *Mathematical Methods in Computer Science*, pages 70–88. Springer, 2008.

30 Engel Lefaucheux. Private Communication, 2022.

31 A. Markov. On certain insoluble problems concerning matrices. *Doklady Akad. Nauk SSSR*, 57(6):539–542, 1947.

32 K. A. Mikhailova. The occurrence problem for direct products of groups. *Matematicheskii Sbornik*, 112(2):241–251, 1966.

33 J. von Neumann. Die Eindeutigkeit der Schrödingerschen Operatoren. *Mathematische Annalen*, 104:570–578, 1931. URL: `http://eudml.org/doc/159483`.

34 Michael S. Paterson. Unsolvability in 3 × 3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.

35 Igor Potapov and Pavel Semukhin. Decidability of the membership problem for 2× 2 integer matrices. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–186. SIAM, 2017.

36 Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

37 Hermann Weyl. *The theory of groups and quantum mechanics*. Courier Corporation, 1950.

38 Jae-Hyun Yang. Harmonic analysis on the quotient spaces of heisenberg groups. *Nagoya mathematical journal*, 123:103–117, 1991.

A Omitted proofs and remarks

▶ **Corollary 2.** *Intersection Emptiness is decidable:*

(i) *in PTIME, for the Heisenberg groups* $\mathrm{H}_n(\mathbb{K})$ *over any algebraic number field* $\mathbb{K}$*, and for any direct product of Heisenberg groups.*

(ii) *for finitely generated 2-step nilpotent groups.*

Proof. (i) By the remark in Section 3, the Heisenberg group $\mathrm{H}_n(\mathbb{K})$ can be embedded as a subgroup of the group $\mathsf{UT}(n', \mathbb{Q})$ for some n', such that the input size only changes at most polynomially. A direct product of Heisenberg groups $\mathrm{H}_{n_1}(\mathbb{K}_1) \times \cdots \times \mathrm{H}_{n_s}(\mathbb{K}_s)$ can hence be embedded as a subgroup of some direct product $\mathsf{UT}(n'_1, \mathbb{Q}) \times \cdots \times \mathsf{UT}(n'_s, \mathbb{Q})$, which is itself a subgroup of $\mathsf{UT}(n'_1 + \cdots + n'_s, \mathbb{Q})$. Again, the input size only changes polynomially during these embeddings. The Heisenberg groups $\mathrm{H}_n(\mathbb{K})$ as well as their direct products are 2-step nilpotent [15, Examples 13.36], and the property of being 2-step nilpotent is preserved under isomorphism. Therefore, Theorem 1 shows that Intersection Emptiness for $\mathrm{H}_n(\mathbb{K})$ as well as for their direct products is decidable in PTIME.

(ii) Given a finite presentation or a consistent polycyclic presentation of G, there exists an embedding $\phi : G \hookrightarrow A \times G_0$ where A is finite and G_0 is a 2-step nilpotent subgroup of some $\mathsf{UT}(n, \mathbb{Q})$. Denote by $\pi_0 : A \times G_0 \to G_0$ the projection onto G_0. The composition $\pi_0 \circ \phi$ can be effectively computed (see proof of [14, Corollary 1.8]).

We claim that $\langle \mathcal{G}_1 \rangle \cap \cdots \cap \langle \mathcal{G}_M \rangle \neq \emptyset$ if and only if $\langle \pi_0(\phi(\mathcal{G}_1)) \rangle \cap \cdots \cap \langle \pi_0(\phi(\mathcal{G}_M)) \rangle \neq \emptyset$. Suppose $g \in \langle \mathcal{G}_1 \rangle \cap \cdots \cap \langle \mathcal{G}_M \rangle$, then obviously $\pi_0(\phi(g)) \in \langle \pi_0(\phi(\mathcal{G}_1)) \rangle \cap \cdots \cap \langle \pi_0(\phi(\mathcal{G}_M)) \rangle$. On the other hand, suppose $h \in \langle \pi_0(\phi(\mathcal{G}_1)) \rangle \cap \cdots \cap \langle \pi_0(\phi(\mathcal{G}_M)) \rangle = \pi_0(\phi(\langle \mathcal{G}_1 \rangle)) \cap \cdots \cap \pi_0(\phi(\langle \mathcal{G}_M \rangle))$, then there exist $a_1, \ldots, a_M \in A$, such that $(a_i, h) \in \langle \mathcal{G}_i \rangle$ for all i. Then $(1, h^{\mathrm{card}(A)}) = (a_i, h)^{\mathrm{card}(A)} \in \langle \mathcal{G}_i \rangle$ for all i, hence $\langle \mathcal{G}_1 \rangle \cap \cdots \cap \langle \mathcal{G}_M \rangle \neq \emptyset$.

Since G_0 is a 2-step nilpotent subgroup of $\mathsf{UT}(n, \mathbb{Q})$, one can decide whether $\langle \pi_0(\phi(\mathcal{G}_1))\rangle \cap \cdots \cap \langle \pi_0(\phi(\mathcal{G}_M))\rangle = \emptyset$ by Theorem 1. Thus, we conclude that it is decidable whether $\langle \mathcal{G}_1\rangle \cap \cdots \cap \langle \mathcal{G}_M\rangle \neq \emptyset$. ◀

We did not attempt to analyse the exact complexity of deciding Intersection Emptiness for arbitrary finitely generated 2-step nilpotent groups. This is because this complexity depends on the computation and representation of the embedding ϕ, as well as the size of the finite group A.

▶ **Theorem 3.** *Given elements $T, S \in \mathrm{H}_3(\mathbb{Q})$ and two finite subsets $\mathcal{G}, \mathcal{H}$ of $\mathrm{H}_3(\mathbb{Q})$, it is decidable whether $T \cdot \langle \mathcal{G}\rangle \cap S \cdot \langle \mathcal{H}\rangle = \emptyset$.*

Proof. As mentioned in the beginning of Section 6, one can without loss of generality suppose $T = I$, and decide whether $\langle \mathcal{G}\rangle \cap S \cdot \langle \mathcal{H}\rangle \neq \emptyset$. Given $\mathcal{G}$ and $\mathcal{H}$, one can effectively compute $\mathcal{C}_\mathcal{G} \cap \mathcal{C}_\mathcal{H}$ and its dimension using linear programming [36].

If $\mathcal{C}_\mathcal{G} \cap \mathcal{C}_\mathcal{H}$ has dimension zero or one, then Proposition 8 shows we can decide whether $\langle \mathcal{G}\rangle \cap S \cdot \langle \mathcal{H}\rangle \neq \emptyset$ by solving a finite number of linear Diophantine equations of the form (29).

If $\mathcal{C}_\mathcal{G} \cap \mathcal{C}_\mathcal{H}$ has dimension two, then Proposition 9 shows we can decide whether $\langle \mathcal{G}\rangle \cap S \cdot \langle \mathcal{H}\rangle \neq \emptyset$ by solving Equations (21), (22) and (23). Equation (23) can be replaced by a boolean combination of conditions of the form "$x_i \equiv 0 \mod 2$", "$x_i \equiv 1 \mod 2$", "$y_i \equiv 0 \mod 2$", ..., or "$d_{ij} \equiv 1 \mod 2$". Each of these conditions can be expressed as a linear equation over integers, for example "$x_i \equiv 1 \mod 2$" is equivalent to "$x_i = 2x_i' + 1, x_i' \in \mathbb{Z}$". Therefore, solving Equations (21), (22) and (23) is equivalent to solving a boolean combination of linear equations over integers, which is decidable by integer programming. ◀

▶ **Proposition 8.** *Suppose the cone $\mathcal{C}_\mathcal{G} \cap \mathcal{C}_\mathcal{H}$ has dimension zero or one. Deciding whether $\langle \mathcal{G}\rangle \cap S \cdot \langle \mathcal{H}\rangle \neq \emptyset$ can be done by solving finitely many linear Diophantine equations.*

Proof. Let $\mathcal{L} \subseteq \mathbb{Q}^2$ be a linear space of dimension one that contains $\mathcal{C}_\mathcal{G} \cap \mathcal{C}_\mathcal{H}$. Then we decompose $\mathcal{G}$ and $\mathcal{H}$ into disjoint subsets: $\mathcal{G} = \mathcal{G}_0 \cup \mathcal{G}_+$, $\mathcal{H} = \mathcal{H}_0 \cup \mathcal{H}_+$, where

$$\mathcal{G}_0 \coloneqq \{A_i \in \mathcal{G} \mid \varphi(\log A_i) \in \mathcal{L}\}, \quad \mathcal{G}_+ \coloneqq \mathcal{G} \setminus \mathcal{G}_0;$$
$$\mathcal{H}_0 \coloneqq \{B_i \in \mathcal{H} \mid \varphi(\log B_i) \in \mathcal{L}\}, \quad \mathcal{H}_+ \coloneqq \mathcal{H} \setminus \mathcal{H}_0.$$

The key observation is that all matrices in $\mathcal{G}_0$ and in $\mathcal{H}_0$ commute with each other (all $\varphi(\log A_i)$ and $\varphi(\log B_j)$ are linearly dependant, so $[\log A_i, \log A_j] = [\log B_i, \log B_j] = 0$).

Suppose $\langle \mathcal{G}\rangle \cap S \cdot \langle \mathcal{H}\rangle \neq \emptyset$, that is, there exist words v in the alphabet $\mathcal{G}$ and w in the alphabet $\mathcal{H}$ such that $\log v = \log Sw$. We show that the number of occurrences of letters of $\mathcal{G}_+$ in v is bounded; similarly, the number of occurrences of letters of $\mathcal{H}_+$ in w is bounded.

Let $\boldsymbol{n}$ be a non-zero vector orthogonal to $\mathcal{L}$, then $\boldsymbol{x} \mapsto \boldsymbol{n}^\top \boldsymbol{x}$ is the projection parallel to $\mathcal{L}$. Since $\mathcal{C}_\mathcal{G} \cap \mathcal{C}_\mathcal{H} \subseteq \mathcal{L}$, the values $\boldsymbol{n}^\top \varphi(\log A_i), A_i \in \mathcal{G}$ have signs opposite to that of $\boldsymbol{n}^\top \varphi(\log B_j), B_j \in \mathcal{H}$. Without loss of generality, suppose $\boldsymbol{n}^\top \varphi(\log A_i) \geq 0$ for all $A_i \in \mathcal{G}$ and $\boldsymbol{n}^\top \varphi(\log B_j) \leq 0$ for all $B_j \in \mathcal{H}$. Since $\boldsymbol{n}$ is orthogonal to $\mathcal{L}$, we have furthermore $\boldsymbol{n}^\top \varphi(\log A_i) > 0$ for all $A_i \in \mathcal{G}_+$ and $\boldsymbol{n}^\top \varphi(\log B_j) < 0$ for all $B_j \in \mathcal{H}_+$; as well as $\boldsymbol{n}^\top \varphi(\log X) = 0$ for all $X \in \mathcal{G}_0 \cup \mathcal{H}_0$.

Now, $\log v = \log Sw$ yields $\varphi(\log v) = \varphi(\log S) + \varphi(\log w)$. Projecting onto $\boldsymbol{n}$, this shows

$$\sum_{i, A_i \in \mathcal{G}_+} \mathrm{PI}_i^\mathcal{G}(v) \cdot \boldsymbol{n}^\top \varphi(\log A_i) = \boldsymbol{n}^\top \varphi(S) + \sum_{i, B_i \in \mathcal{H}_+} \mathrm{PI}_i^\mathcal{H}(w) \cdot \boldsymbol{n}^\top \varphi(\log B_i).$$

This yields

$$\mathrm{PI}_i^\mathcal{G}(v) \leq \frac{\boldsymbol{n}^\top \varphi(\log S)}{\boldsymbol{n}^\top \varphi(\log A_i)}, \quad \mathrm{PI}_j^\mathcal{H}(v) \leq \frac{\boldsymbol{n}^\top \varphi(\log S)}{\boldsymbol{n}^\top \varphi(\log B_j)}, \tag{28}$$

for all $A_i \in \mathcal{G}_+$ and $B_j \in \mathcal{H}_+$. This gives bounds $\beta_{\mathcal{G}} \coloneqq \sum_{i,A_i\in\mathcal{G}_+} \frac{\boldsymbol{n}^\top \varphi(\log S)}{\boldsymbol{n}^\top \varphi(\log A_i)}$ and $\beta_{\mathcal{H}} \coloneqq \sum_{i,B_i\in\mathcal{H}_+} \frac{\boldsymbol{n}^\top \varphi(\log S)}{\boldsymbol{n}^\top \varphi(\log B_i)}$, such that if $\log v = \log Sw$, then the number of letters of $\mathcal{G}_+$ in v is bounded by $\beta_{\mathcal{G}}$; and similarly the number of letters of $\mathcal{H}_+$ in w is bounded by $\beta_{\mathcal{H}}$.

Write $v = v_0C_1v_1C_2\cdots v_{s-1}C_sv_s$, where $C_1,\ldots,C_s$ are matrices in $\mathcal{G}_+$, and $v_0,\ldots,v_s$ are words in the alphabet $\mathcal{G}_0$. Similarly, write $w = w_0D_1w_1D_2\cdots w_{t-1}D_tw_t$, where $D_1,\ldots,D_t$ are matrices in $\mathcal{H}_+$, and $w_0,\ldots,w_t$ are words in the alphabet $\mathcal{H}_0$. Write $\mathcal{G}_0 = \{A'_1,\ldots,A'_{K'}\}$ and $\mathcal{H}_0 = \{B'_1,\ldots,B'_{M'}\}$. Define $x_{ij} \coloneqq \mathrm{PI}_j^{\mathcal{G}_0}(v_i)$ for $0 \le i \le s, 1 \le j \le K'$, and $y_{ij} \coloneqq \mathrm{PI}_j^{\mathcal{H}_0}(w_i)$ for $0 \le i \le t, 1 \le j \le M'$. Then $\log v = \log Sw$ is equivalent to

$$\begin{aligned}
&\sum_{i=1}^{s} \log C_i + \sum_{i=0}^{s}\sum_{j=1}^{K'} x_{ij}\log A'_j + \frac{1}{2}\sum_{0\le i<k\le s}\sum_{j=1}^{K'} x_{ij}[\log A'_j, \log C_k] \\
&+ \frac{1}{2}\sum_{1\le k\le i\le s}\sum_{j=1}^{K'} x_{ij}[\log C_k, \log A'_j] \\
&= \log S + \sum_{i=1}^{t}(\log D_i + \frac{1}{2}[\log S, \log D_i]) + \frac{1}{2}\sum_{i=0}^{t}\sum_{j=1}^{M'} y_{ij}[\log S, \log B'_j] \\
&\qquad + \frac{1}{2}\sum_{0\le i<k\le t}\sum_{j=1}^{M'} y_{ij}[\log B'_j, \log D_k] + \frac{1}{2}\sum_{1\le k\le i\le t}\sum_{j=1}^{M'} y_{ij}[\log D_k, \log B'_j] \quad (29)
\end{aligned}$$

All other terms contain $[\log A'_i, \log A'_j]$ or $[\log B'_i, \log B'_j]$ and hence vanish by the commutativity of $\mathcal{G}_0$ and $\mathcal{H}_0$. Note that Equation (29) is a linear Diophantine equation in the variables x_{ij}, y_{ij}. Therefore, $\log v = \log Sw$ has a solution if and only if there exist matrices $C_1,\ldots,C_s$ in $\mathcal{G}_+$ and matrices $D_1,\ldots,D_t$ in $\mathcal{H}_+$, such that Equation (29) has a solution in non-negative integers, with the additional constraint that, if $s = 0$, then $(x_{01},\ldots,x_{0K'}) \ne \mathbf{0}$; and if $t = 0$, then $(y_{01},\ldots,y_{0M'}) \ne \mathbf{0}$. This additional constraint comes from the condition that v, w are not empty words. Recall the bounds $s \le \beta_{\mathcal{G}}$ and $t \le \beta_{\mathcal{H}}$. Hence, deciding whether $\log v = \log Sw$ has a solution amounts to solving finitely many linear Diophantine equations of the form (29). ◀

In theory, it is possible to give a bound on the complexity of the procedure described in Proposition 8. The size of the each bound in Equation (28) is exponential in the bit size of the entries $S, \mathcal{G}, \mathcal{H}$. Hence the procedure consists of solving exponentially many linear Diophantine equations.

▶ **Proposition 10.** *Algorithm 1 is correct and terminates in polynomial time.*

Proof. We prove that Algorithm 1 outputs **False** if and only if $\langle\mathcal{G}_1\rangle \cap \cdots \cap \langle\mathcal{G}_M\rangle \ne \emptyset$.

After each iteration of Step 2, $\mathrm{card}(S_1) + \cdots + \mathrm{card}(S_M)$ strictly decreases. Therefore, the algorithm terminates after at most $K_1 + \cdots + K_M$ iterations of Step 2.

We now show correctness of the algorithm. We first show that when Algorithm 1 returns **False**, then $\bigcap_{i=1}^{M}\langle\mathcal{G}_i\rangle \ne \emptyset$. Suppose the algorithm terminates with output False, the condition in Step 2(d) shows that $\mathrm{supp}(\Lambda) \cap S_m = S_m$ for all $1 \le m \le M$. By the additivity of Λ (that is, $\boldsymbol{a}, \boldsymbol{b} \in \Lambda \implies \boldsymbol{a} + \boldsymbol{b} \in \Lambda$), there exists a vector $\boldsymbol{\ell} = (\boldsymbol{\ell}_1,\ldots,\boldsymbol{\ell}_M) \in \Lambda$ such that $\mathrm{supp}(\boldsymbol{\ell}) = \mathrm{supp}(\Lambda)$. This yields $\mathrm{supp}(\boldsymbol{\ell}_m) = \mathrm{supp}(\Lambda) \cap S_m = S_m$ for all m. Since $\mathrm{supp}(\boldsymbol{\ell}_m) = S_m \ne \emptyset$, we have $\boldsymbol{\ell}_m \ne \mathbf{0}$ for all $1 \le m \le M$. By the definition (19) of $\pi_{\boldsymbol{\ell}}(W)$, there exist rational numbers $(c_{mij})_{1\le m\le M, i,j\in S_m}$ such that

$$\sum_{j=1}^{K_1} \ell_{1j} \log A_{1j} + \sum_{\substack{i<j \\ i,j\in S_1}} c_{1ij}[\log A_{1i}, \log A_{1j}] = \sum_{j=1}^{K_2} \ell_{2j} \log A_{2j} + \sum_{\substack{i<j \\ i,j\in S_2}} c_{2ij}[\log A_{2i}, \log A_{2j}]$$

$$= \cdots = \sum_{j=1}^{K_M} \ell_{Mj} \log A_{Mj} + \sum_{\substack{i<j \\ i,j\in S_M}} c_{Mij}[\log A_{Mi}, \log A_{Mj}] \quad (30)$$

Since $S_m = \mathrm{supp}(\boldsymbol{\ell}_m)$ for all m, Equation (30) is identical to Equation (14) in Proposition 7. Therefore Proposition 7 shows $\bigcap_{i=1}^{M} \langle \mathcal{G}_i \rangle \neq \emptyset$.

Next, we show that if $\bigcap_{i=1}^{M} \langle \mathcal{G}_i \rangle \neq \emptyset$, then Algorithm 1 returns **False**. Suppose $\bigcap_{i=1}^{M} \langle \mathcal{G}_i \rangle \neq \emptyset$. By Proposition 7, there exist $\boldsymbol{\ell}_1 = (\ell_{1j})_{1\le j\le K_1} \in \mathbb{Z}_{\ge 0}^{K_1} \setminus \{\mathbf{0}\}, \ldots, \boldsymbol{\ell}_M = (\ell_{Mj})_{1\le j\le K_M} \in \mathbb{Z}_{\ge 0}^{K_M} \setminus \{\mathbf{0}\}$, and rational numbers $(c_{mij})_{1\le m\le M, i,j\in \mathrm{supp}(\boldsymbol{\ell}_m)}$ that satisfies Equation (14) in Proposition 7. We show that "$\mathrm{supp}(\boldsymbol{\ell}_m) \subseteq S_m$ for all $1 \le m \le M$" is an invariant of the algorithm.

At initialization, we obviously have $\mathrm{supp}(\boldsymbol{\ell}_m) \subseteq S_m = \{1, \ldots, K_m\}$. Before each iteration of 2(d), suppose we have $\mathrm{supp}(\boldsymbol{\ell}_m) \subseteq S_m$ for all m, then Equation (14) shows that $(\ell_{mj})_{1\le m\le M, 1\le j\le K_m} \in \pi_{\boldsymbol{\ell}}(W)$. Consequently, $\mathrm{supp}(\boldsymbol{\ell}_m) \subseteq \mathrm{supp}(\Lambda)$, meaning $\mathrm{supp}(\boldsymbol{\ell}_m) \subseteq S_m$ still holds after 2(d).

This invariant shows that $\mathrm{supp}(\boldsymbol{\ell}_m) \subseteq S_m$ for all m by the start of Step 3. Since $\boldsymbol{\ell}_m \in \mathbb{Z}_{\ge 0}^{K_m} \setminus \{\mathbf{0}\}$, $\mathrm{supp}(\boldsymbol{\ell}_m)$ is non-empty for every m. We conclude that $S_m \neq \emptyset$ for all m by the start of Step 3. Therefore, Algorithm 1 returns **False**.

Finally, we show that Algorithm 1 terminates in polynomial time. Recall that the algorithm terminates after at most $K_1 + \cdots + K_M$ iterations of Step 2. At each iteration of Step 2(b), the projection can be computed in polynomial time by eliminating the variables $(c_{mij})_{1\le m\le M, i,j\in S_m}$ from the equations defining W. Then, at each iteration of Step 2(c) the support $\mathrm{supp}(\Lambda)$ is computed by Lemma 4. The total input size of the linear programming instances is polynomial with respect to the total bit length of the matrix entries in $\mathcal{G}_1, \ldots, \mathcal{G}_M$. Indeed, the total bit length of $\log A_{mi}$ and $[\log A_{mi}, \log A_{mj}]$ is at most of quadratic size in $\mathcal{G}_m$; and the projection performed in Step 2(b) can only alter the total entry bit size at most polynomially. From this, one can express $\pi_{\boldsymbol{\ell}}(W)$ as the solution set of a system of homogeneous linear equations whose total bit length is polynomial in $\mathcal{G}_1, \ldots, \mathcal{G}_M$. Hence Lemma 4 computes the support of $\Lambda \coloneqq \mathbb{Z}_{\ge 0}^{\sum_{m=1}^{M} K_m} \cap \pi_{\boldsymbol{\ell}}(W)$ in polynomial time. Therefore, each iteration of Step 2 takes polynomial time, and thus the overall complexity of Algorithm 1 is polynomial with respect to the input $\mathcal{G}_1, \ldots, \mathcal{G}_M$. ◀

Solving Homogeneous Linear Equations over Polynomial Semirings

Ruiwen Dong ✉
Department of Computer Science, University of Oxford, UK

Abstract

For a subset B of $\mathbb{R}$, denote by $\mathrm{U}(B)$ be the semiring of (univariate) polynomials in $\mathbb{R}[X]$ that are strictly positive on B. Let $\mathbb{N}[X]$ be the semiring of (univariate) polynomials with non-negative integer coefficients. We study solutions of homogeneous linear equations over the polynomial semirings $\mathrm{U}(B)$ and $\mathbb{N}[X]$. In particular, we prove local-global principles for solving single homogeneous linear equations over these semirings. We then show PTIME decidability of determining the existence of non-zero solutions over $\mathbb{N}[X]$ of single homogeneous linear equations.

Our study of these polynomial semirings is largely motivated by several semigroup algorithmic problems in the wreath product $\mathbb{Z} \wr \mathbb{Z}$. As an application of our results, we show that the Identity Problem (whether a given semigroup contains the neutral element?) and the Group Problem (whether a given semigroup is a group?) for finitely generated sub-semigroups of the wreath product $\mathbb{Z} \wr \mathbb{Z}$ is decidable when elements of the semigroup generator have the form $(y, \pm 1)$.

2012 ACM Subject Classification Computing methodologies → Symbolic and algebraic manipulation

Keywords and phrases wreath product, identity problem, polynomial semiring, positive polynomial

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.26

Funding The author acknowledges support from UKRI Frontier Research Grant EP/X033813/1.

Acknowledgements The author would like to thank Markus Schweighofer for useful discussions and feedback and for pointing out the references [24] and [25].

1 Introduction

Linear equations over semirings appear in various domains in mathematics and computer science, such as automata theory, optimization, and algebra of formal processes [2, 3, 6, 11]. There have been numerous studies on linear equations over different semirings [12], for example the semiring of natural numbers (integer programming), tropical semirings [23] and polynomial semirings [9, 22]. Given a semiring S, define $S[X]$ to be the set of polynomials in variable X whose coefficients are elements of S. The set $S[X]$ is again a semiring. One of the simplest polynomial semirings is the semiring $\mathbb{N}[X]$ of single variable polynomials with non-negative integer coefficients. The problem of solving a system of linear equations over $\mathbb{N}[X]$ was shown to be undecidable by Narendran [22] using a reduction from Hilbert's tenth problem. More precisely, given integer polynomials $h_{ij}, g_j \in \mathbb{Z}[X], i = 1, \ldots, n, j = 1, \ldots, k$, it is undecidable whether the system of equations

$$f_1 h_{1j} + \cdots + f_n h_{nj} = g_j, \quad j = 1, \ldots, k, \tag{1}$$

has a solution $(f_1, \ldots, f_n)$ over $\mathbb{N}[X]$. This contrasts with the decidability of solving systems of linear equations over $\mathbb{N}$ and over $\mathbb{Z}[X]$ (using respectively integer programming [15] and Smith canonical forms [16]).


© Ruiwen Dong;
licensed under Creative Commons License CC-BY 4.0


In this paper, we show a decidability result for finding a non-zero solution of a *single homogeneous* linear equation over $\mathbb{N}[X]$. In particular, we are concerned with the following problem: given integer polynomials $h_1, \ldots, h_n \in \mathbb{Z}[X]$, does the equation

$$f_1 h_1 + \cdots + f_n h_n = 0 \tag{2}$$

admit a solution $(f_1, \ldots, f_n)$ over $\mathbb{N}[X] \setminus \{0\}$ (i.e. none of the f_i is zero)?

In Section 6 of this paper we give a PTIME algorithm that decides this problem. Our algorithm relies on a local-global principle which we prove in Section 5, and reduces the decision problem to the existential theory of the reals in one variable. Formal definitions of these results will be given in Section 2.

It turns out that the problem of solving linear equations over the semiring $\mathbb{N}[X]$ is closely related to solving the same equation over the semiring $\mathrm{U}(B)$, consisting of polynomials in $\mathbb{R}[X]$ that are *strictly positive* on a subset B of $\mathbb{R}$. It is also related to the semiring $\mathrm{W}(B)$ of polynomials that are *non-negative* on B. The characterization of polynomials in $\mathrm{U}(B)$ and $\mathrm{W}(B)$ is a central subject in the theory of real algebra. In particular, when B is a semialgebraic set, variants of the *positivstellensatz* give explicit descriptions of the semirings $\mathrm{U}(B)$ and $\mathrm{W}(B)$. This theory can be traced back to the celebrated Hilbert's seventeenth problem: given a polynomial that takes only non-negative values over the reals, can it be represented as a sum of squares of rational functions? This has been answered positively by Artin [1] using a model theoretic approach. The techniques proposed by Artin have since developed into the rich theory of real algebra; for a comprehensive account of this subject, see [24] or [25]. An important result in solving homogeneous linear equations over $\mathrm{W}(\mathbb{R})$ is the Bröcker-Prestel's local-global principle for weak isotropy of quadratic forms [24, Theorem 8.12, 8.13]. Applied over the function field $\mathbb{R}(X)$, the Bröcker-Prestel local-global principle relates the existence of non-trivial solutions over *sums of squares* in $\mathbb{R}(X)$ (and hence over $\mathrm{W}(\mathbb{R})$) of a homogeneous linear equation, to the behaviour of the equation in all *Henselizations* of $\mathbb{R}(X)$. In Section 4 of this paper we prove a "strictly positive" version of the Bröcker-Prestel local-global principle, which characterizes the existence of solutions over $\mathrm{U}(B)$. This will serve as a base for proving further results in Section 5 and 6. Our proof is inspired by Prestel's proof of the original theorem. However, several new ideas are introduced to deal with the strict positivity as well as the positivity constraint over a subset of $\mathbb{R}$.

An important motivation for studying linear equations over $\mathbb{N}[X]$ comes from a semigroup algorithmic problem in the *wreath product* $\mathbb{Z} \wr \mathbb{Z}$. The wreath product is a fundamental construction in group and semigroup theory. Given two groups G and H, their wreath product $G \wr H$ is defined in the following way. Let G^H be the set of all functions $y \colon H \to G$ with finite support; it is a group with respect to pointwise multiplication. The group H acts on G^H as a group of automorphisms: if $h \in H, y \in G^H$, then $y^h(b) = y(bh^{-1})$ for all $b \in H$. The wreath product $G \wr H$ is then defined as the semi-direct product $G^H \rtimes H$, that is, the set of all pairs (y, h) where $y \in G^H, h \in H$, with multiplication operation given by

$$(y, h)(z, k) = (y^k z, hk).$$

One easy way to understand the group $\mathbb{Z} \wr \mathbb{Z}$ is through its isomorphism to a matrix group over the Laurent polynomial ring $\mathbb{Z}[X, X^{-1}]$ [19]:

$$\varphi \colon \mathbb{Z} \wr \mathbb{Z} \xrightarrow{\sim} \left\{ \begin{pmatrix} 1 & f \\ 0 & X^b \end{pmatrix} \;\middle|\; f \in \mathbb{Z}[X, X^{-1}], b \in \mathbb{Z} \right\}, \quad (y, b) \mapsto \begin{pmatrix} 1 & \sum_{k \in \mathbb{Z}} y(k) X^k \\ 0 & X^b \end{pmatrix}. \tag{3}$$

A large number of important groups are constructed using the wreath product, such as the lamplighter group $\mathbb{Z}_2 \wr \mathbb{Z}$ [13] and groups resulting from the Magnus embedding theorem [19]. The wreath product also plays an important role in the algebraic theory of automata. The Krohn–Rhodes theorem states that every finite semigroup (and correspondingly, every finite automaton) can be decomposed into elementary components using wreath products [17].

In Section 7 we give an application of our results to the *Identity Problem* in $\mathbb{Z} \wr \mathbb{Z}$. Given a finite set of elements $\mathcal{G} = \{A_1, \ldots, A_k\}$ in a group G as well as a target element $A \in G$, denote by $\langle \mathcal{G} \rangle$ the semigroup of generated by $\mathcal{G}$, and by $\langle \mathcal{G} \rangle_{grp}$ the group generated by $\mathcal{G}$. Consider the following decision problems:

(i) *(Group Membership Problem)* whether $A \in \langle \mathcal{G} \rangle_{grp}$?
(ii) *(Semigroup Membership Problem)* whether $A \in \langle \mathcal{G} \rangle$?
(iii) *(Identity Problem)* whether the neutral element I of G is contained in $\langle \mathcal{G} \rangle$?

All three problems remain undecidable even when the ambient group G is restricted to relatively simple groups, such as the direct product $F_2 \times F_2$ of two free groups over two generators [5, 21]. Indeed, one of the first undecidability results in algorithmic theory was the undecidability of the Semigroup Membership Problem for integer matrices, obtained by Markov [20]. Some decidability results for the Identity Problem include its NP-completeness in $\mathsf{SL}(2, \mathbb{Z})$ [4] and its PTIME decidability in nilpotent groups of class at most ten [10].

Let $p \in \mathbb{Z}_{>0}$. The group $\mathbb{Z} \wr \mathbb{Z}$ shares some common properties with the wreath product $(\mathbb{Z}/p\mathbb{Z}) \wr \mathbb{Z}$ and with the Baumslag-Solitar group $\mathsf{BS}(1, p)$. Similar to the isomorphism (3), both $(\mathbb{Z}/p\mathbb{Z}) \wr \mathbb{Z}$ and $\mathsf{BS}(1, p)$ can be represented as 2×2 upper triangular matrix groups:

$$(\mathbb{Z}/p\mathbb{Z}) \wr \mathbb{Z} \cong \left\{ \begin{pmatrix} 1 & f \\ 0 & X^b \end{pmatrix} \;\middle|\; f \in (\mathbb{Z}/p\mathbb{Z})[X, X^{-1}], b \in \mathbb{Z} \right\},$$

$$\mathsf{BS}(1, p) \cong \left\{ \begin{pmatrix} 1 & f \\ 0 & p^b \end{pmatrix} \;\middle|\; f \in \mathbb{Z}[1/p], b \in \mathbb{Z} \right\}.$$

Lohrey, Steinberg and Zetzsche showed decidability of the *Rational Subset Membership Problem* (which subsumes all three decision problems mentioned above) in $H \wr V$, where H is a finite and V is virtually free [18]. This notably implies its decidability in $(\mathbb{Z}/p\mathbb{Z}) \wr \mathbb{Z}$. Cadilhac, Chistikov and Zetzsche proved its decidability in $\mathsf{BS}(1, p)$ [7]. For $\mathbb{Z} \wr \mathbb{Z}$, decision problems are much harder due to higher encoding power of the ring $\mathbb{Z}[X, X^{-1}]$. The Group Membership Problem in $\mathbb{Z} \wr \mathbb{Z}$ can be reduced to the membership problem for modules over the ring $\mathbb{Z}[X, X^{-1}]$, and is hence decidable [26]. As for the Semigroup Membership Problem in $\mathbb{Z} \wr \mathbb{Z}$, Lohrey *et al.* showed its undecidability using an encoding of 2-counter machines [18]. Decidability of the Identity Problem in $\mathbb{Z} \wr \mathbb{Z}$ remains an intricate open problem. In this paper we give a decidability result in the case where all the elements of the generator $\mathcal{G}$ are of the form $(y, \pm 1)$.

2 Main results

In this section we sum up the main results of this paper. For a subset B of $\mathbb{R}$, denote by $\mathrm{U}(B)$ the set of polynomials in $\mathbb{R}[X]$ that are strictly positive on B:

$$\mathrm{U}(B) \coloneqq \{f \in \mathbb{R}[X] \mid f(x) > 0 \text{ for all } x \in B\}.$$

Define $\overline{B}$ to be the closure of B in $\mathbb{R}$ under the Euclidean topology. Our first result is a local-global principle for solutions of homogeneous linear equations over $\mathrm{U}(B)$. Theorem 2.1 will be proved in Section 4.

▶ **Theorem 2.1.** *Let B be a subset of $\mathbb{R}$. Assume that $h_1, \ldots, h_n \in \mathbb{R}[X]$ are polynomials that satisfy $\gcd(h_1, \ldots, h_n) = 1$. If the equation $f_1h_1 + \cdots + f_nh_n = 0$ has no solution $(f_1, \ldots, f_n)$ over $\mathrm{U}(B)$, then there exists a real number $t \in \overline{B}$, such that the values $h_i(t), i = 1, \ldots, n$, are either all non-negative or all non-positive.*

Our second result is a corollary of the previous theorem, it provides a similar local-global principle for solutions over $\mathbb{N}[X] \setminus \{0\}$. Theorem 2.2 will be proved in Section 5.

▶ **Theorem 2.2.** *Given polynomials $h_1, \ldots, h_n \in \mathbb{Z}[X]$ with $\gcd(h_1, \ldots, h_n) = 1$. If the equation $f_1h_1 + \cdots + f_nh_n = 0$ has no solution $(f_1, \ldots, f_n)$ over $\mathbb{N}[X] \setminus \{0\}$, then there exists $t \in \mathbb{R}_{\geq 0}$, such that the values $h_i(t), i = 1, \ldots, n$ are either all non-negative or all non-positive.*

Our next result shows that it is decidable in PTIME whether a linear homogeneous equation is solvable over $\mathbb{N}[X] \setminus \{0\}$. The input size is defined as the total number of bits used to encode all the coefficients of all h_i. Theorem 2.3 will be proved in Section 6.

▶ **Theorem 2.3.** *Given as input $h_1, \ldots, h_n \in \mathbb{Z}[X]$. It is decidable in polynomial time whether the equation $f_1h_1 + \cdots + f_nh_n = 0$ has solutions $f_1, \ldots, f_n$ over $\mathbb{N}[X] \setminus \{0\}$.*

An application of this theorem is the following partial decidability result on the Identity Problem in the wreath product $\mathbb{Z} \wr \mathbb{Z}$. This will be the main topic of Section 7.

▶ **Theorem 2.4.** *Given a finite set of elements $\mathcal{G} = \{(y_1, b_1), \ldots, (y_n, b_n)\}$ in $\mathbb{Z} \wr \mathbb{Z}$, where $b_i = \pm 1$ for all i. The following are decidable:*
1. *(Group Problem) whether the semigroup $\langle \mathcal{G} \rangle$ generated by $\mathcal{G}$ is a group.*
2. *(Identity Problem) whether the neutral element I is in the semigroup $\langle \mathcal{G} \rangle$.*

3 Preliminaries

In this section we introduce the necessary mathematical tools on (semi)orderings of fields as well as valuations. Most notations and definitions follow those given in Prestel's book [24].

3.1 Orderings and semiorderings

▶ **Definition 3.1** (Ordering). A linear ordering of a set S is a binary relation that satisfies
(i) $a \leq a$,
(ii) $a \leq b, b \leq c \implies a \leq c$,
(iii) $a \leq b, b \leq a \implies a = a$,
(iv) $a \leq b$ or $b \leq a$
for all $a, b, c \in S$.

Given a field F of characteristic zero, a *(field) ordering* of F is a linear ordering $\leq$ of the underlying set of F that additionally satisfies
(i) $a \leq b \implies a + c \leq b + c$,
(ii) $0 \leq a, 0 \leq b \implies 0 \leq ab$
for all $a, b, c \in F$. A field is called *formally real* if it admits at least one ordering.

The *semiordering* of a field, defined below, is a weaker version of the field ordering.

▶ **Definition 3.2** (Semiordering). A *semiordering* of a field F is a linear ordering $\leq$ of the underlying set of F that satisfies
(i) $a \leq b \implies a + c \leq b + c$,
(ii) $0 \leq 1$,
(iii) $0 \leq a \implies 0 \leq ab^2$
for all $a, b, c \in F$.

In a field F with semiordering $\leq$, we have $0 \leq x^2$ for all $x \in F$. The field of real numbers $\mathbb{R}$ hence admits a unique semiordering, since every positive real can be written as a square. This semiordering is simply the natural ordering on $\mathbb{R}$.

It is easy to see that an ordering is always a semiordering. Conversely, a semiordering need not be an ordering. However, in any field, the existence of a semiordering implies that of an ordering.

▶ **Lemma 3.3** ([24, Corollary 1.15]). *A field F is formally real (admits an ordering) if and only if it admits a semiordering.*

For any subset P of F, define $-P := \{-x \mid x \in P\}$. For a semiordering $\leq$ of F, the set $P := \{a \in F \mid 0 \leq a\}$ satisfies

(i) $P + P \subseteq P$,
(ii) $F^2 \cdot P \subseteq P$ and $1 \in P$,
(iii) $P \cap -P = \{0\}$,
(iv) $P \cup -P = F$.

Such a set will be called a *semicone* of F. A semicone P of F determines a semiordering $\leq$ of F by $a \leq b \iff b - a \in P$. Therefore, we will sometimes call P a semiordering as well.

The *pre-semicone* is yet a weaker version of the semiordering (or semicone).

▶ **Definition 3.4** (Pre-semicone). A *pre-semicone* of a field F is a subset P of F that satisfies

(i) $P + P \subseteq P$,
(ii) $F^2 \cdot P \subseteq P$,
(iii) $P \cap -P = \{0\}$.

The only difference between a pre-semicone and a semicone is the absence of the rule (iv) and the condition $1 \in P$ in (ii). Obviously every semicone is also a pre-semicone. Conversely, a pre-semicone need not be a semicone, but it can always be extended to one.

▶ **Lemma 3.5** ([24, Lemma 1.13]). *For every pre-semicone P_0 of a formally real field F there exists a set $P \supseteq P_0$ such that P or $-P$ is a semicone of F.*

Suppose F is of characteristic zero. A semiordering or an ordering $\leq$ of F is called *archemedean* if for each $a \in F$ there exists $n \in \mathbb{N} \subseteq F$ such that $a \leq n$.

▶ **Lemma 3.6** ([24, Lemma 1.20]). *Every archimedean semiordering is an ordering.*

3.2 Valuations

Let F be a field. A *valuation* of F is a surjective map $v\colon F \to \Gamma \cup \{\infty\}$, where the *value group* Γ is an abelian totally ordered group[1], such that the following conditions are satisfied for all $a, b \in F$:

(i) $v(a) = \infty$ if and only if $a = 0$,
(ii) $v(ab) = v(a) + v(b)$,
(iii) $v(a + b) \geq \min\{v(a), v(b)\}$, with equality if $v(a) \neq v(b)$.

A valuation is called *non-trivial* if $\Gamma \neq \{0\}$. A *valued field* is a pair (F, v) where F is a field and v is a valuation of F. Its *valuation ring* A_v is defined as

$$A_v := \{a \in F \mid v(a) \geq 0\}.$$

[1] An abelian totally ordered group Γ is an abelian group equipped with a linear ordering $\leq$, such that $a \leq b \implies a + c \leq b + c$ for all $a, b, c \in \Gamma$. Here, the group law of Γ is written additively. The ordering and the group law on Γ can be extended to the set $\Gamma \cup \{\infty\}$ by defining $a \leq \infty$ and $a + \infty = \infty + a = \infty + \infty = \infty$ for all $a \in \Gamma$.

We have $A_v \neq F$ if and only if v is non-trivial. A_v is a ring with a unique maximal ideal

$$M_v \coloneqq \{a \in F \mid v(a) > 0\}.$$

The quotient $F_v \coloneqq A_v/M_v$ is called the *residue field* of (F, v). It is indeed a field since M_v is maximal. A valuation v is called a *real place* of F if the residue field F_v is formally real.

Consider the field $F = \mathbb{R}(X)$. The following proposition gives a well-known characterization (up to isomorphism of the value group Γ) of the set of all non-trivial real places $\mathbb{R}(X)$ whose valuation ring contains the subfield $\mathbb{R}$.

▶ **Proposition 3.7.** *Let v be a non-trivial real place of $\mathbb{R}(X)$ such that $\mathbb{R} \subseteq A_v$. Then v belongs to one of the two following types of real places:*

1. *For every $t \in \mathbb{R}$ there is a real place $v_t \colon \mathbb{R}(X) \to \mathbb{Z} \cup \{\infty\}$, defined by $v_t(y) = a$, where $a \in \mathbb{Z}$ is such that y can be written as $y = (X-t)^a \cdot \frac{f}{g}$, with f, g being polynomials in $\mathbb{R}[X]$ not divisible by $X - t$. The residue field $\mathbb{R}(X)_{v_t}$ is isomorphic to $\mathbb{R}$ by the natural homomorphism $y + M_{v_t} \mapsto y(t)$.*
2. *There is a real place $v_\infty \colon \mathbb{R}(X) \to \mathbb{Z} \cup \{\infty\}$, defined by $v_t(\frac{f}{g}) = \deg g - \deg f$, where f, g are polynomials in $\mathbb{R}[X]$. The residue field $\mathbb{R}(X)_{v_\infty}$ is isomorphic to $\mathbb{R}$ by the natural homomorphism $y + M_{v_\infty} \mapsto \lim_{t\to\infty} y(t)$.*

Let P be a semicone of a field F, and F_0 be a subfield of F. Denote by $\leq$ the corresponding semiordering of P; define the set

$$A^P_{F_0} \coloneqq \{a \in F \mid a \leq b \text{ and } -a \leq b \text{ for some } b \in F_0\}. \tag{4}$$

The following lemmas show that $A^P_{F_0}$ is a valuation ring, and that its corresponding residue field admits a semiordering induced by P under additional conditions.

▶ **Lemma 3.8** ([24, Lemma 7.13])**.** *Let P be a semiordering of a field F and F_0 a subfield of F. Then $A^P_{F_0}$ is a valuation ring of some valuation of F.*

▶ **Lemma 3.9** ([24, Lemma 7.15])**.** *Let P be a semiordering of a field F and F_0 a subfield of F, such that there exists $b \in F$ with $a \leq b$ for all $a \in F_0$. Let the valuation v of F correspond to $A^P_{F_0}$. Then $(A_v \cap P)/M_v$ is a semiordering of F_v.*

4 Local-global principle over strictly positive polynomials

For a subset B of $\mathbb{R}$, define the set $\mathrm{W}(B)$ of polynomials that are non-negative on B:

$$\mathrm{W}(B) \coloneqq \{f \in \mathbb{R}[X] \mid f(x) \geq 0 \text{ for all } x \in B\}.$$

Obviously $\mathrm{U}(B) \subseteq \mathrm{W}(B)$. For $f, g \in \mathrm{W}(\mathbb{R}) \setminus \{0\}$, by the fundamental theorem of algebra, one can write (uniquely)

$$f = c\prod_{j\in J}(x-r_j)^{d_j}\prod_{k\in K}(x^2+a_kx+b_k)^{e_k}, \quad g = c'\prod_{j\in J}(x-r_j)^{d'_j}\prod_{k\in K}(x^2+a_kx+b_k)^{e'_k}$$

where $c, c', r_j, a_k, b_k \in \mathbb{R}$ and d_j, d'_j, e_k, e'_k are non-negative integers, and the polynomials $x^2 + a_kx + b_k$ have no real root. Here, J indexes all real roots of f and g, and K indexes all conjugate pairs of imaginary roots of f and g. Since f and g are non-negative on $\mathbb{R}$, all d_j and d'_j are even, and c, c' are positive. Therefore, the *greatest common divisor* of f and g, defined by

$$\gcd(f,g) \coloneqq \prod_{j\in J}(x-r_j)^{\min\{d_j,d_j'\}} \prod_{k\in K}(x^2+a_kx+b_k)^{\min\{e_k,e_k'\}}$$

is also non-negative on $\mathbb{R}$. It follows that the polynomials $\gcd(f,g)$, $f/\gcd(f,g)$ and $g/\gcd(f,g)$ are all in $\mathrm{W}(\mathbb{R})$.

We now give a proof of our first main result, which can be considered as a "strictly positive" version of the Bröcker-Prestel local-global principle. A comparison of our proof with the proof of the original theorem is given in Appendix C.

▶ **Theorem 2.1.** *Let B be a subset of $\mathbb{R}$. Assume that $h_1,\dots,h_n \in \mathbb{R}[X]$ are polynomials that satisfy $\gcd(h_1,\dots,h_n)=1$. If the equation $f_1h_1+\cdots+f_nh_n=0$ has no solution $(f_1,\dots,f_n)$ over $\mathrm{U}(B)$, then there exists a real number $t\in\overline{B}$, such that the values $h_i(t), i=1,\dots,n$, are either all non-negative or all non-positive.*

Proof. The theorem is trivially true if B is empty, hence we suppose $B\neq\emptyset$. Suppose $f_1h_1+\cdots+f_nh_n=0$ has no solution $(f_1,\dots,f_n)$ over $\mathrm{U}(B)$. Consider the following subset of the field $\mathbb{R}(X)$:

$$P_0 \coloneqq \left\{ \frac{g}{G}\cdot\sum_{i=1}^n f_ih_i, \text{ where all } f_i\in\mathrm{U}(B) \text{ and } g,G\in\mathrm{W}(\mathbb{R})\setminus\{0\} \right\}.$$

Since $f_1h_1+\cdots+f_nh_n=0$ has no solution $(f_1,\dots,f_n)$ over $\mathrm{U}(B)$, we have $0\notin P_0$. We claim that $P_0'=P_0\cup\{0\}$ is a pre-semicone of $\mathbb{R}(X)$. Indeed, we verify the three conditions given in Definition 3.4:

- **(i)** $P_0'+P_0'\subseteq P_0'$. It suffices to show $P_0+P_0\subseteq P_0$. Let $c=\frac{g}{G}\cdot\sum_{i=1}^n f_ih_i$, $c'=\frac{g'}{G'}\cdot\sum_{i=1}^n f_i'h_i$ be elements of P_0. Without loss of generality we can suppose $\gcd(g,G)=\gcd(g',G')=1$. Write $d\coloneqq\gcd(g,g')$, $D\coloneqq\gcd(G,G')$, then the polynomials d, $\frac{g}{d}$, $\frac{g'}{d}$, D, $\frac{G}{D}$, $\frac{G'}{D}$ are all elements of $\mathrm{W}(\mathbb{R})\setminus\{0\}$, and $\gcd(\frac{gG'}{dD},\frac{g'G}{dD})=1$. Hence,

$$c+c'=\sum_{i=1}^n\left(f_i\frac{g}{G}+f_i'\frac{g'}{G'}\right)h_i=\frac{dD}{GG'}\sum_{i=1}^n\left(f_i\frac{gG'}{dD}+f_i'\frac{g'G}{dD}\right)h_i \tag{5}$$

  For any $x\in B$, we have $\frac{gG'}{dD}(x)\geq 0$ and $\frac{g'G}{dD}(x)\geq 0$. Since $\gcd(\frac{gG'}{dD},\frac{g'G}{dD})=1$, the two polynomials $\frac{gG'}{dD},\frac{g'G}{dD}$ cannot both vanish at x. Therefore either $\frac{gG'}{dD}(x)>0$ or $\frac{g'G}{dD}(x)>0$. Because $f_i(x)>0$ and $f_i'(x)>0$, it follows that $\left(f_i\frac{gG'}{dD}+f_i'\frac{g'G}{dD}\right)(x)>0$. So $f_i\frac{gG'}{dD}+f_i'\frac{g'G}{dD}\in\mathrm{U}(B)$, and $c+c'\in P_0$.
- **(ii)** $\mathbb{R}(X)^2\cdot P_0'\subseteq P_0'$. This is obvious since $\mathbb{R}[X]^2\cdot\mathrm{W}(\mathbb{R})\subseteq\mathrm{W}(\mathbb{R})$.
- **(iii)** $P_0'\cap -P_0'=\{0\}$. It suffices to show $P_0\cap -P_0=\emptyset$. On the contrary suppose $c\in P_0\cap -P_0$, then $0=c+(-c)\in P_0+P_0\subseteq P_0$, a contradiction.

By Lemma 3.5, P_0' can be extended to some P such that either P or $-P$ is a semicone of the field $\mathbb{R}(X)$. Without loss of generality suppose $P\supseteq P_0'$ is a semicone, otherwise we can replace all h_i by $-h_i$. Since the field $\mathbb{R}(X)$ has no archimedean ordering [25, Example 1.1.4(2)], the *semiordering* corresponding to P must be non-archimedean (otherwise by Lemma 3.6 it must be an archimedean ordering). Consider the subfield $\mathbb{R}$ of $\mathbb{R}(X)$, by Lemma 3.8 the valuation ring $A_{\mathbb{R}}^P$ (as defined in (4)) corresponds to some valuation v of $\mathbb{R}(X)$. Since P is non-archimedean, there exists some $a\in\mathbb{R}(X)$ such that $a-r\in P$ for all $r\in\mathbb{R}$, hence $A_{\mathbb{R}}^P\neq\mathbb{R}(X)$. Also, Lemma 3.9 shows that the residue field F_v admits a semiordering $(P\cap A_v)/M_v$. By Lemma 3.3, F_v is formally real. Therefore, v is a non-trivial real place of $\mathbb{R}(X)$, and from the definition of $A_{\mathbb{R}}^P$ we have $\mathbb{R}\subseteq A_{\mathbb{R}}^P=A_v$.

Using the classification of real places of $\mathbb{R}(X)$ given in Proposition 3.7, consider the following three cases. Since F_v is isomorphic to $\mathbb{R}$, the semiordering $(P \cap A_v)/M_v$ corresponds to the only ordering on $\mathbb{R}$.

1. The real place v is equivalent to a place v_t for some $t \in \overline{B} \subseteq \mathbb{R}$. In this case $\mathbb{R}[X] \subseteq A_v$. We show that $h_i(t) \geq 0$ for all i. By symmetry it suffices to show $h_1(t) \geq 0$. For every $\varepsilon \in \mathbb{R}_{>0}$, we have $\varepsilon \in \mathrm{U}(B)$, so $h_1 + \varepsilon(h_2 + \cdots + h_n) \in P_0 \subseteq P$. Since $h_1 + \varepsilon(h_2 + \cdots + h_n) \in \mathbb{R}[X] \subseteq A_v$, we have $h_1 + \varepsilon(h_2 + \cdots + h_n) \in P \cap A_v$, which gives
$$h_1 + \varepsilon(h_2 + \cdots + h_n) + M_v \in (P \cap A_v)/M_v. \tag{6}$$
Since the residue field $\mathbb{R}(X)_v$ is isomorphic to $\mathbb{R}$ by the natural homomorphism $y + M_v \mapsto y(t)$, Equation (6) yields
$$h_1(t) + \varepsilon(h_2(t) + \cdots + h_n(t)) \geq 0.$$
Since this is true for all $\varepsilon > 0$, we conclude that $h_1(t) \geq 0$ and thus $h_i(t) \geq 0$ for all i.
2. The real place v is equivalent to a place v_t for some $t \in \mathbb{R} \setminus \overline{B}$. There exists a polynomial $H_B \in \mathbb{R}[X]$, such that $H_B(x) > 0$ for all $x \in B$ but $H_B(t) < 0$. Indeed, since $t \notin \overline{B}$, there exists an interval $(t-\delta, t+\delta)$ disjoint from B; it then suffices to take $H_B \coloneqq (X-t)^2 - \delta^2$. As in the previous case, we have $h_1(t) \geq 0$. Furthermore, since $H_B \in \mathrm{U}(B)$ by its definition, we have $H_B h_1 + \varepsilon(h_2 + \cdots + h_n) \in P_0 \subseteq P$ for all $\varepsilon \in \mathbb{R}_{>0}$. This yields $(H_B h_1)(t) \geq 0$. However, we have $H_B(t) < 0$ by its definition. This together with $h_1(t) \geq 0$ yields $h_1(t) = 0$. By symmetry we can prove $h_i(t) = 0$ for all i, this contradicts the condition $\gcd(h_1, \ldots, h_n) = 1$.
3. The real place v is equivalent to the place v_∞. We divide $\{h_1, \ldots, h_n\}$ into two parts according to the parity of its degree. Without loss of generality, suppose $h_1, \ldots, h_k$ have even degree, and $h_{k+1}, \ldots, h_n$ have odd degree.
Define the *leading coefficient* of a polynomial as the coefficient of its highest degree monomial. First we claim that the leading coefficients of $h_1, \ldots, h_k$ are all positive. By symmetry, we only prove positivity of the leading coefficients of h_1.
Let $m = \max\{\deg h_1, \ldots, \deg h_n\} + 1$. Since $(X^2+1)^m \in \mathrm{U}(B)$ and $X^{\deg h_1} \in \mathrm{W}(\mathbb{R})$, we have
$$\frac{h_1}{X^{\deg h_1}} + \frac{(X^2+1)^m}{(X^2+1)^{2m}}(h_2 + \cdots + h_n) + M_v \in (P \cap A_v)/M_v. \tag{7}$$
Since the residue field $\mathbb{R}(X)_v$ is isomorphic to $\mathbb{R}$ by the natural homomorphism $y + M_{v_t} \mapsto \lim_{t\to\infty} y(t)$, Equation (7) shows that the leading coefficient of h_1 is positive. Therefore by symmetry, the leading coefficient of h_i is positive for all $1 \leq i \leq k$.
We then separate four cases.
   a. If B is bounded, that is, $B \subset (a, b)$ for some $a, b \in \mathbb{R}$. Let $s > \max\{|a|, |b|\}$, then $X + s \in \mathrm{U}(B)$. Since $\deg h_n$ is odd, we have $X^{\deg h_n + 1} \in \mathrm{W}(\mathbb{R})$. Therefore,
   $$\frac{(X+s)h_n}{X^{\deg h_n + 1}} + \frac{(X^2+1)^m}{(X^2+1)^{2m}}(h_1 + \cdots + h_{n-1}) + M_v \in (P \cap A_v)/M_v. \tag{8}$$
   This shows that the leading coefficient of h_n is positive.
   However, we also have $-X + s \in \mathrm{U}(B)$, so we can replace $(X+s)$ with $(-X+s)$ in Equation (8). This shows that the leading coefficient of h_n is negative. Therefore h_n does not exist, so all $h_1, \ldots, h_n$ must have even degree. But then $(X+1-a)(b+1-X) \in \mathrm{U}(B)$, so
   $$\frac{(X+1-a)(b+1-X)h_1}{X^{\deg h_1 + 2}} + \frac{(X^2+1)^m}{(X^2+1)^{2m}}(h_2 + \cdots + h_n) + M_v \in (P \cap A_v)/M_v. \tag{9}$$

This shows that the leading coefficient of h_1 is negative, a contradiction.

b. If $B \subset (a, +\infty)$ for some $a \in \mathbb{R}$, and B contains arbitrarily large positive reals, that is, $B \cap (b, +\infty) \neq \emptyset$ for all $b \in \mathbb{R}$. Then $X + 1 - a \in \mathrm{U}(B)$, so

$$\frac{(X+1-a)h_n}{X^{\deg h_n+1}} + \frac{(X^2+1)^m}{(X^2+1)^{2m}}(h_1 + \cdots + h_{n-1}) + M_v \in (P \cap A_v)/M_v. \tag{10}$$

This shows that the leading coefficient of h_n is positive. By symmetry, the leading coefficients of $h_{k+1}, \ldots, h_n$ are all positive. Therefore, for large enough $t \in B$, $h_1(t), \ldots, h_n(t)$ are all positive.

c. If $B \subset (-\infty, a)$ for some $a \in \mathbb{R}$, and B contains arbitrarily small reals, that is, $B \cap (-\infty, b) \neq \emptyset$ for all $b \in \mathbb{R}$. Then $a + 1 - X \in \mathrm{U}(B)$, so

$$\frac{(a+1-X)h_n}{X^{\deg h_n+1}} + \frac{(X^2+1)^m}{(X^2+1)^{2m}}(h_1 + \cdots + h_{n-1}) + M_v \in (P \cap A_v)/M_v. \tag{11}$$

This shows that the leading coefficient of h_n is negative. By symmetry, the leading coefficients of $h_{k+1}, \ldots, h_n$ are all negative. Therefore, for small enough $0 > t \in B$, $h_1(t), \ldots, h_n(t)$ are all positive.

d. If B contains arbitrarily large and arbitrarily small reals. We claim that the leading coefficients of $h_{k+1}, \ldots, h_n$ all have the same sign. Suppose on the contrary that they have different signs, denote by a_i the leading coefficient of h_i, so $h_i = a_i X^{\deg h_i} + H_i$ for some polynomial H_i of degree at most $\deg h_i - 1$. Then there exist *strictly positive* reals $r_{k+1}, \ldots, r_n$ such that $r_{k+1}a_{k+1} + \cdots + r_n a_n = 0$. Then, for any $s \in \mathbb{R}$, we have $X^2 - 2sX + s^2 + 1 \in \mathrm{U}(B)$, so

$$\begin{aligned}\frac{(X^2+1)^m}{(X^2+1)^{2m}}(h_1 + \cdots + h_k) + \frac{r_{k+1}(X^2 - 2sX + s^2 + 1)}{X^{\deg h_{k+1}+1}}h_{k+1} \\ + \frac{r_{k+2}}{X^{\deg h_{k+2}-1}}h_{k+2} + \cdots + \frac{r_n}{X^{\deg h_n - 1}}h_n + M_v \in (P \cap A_v)/M_v.\end{aligned} \tag{12}$$

The limit of the left hand side when X tends to infinity is equal to

$$\begin{aligned} g(s) &:= \lim_{X\to\infty}\left(\frac{r_{k+1}(X^2 - 2sX)}{X^{\deg h_{k+1}+1}}h_{k+1}(X) + \sum_{j=k+2}^{n}\frac{r_j}{X^{\deg h_j - 1}}h_j(X)\right) \\ &= -2sa_{k+1}r_{k+1} + \lim_{X\to\infty}\left(\sum_{j=k+1}^{n}\frac{r_j}{X^{\deg h_j - 1}}H_j(X)\right)\end{aligned}$$

because $r_{k+1}a_{k+1} + \cdots + r_n a_n = 0$. According to whether $a_{k+1}r_{k+1}$ is positive or negative, we can take a positive or negative s with large enough absolute value, so that the value of $g(s)$ is negative. This contradicts Equation (12), which shows that the limit of the left hand side when $X \to \infty$ is positive.

We therefore conclude that the leading coefficients of $h_{k+1}, \ldots, h_n$ all have the same sign. If they are positive, then for large enough $t \in B$, $h_1(t), \ldots, h_n(t)$ are all positive. If they are negative, then for small enough $t \in B$, $h_1(t), \ldots, h_n(t)$ are all positive.

To sum up, in all possible cases, we have $t \in \overline{B}$ with $h_i(t) \geq 0$ for all i. If $-P$ is a semicone instead of P, analogously we can find $t \in \overline{B}$ such that $h_i(t) \leq 0$ for all i. ◀

5 Local-global principle over $\mathbb{N}[X]$

In this section we prove Theorem 2.2. Omitted proofs are given in Appendix A. The key to bridging the difference between the semirings $\mathrm{U}(B)$ and $\mathbb{N}[X]$ is Pólya's Theorem:

▶ **Lemma 5.1** (Pólya's Theorem [14, Theorem 56]). *If a homogeneous polynomial $f \in \mathbb{R}[X_1, \ldots, X_n]$ is strictly positive for all $(X_1, \ldots, X_n)$ on $(\mathbb{R}_{\geq 0})^n \setminus \{0\}$, then there exists $p \in \mathbb{N}$ such that $(X_1 + \cdots + X_n)^p \cdot f \in \mathbb{R}_{\geq 0}[X_1, \ldots, X_n]$.*

The following proposition reduces Theorem 2.2 to real polynomials.

▶ **Proposition 5.2.** *Given $h_1, \ldots, h_n \in \mathbb{Z}[X]$. The equation $f_1 h_1 + \cdots + f_n h_n = 0$ has a solution $(f_1, \ldots, f_n)$ over $\mathbb{N}[X] \setminus \{0\}$ if and only if it has a solution over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$.*

The next proposition further reduces it to $\mathrm{U}(\mathbb{R}_{>0})$. The key to its proof is Lemma 5.1.

▶ **Proposition 5.3.** *Given $h_1, \ldots, h_n \in \mathbb{Z}[X]$. The equation $f_1 h_1 + \cdots + f_n h_n = 0$ has a solution $(f_1, \ldots, f_n)$ over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$ if and only if it has a solution over $\mathrm{U}(\mathbb{R}_{>0})$.*

This justifies the need for a "strictly positive" version of the Bröcker-Prestel principle, since Proposition 5.3 no longer holds if we replace $\mathrm{U}(\mathbb{R}_{>0})$ with $\mathrm{W}(\mathbb{R}_{>0}) \setminus \{0\}$ (see Remark A.1).

We now prove the local-global principle for homogeneous linear equations over $\mathbb{N}[X]$.

▶ **Theorem 2.2.** *Given polynomials $h_1, \ldots, h_n \in \mathbb{Z}[X]$ with $\gcd(h_1, \ldots, h_n) = 1$. If the equation $f_1 h_1 + \cdots + f_n h_n = 0$ has no solution $(f_1, \ldots, f_n)$ over $\mathbb{N}[X] \setminus \{0\}$, then there exists $t \in \mathbb{R}_{\geq 0}$, such that the values $h_i(t), i = 1, \ldots, n$ are either all non-negative or all non-positive.*

Proof. Suppose the equation $f_1 h_1 + \cdots + f_n h_n = 0$ has no solution $(f_1, \ldots, f_n)$ over $\mathbb{N}[X] \setminus \{0\}$. By Proposition 5.2 and 5.3, it has no solution over $\mathrm{U}(\mathbb{R}_{>0})$. Hence, by Theorem 2.1, there exists a real number $t \in \overline{\mathbb{R}_{>0}} = \mathbb{R}_{\geq 0}$ such that $h_i(t)$ are all non-negative or all non-positive. ◀

6 Decidability

In this section we show our main decidability result.

▶ **Theorem 2.3.** *Given as input $h_1, \ldots, h_n \in \mathbb{Z}[X]$. It is decidable in polynomial time whether the equation $f_1 h_1 + \cdots + f_n h_n = 0$ has solutions $f_1, \ldots, f_n$ over $\mathbb{N}[X] \setminus \{0\}$.*

Proof. (A summary of the algorithm constructed in this proof is given in Appendix B.)

By the homogeneity of the linear equation, we can divide $h_1, \ldots, h_n$ by their greatest common divisor and suppose $\gcd(h_1, \ldots, h_n) = 1$. Computing the greatest common divisor can be done in polynomial time using the Euclidean algorithm.

We then show that we can simplify the equation so that $h_1, \ldots, h_n$ satisfy

$$h_i(0) > 0, h_j(0) < 0, \quad \text{for some } i, j. \tag{13}$$

Suppose this is not already the case, that $h_i(0) \geq 0$ for all i or $h_i(0) \leq 0$ for all i. Without loss of generality suppose $h_i(0) \geq 0$ for all i. We write $h_1(0) = 0, \ldots, h_k(0) = 0, h_{k+1}(0) > 0, \ldots, h_n(0) > 0$. Then $X \mid h_i$ for $i = 1, \ldots, k$.

If $k = 0$, that is $h_i(0) > 0$ for all i, then $f_1 h_1 + \cdots + f_n h_n = 0$ has no solution over $\mathbb{N}[X] \setminus \{0\}$. Indeed, suppose on the contrary that $(f_1, \ldots, f_n)$ is such a solution. Dividing all f_i by a suitable power of X we can suppose $f_s(0) \neq 0$ for some s. Then $f_i(0) \geq 0$ for all i while $f_s(0) > 0$, which yields $f_1(0)h_1(0) + \cdots + f_n(0)h_n(0) > 0$, a contradiction.

If $k \geq 1$, we show that the equation

$$f_1 h_1 + \cdots + f_n h_n = 0 \tag{14}$$

has a solution over $\mathbb{N}[X] \setminus \{0\}$ if and only if the equation

$$f_1 \cdot \frac{h_1}{X} + \cdots + f_k \cdot \frac{h_k}{X} + f_{k+1} h_{k+1} + \cdots + f_n h_n = 0 \tag{15}$$

has a solution over $\mathbb{N}[X]\setminus\{0\}$. Let $(f_1,\dots,f_n)$ be a solution over $\mathbb{N}[X]\setminus\{0\}$ of Equation (14), then $f_1(0)h_1(0)+\cdots+f_n(0)h_n(0)=0$. Since $h_i(0)=0$ for all $i=1,\dots,k$, $h_i(0)>0$ for $i=k+1,\dots,n$ and $f_i(0)\geq 0$ for $i=1,\dots,n$, we must have $f_{k+1}(0)=0,\dots,f_n(0)=0$. That is, $X\mid f_{k+1},\dots,X\mid f_n$. Therefore $(f_1,\dots,f_k,f_{k+1}/X,\dots,f_n/X)$ is a solution over $\mathbb{N}[X]\setminus\{0\}$ of Equation (15). This shows that we can divide $h_1,\dots,h_k$ by X without changing the existence of solutions of Equation (14). Repeating this division process, one eventually terminates by obtaining h_i such that either: $h_i(0)$ are all strictly positive or all strictly negative, in which case Equation (14) has no solution over $\mathbb{N}[X]\setminus\{0\}$; or $h_i(0)>0$ and $h_j(0)<0$ for some i,j, in which case we have achieved the desired simplification to Condition (13). This procedure is repeated at most $\deg h_1+\cdots+\deg h_n$ times, and therefore terminates in polynomial time.

Supposing Condition (13), we claim that $f_1h_1+\cdots+f_nh_n=0$ has no solution over $\mathbb{N}[X]\setminus\{0\}$ if and only if there exists $t\in\mathbb{R}_{\geq 0}$ such that $h_i(t)$ are all non-positive or all non-negative. The first implication is given by Theorem 2.2. Conversely, suppose $h_i(t)$ are all non-positive or all non-negative. Without loss of generality suppose $h_i(t)\geq 0$ for all i. By Condition (13), we have $t\neq 0$. Suppose on the contrary that $(f_1,\dots,f_n)$ is a solution over $\mathbb{N}[X]\setminus\{0\}$, then $f_i(t)>0$ for all i since $t>0$. Since $\gcd(h_1,\dots,h_n)=1$, at least one of $h_i(t)$ must be non-zero. Since $h_i(t)\geq 0$ for all i, we have $f_1(t)h_1(t)+\cdots+f_n(t)h_n(t)>0$, a contradiction.

Thus, it suffices to decide whether there exists $t\geq 0$ such that $h_i(t)$ are all non-positive or all non-negative. This can be expressed in the existential theory of the reals:

$$\exists X\ (X\geq 0\wedge h_1(X)\geq 0\wedge\cdots\wedge h_n(X)\geq 0)\vee(X\geq 0\wedge h_1(X)\leq 0\wedge\cdots\wedge h_n(X)\leq 0)\,. \quad (16)$$

Deciding the existential theory of the reals *in one variable* can be done in polynomial time with respect to the total bit length used to encode the sentence, due to a classic result by Collins[2] [8]. Therefore, one can decide the correctness of the sentence (16) in polynomial time. Combining all the steps, we conclude that the total complexity is in PTIME. ◀

7 Application to wreath product

In this section we show the following result on wreath products.

▶ **Theorem 2.4.** *Given a finite set of elements $\mathcal{G}=\{(y_1,b_1),\dots,(y_n,b_n)\}$ in $\mathbb{Z}\wr\mathbb{Z}$, where $b_i=\pm 1$ for all i. The following are decidable:*

1. *(Group Problem) whether the semigroup $\langle\mathcal{G}\rangle$ generated by $\mathcal{G}$ is a group.*
2. *(Identity Problem) whether the neutral element I is in the semigroup $\langle\mathcal{G}\rangle$.*

Let φ be the isomorphism defined in (3). Fix a finite set of elements $\mathcal{G}$ as in Theorem 2.4. For $i=1,\dots,n$, denote by $H_i\in\mathbb{Z}[X,X^{-1}]$ the Laurent polynomial in the upper-right entry of the image of $\varphi((y_i,b_i))$. Write $\mathcal{G}=\mathcal{G}_+\cup\mathcal{G}_-$ where $\mathcal{G}_+:=\{(y_i,b_i)\in\mathcal{G}\mid b_i=1\}$ and $\mathcal{G}_-:=\{(y_j,b_j)\in\mathcal{G}\mid b_j=-1\}$. Let $\varphi(\mathcal{G}),\varphi(\mathcal{G}_+),\varphi(\mathcal{G}_-)$ be the set of matrices that are images under φ of elements in $\mathcal{G},\mathcal{G}_+,\mathcal{G}_-$. Define the sets of indices

$$I:=\{i\mid b_i=1\},\quad J:=\{i\mid b_i=-1\}.$$

[2] The algorithm by Collins [8] has complexity $L^3(nd)^{2^{O(K)}}$, where L is the total coefficient bit length, n the number of polynomials, d the total degree of the polynomials, and K the number of variables. In the one variable case, $K=1$, the algorithm takes polynomial time with respect to the total bit length.

For simplicity, we write $A_i, i \in I$ for the matrices in $\varphi(\mathcal{G}_+)$, and $B_j, j \in J$ the matrices in $\varphi(\mathcal{G}_-)$. For every tuple $(i,j) \in I \times J$, define the Laurent polynomial

$$h_{ij} \coloneqq X^{-1}H_i + H_j \in \mathbb{Z}[X, X^{-1}]. \tag{17}$$

This is the upper-right entry of the matrix A_iB_j.

For a subset $S \subseteq I \times J$, denote by $\pi_I(S)$ its projection onto the I coordinates, that is, $\pi_I(S) \coloneqq \{i \in I \mid \exists j \in J, (i,j) \in S\}$. Define $\pi_J(S)$ likewise. The key to proving the partial decidability of the Group Problem in $\mathbb{Z} \wr \mathbb{Z}$ is the following proposition that relates sub-semigroups of $\mathbb{Z} \wr \mathbb{Z}$ to equations over $\mathbb{N}[X] \setminus \{0\}$.

▶ **Proposition 7.1.** *Given a set $\mathcal{G} = \mathcal{G}_+ \cup \mathcal{G}_-$ of generators defined as above. Let $h_{ij} \in \mathbb{Z}[X, X^{-1}]$ be the polynomials defined in* (17)*. The semigroup $\langle \mathcal{G} \rangle$ is a group if and only if there exists a set $S \subseteq I \times J$ satisfying $\pi_I(S) = I, \pi_J(S) = J$, such that the equation $\sum_{(i,j)\in S} f_{ij}h_{ij} = 0$ has a solution $(f_{ij})_{(i,j)\in S}$ over $\mathbb{N}[X] \setminus \{0\}$.*

Proof. For a word w in the alphabet $\varphi(\mathcal{G})$, define its *product* $\pi(w)$ to be the matrix obtained by multiplying all the matrices in w consecutively. Denote by $|w|_+$ (respectively, $|w|_-$) the number of letters in w belonging in $\varphi(\mathcal{G}_+)$ (respectively, $\varphi(\mathcal{G}_-)$). Define the *height* of the word w to be $h(w) \coloneqq |w|_+ - |w|_-$, then we have $\pi(w) = \begin{pmatrix} 1 & * \\ 0 & X^{h(w)} \end{pmatrix}$, where $*$ is some element in $\mathbb{Z}[X, X^{-1}]$.

For a finite alphabet $\mathcal{A}$, denote by $\mathcal{A}^+$ the set of non-empty words over $\mathcal{A}$. We claim that for any non-empty word $w \in \varphi(\mathcal{G})^+$ such that $h(w) = 0$, the upper right entry of $\pi(w)$ can be written as a sum $\sum_{(i,j)\in I\times J} f_{ij}h_{ij}$, where f_{ij} are elements in $\mathbb{N}[X, X^{-1}]$. We prove this by induction on the length of the word w. For the sake of simplicity, denote $U(\pi(w))$ the upper right entry of $\pi(w)$.

If w has length at most two, then it must be of the form A_iB_j or B_jA_i, and the claim is easy to verify. Suppose the claim is true for all words w of length less then $\ell > 2$. We prove the claim for words w of length ℓ. Distinguish the following two cases.

1. **The word w is of the form $A_iw'B_j$ or $B_jw'A_i$ for some $i \in I, j \in J, w' \in \varphi(\mathcal{G})^+$.** Since w' has length at most $\ell - 2$ and is of height 0, by induction hypothesis, $\pi(w') = \begin{pmatrix} 1 & r \\ 0 & 1 \end{pmatrix}$, with r a linear combination of h_{ij} with coefficients in $\mathbb{N}[X, X^{-1}]$. If $w = A_iw'B_j$, then

$$\pi(w) = \begin{pmatrix} 1 & H_i \\ 0 & X \end{pmatrix}\begin{pmatrix} 1 & r \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & H_j \\ 0 & X^{-1} \end{pmatrix} = \begin{pmatrix} 1 & X^{-1}r + (X^{-1}H_i + H_j) \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & X^{-1}r + h_{ij} \\ 0 & 1 \end{pmatrix}.$$

   So $U(\pi(w)) = X^{-1}r + h_{ij}$ can also be written as a linear combination of $h_{ij}, i \in I, j \in J$ with coefficients in $\mathbb{N}[X, X^{-1}]$. If $w = B_jw'A_i$, then

$$\pi(w) = \begin{pmatrix} 1 & H_j \\ 0 & X^{-1} \end{pmatrix}\begin{pmatrix} 1 & r \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & H_i \\ 0 & X \end{pmatrix} = \begin{pmatrix} 1 & Xr + H_i + XH_j \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & X(r + h_{ij}) \\ 0 & 1 \end{pmatrix}.$$

   So $U(\pi(w)) = X(r + h_{ij})$ can also be written as a linear combination of $h_{ij}, i \in I, j \in J$ with coefficients in $\mathbb{N}[X, X^{-1}]$.
2. **The word w is of the form $A_iw'A_{i'}$ or $B_jw'B_{j'}$ for some $i, i' \in I$ or $j, j' \in J$.** First suppose $w = A_iw'A_{i'}$. Since $h(A_i) = 1 > 0$ and $h(A_iw') = -1 < 0$, there must exist a strict prefix v of w with height zero. This is because by reading the word w letter by letter, this height of consecutive prefixes differs by at most one. We have $w = vv'$ with $h(v) = h(v') = 0$ where v, v' are non-empty words. By induction hypothesis,

$U(\pi(v)), U(\pi(v'))$ can be written as a linear combination of h_{ij} with coefficients in $\mathbb{N}[X, X^{-1}]$. Therefore $U(\pi(w)) = U(\pi(v)) + U(\pi(v'))$ also satisfies this claim. The case where $w = B_j w' B_{j'}$ is completely analogous.

Combining the two cases concludes the induction. It is easy to see from the induction process that if the letter A_i appears in w, then the coefficient of the term h_{ij} in the linear combination is not zero for some $j \in J$. This is because at some point we have replaced r with either $X^{-1}r + h_{ij}$ or $X(r + h_{ij})$. Similarly, if the letter B_j appears in w, then the coefficient of the term h_{ij} in the linear combination is non-zero for some $i \in I$.

If the semigroup $\langle \mathcal{G} \rangle$ is a group, then there exists a word v in the alphabet $\mathcal{G}$ using all letters in $\mathcal{G}$, whose corresponding product is the neutral element. Taking the image under φ yields a word $w = \varphi(v)$ in the alphabet $\varphi(\mathcal{G})$ such that $h(w) = 0$ and $U(\pi(w)) = 0$. The claim above and the discussion following it show that there exist Laurent polynomials $f_{ij} \in \mathbb{N}[X, X^{-1}]$ such that $\sum_{(i,j)\in I\times J} f_{ij}h_{ij} = 0$. Furthermore, all letters $A_i, i \in I$ and $B_j, j \in J$ appear in w, so for every i, the coefficient f_{ij} in the linear combination is not zero for some $j \in J$; and for every j, the coefficient f_{ij} is not zero for some $i \in I$. Let $S \coloneqq \{(i,j) \in I \times J \mid f_{ij} \neq 0\}$, then $\sum_{(i,j)\in S} f_{ij}h_{ij} = 0$, and $\pi_I(S) = I, \pi_J(S) = J$. By the homogeneity of the equation $\sum_{(i,j)\in S} f_{ij}h_{ij} = 0$, one can multiply all f_{ij} by the monomial X^n for a sufficiently large n, and suppose $f_{ij} \in \mathbb{N}[X] \setminus \{0\}$ instead of $\mathbb{N}[X, X^{-1}] \setminus \{0\}$. This completes the proof of the first direction of implication in Proposition 7.1.

For the other direction of implication, suppose there exists a set $S \subseteq I \times J$ satisfying $\pi_I(S) = I, \pi_J(S) = J$, such that the equation $\sum_{(i,j)\in S} f_{ij}h_{ij} = 0$ has a solution $(f_{ij})_{(i,j)\in S}$ over $\mathbb{N}[X] \setminus \{0\}$. By the homogeneity of the equation, suppose that there is a tuple $(u, v) \in S$ such that $X \nmid f_{uv}$. Let $(y, z) \in S$ be a tuple such that $\deg f_{yz} \geq \deg f_{ij}$ for all $(i, j) \in S$.

Denote by $\mathbb{N}_{>0}[X]$ the set of polynomials of the form $\sum_{i=0}^{d} a_i X^i$, where $d \geq 0$ and $a_i > 0$ for all i. By multiplying all f_{ij} by the polynomial $(1 + X)^m$ for a sufficiently large m, we can suppose that $f_{uv} \in \mathbb{N}_{>0}[X]$, $X^{-v_0(f_{yz})} f_{yz} \in \mathbb{N}_{>0}[X]$, and $\deg f_{uv} \geq v_0(f_{yz})$. Indeed, we can take any $m \geq \max\{\deg f_{uv}, \deg X^{-v_0(f_{yz})} f_{yz}, v_0(f_{yz})\}$. Additionally, the condition that $\deg f_{yz} \geq \deg f_{ij}$ for all $(i, j) \in S$ is still satisfied after this multiplication.

We now construct a word $w \in \varphi(\mathcal{G})^+$ that uses every letter in $\varphi(\mathcal{G})$, such that $h(\pi(w)) = 0$, $U(\pi(w)) = \sum_{(i,j)\in S} f_{ij}h_{ij} = 0$. We start with the word

$$w_0 \coloneqq A_u^{\deg f_{uv}} A_y^{\deg f_{yz} - \deg f_{uv}} B_z^{\deg f_{yz} - \deg f_{uv}} B_v^{\deg f_{uv}},$$

which has height 0, and whose product has upper-right entry

$$U(\pi(w_0)) = h_{uv} \cdot \sum_{i=0}^{\deg f_{uv}-1} X^i + h_{yz} \cdot \sum_{i=\deg f_{uv}}^{\deg f_{yz}-1} X^i.$$

Since $f_{uv} \in \mathbb{N}_{>0}[X]$, $X^{-v_0(f_{yz})} f_{yz} \in \mathbb{N}_{>0}[X]$, and $\deg f_{uv} \geq v_0(f_{yz})$, the polynomials $\hat{f}_{uv} \coloneqq f_{uv} - \sum_{i=0}^{\deg f_{uv}-1} X^i$, $\hat{f}_{yz} \coloneqq f_{yz} - \sum_{i=\deg f_{uv}}^{\deg f_{yz}-1} X^i$ are still polynomials in $\mathbb{N}[X] \setminus \{0\}$. For $(i, j) \in S$, define

$$\hat{f}_{ij} \coloneqq \begin{cases} \hat{f}_{uv} & (i,j) = (u,v) \\ \hat{f}_{yz} & (i,j) = (y,z) \\ f_{ij} & \text{otherwise.} \end{cases}$$

These are elements in $\mathbb{N}[X]\setminus\{0\}$ and satisfy $U(\pi(w_0)) + \sum_{(i,j)\in S} \hat{f}_{ij}h_{ij} = \sum_{(i,j)\in S} f_{ij}h_{ij} = 0$.

We then gradually insert "loops" of the form A_iB_j into the word w_0. This insertion does not change the height of the word, but it adds a multiple of h_{ij} to the upper-right entry of the product. Indeed, if $h(vv') = 0$, then we have $h(vA_iB_jv') = 0$ and $U(\pi(vA_iB_jv')) =$

$U(\pi(vv')) + X^{h(v)}h_{ij}$. Note that the initial word w_0 has suffixes of all heights from 0 to $\deg f_{yz}$. For each $k = 0, \dots, \deg f_{yz}$ and each $(i,j) \in S$, after a suffix of height k, we insert $\mathrm{Coef}_{X^k}(\hat{f}_{ij})$ times the "loop" A_iB_j, where $\mathrm{Coef}_{X^k}(\hat{f}_{ij})$ is the coefficient of the monomial X^k in the polynomial $\hat{f}_{ij}$. The upper-right entry of the product after all these insertions will be

$$U(\pi(w_0)) + \sum_{k=0}^{\deg f_{yz}} \sum_{(i,j)\in S} \mathrm{Coef}_{X^k}(\hat{f}_{ij}) X^k \cdot h_{ij} = U(\pi(w_0)) + \sum_{(i,j)\in S} \hat{f}_{ij} h_{ij} = 0,$$

because $\deg f_{yz} \geq \deg f_{ij}$ for all $(i,j) \in I \times J$. See Figure 1 for an example of this construction.

We have thus constructed a word $w \in \varphi(\mathcal{G})^+$ such that $h(\pi(w)) = 0$, $U(\pi(w)) = 0$. Note that we have inserted at least one loop A_iB_j for each $(i,j) \in S$. Since $\pi_I(S) = I, \pi_J(S) = J$, the word w contains every letter $A_i, i \in I$ and $B_j, j \in J$. Because $\pi(w)$ is the neutral element, the inverse of every letter in w can be written as a product of matrices in $\varphi(\mathcal{G})$. Indeed, if $w = vXv'$ then $X^{-1} = \pi(v'v)$. Thus the inverse of every element of $\varphi(\mathcal{G})$ is in $\langle\varphi(\mathcal{G})\rangle$. We conclude that $\langle\varphi(\mathcal{G})\rangle$, and thus $\langle\mathcal{G}\rangle$, is a group. ◀

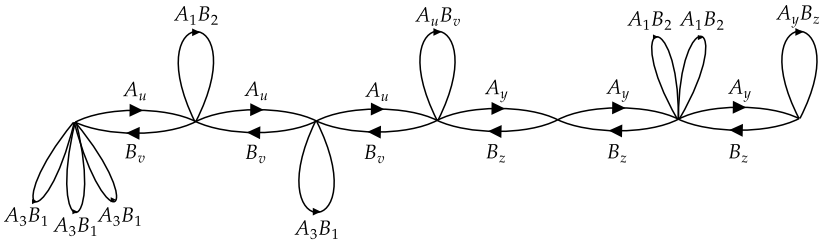


Figure 1 Example of a word constructed in the proof of Proposition 7.1. Here, $S = \{(u,v),(y,z),(1,2),(3,1)\}$, and $f_{uv} = 1 + X + X^2 + X^3$, $f_{yz} = X^3 + X^4 + X^5 + X^6$, $f_{12} = X + 2X^5$, $f_{31} = 3 + X^2$. The constructed word is $A_u(A_1B_2)A_uA_u(A_uB_v)A_yA_y(A_1B_2)(A_1B_2)A_y(A_yB_z)B_zB_zB_zB_v(A_3B_1)B_vB_v(A_3B_1)(A_3B_1)(A_3B_1)$.

We have thus established the link between the Group Problem in $\mathbb{Z} \wr \mathbb{Z}$ and homogeneous linear equations over $\mathbb{N}[X]$. Theorem 2.4 follows from Proposition 7.1 and the decidability result of Theorem 2.3. Its proof is given in Appendix A.

References

1 Emil Artin. Über die zerlegung definiter funktionen in quadrate. In *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*, volume 5, pages 100–115. Springer, 1927.

2 William Ross Ashby. *Automata Studies: Annals of Mathematics Studies. Number 34*. Princeton University Press, 1956.

3 François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

4 Paul C. Bell, Mika Hirvensalo, and Igor Potapov. The identity problem for matrix semigroups in $SL_2(Z)$ is NP-complete. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 187–206. SIAM, 2017.

5 Paul C. Bell and Igor Potapov. On the undecidability of the identity correspondence problem and its applications for word and matrix semigroups. *International Journal of Foundations of Computer Science*, 21(06):963–978, 2010.

6 Jan A. Bergstra and Jan Willem Klop. The algebra of recursively defined processes and the algebra of regular processes. In *International Colloquium on Automata, Languages, and Programming*, pages 82–94. Springer, 1984.

7 Michaël Cadilhac, Dmitry Chistikov, and Georg Zetzsche. Rational subsets of Baumslag-Solitar groups. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 116:1–116:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.116.

8 George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata theory and formal languages*, pages 134–183. Springer, 1975.

9 Louis Dale. Monic and monic free ideals in a polynomial semiring. *Proceedings of the American Mathematical Society*, 56(1):45–50, 1976.

10 Ruiwen Dong. On the identity problem and the group problem for subsemigroups of unipotent matrix groups, 2022. Submitted. doi:10.48550/arXiv.2208.02164.

11 Samuel Eilenberg. *Automata, languages, and machines*. Academic press, 1974.

12 Jonathan S. Golan. *Semirings and affine equations over them: theory and applications*, volume 556. Springer Science & Business Media, 2013.

13 Rostislav I. Grigorchuk and Andrzej Żuk. The lamplighter group as a group generated by a 2-state automaton, and its spectrum. *Geometriae Dedicata*, 87(1):209–244, 2001.

14 Godfrey H. Hardy, John E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, Cambridge, 1952.

15 Äbdelilah Kandri-Rody and Deepak Kapur. Computing a Gröbner basis of a polynomial ideal over a euclidean domain. *Journal of symbolic computation*, 6(1):37–57, 1988.

16 Ravindran Kannan. Solving systems of linear equations over polynomials. *Theoretical Computer Science*, 39:69–88, 1985.

17 Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965.

18 Markus Lohrey, Benjamin Steinberg, and Georg Zetzsche. Rational subsets and submonoids of wreath products. *Information and Computation*, 243:191–204, 2015.

19 Wilhelm Magnus. On a theorem of Marshall Hall. *Annals of Mathematics*, pages 764–768, 1939.

20 A. Markov. On certain insoluble problems concerning matrices. *Doklady Akad. Nauk SSSR*, 57(6):539–542, 1947.

21 K. A. Mikhailova. The occurrence problem for direct products of groups. *Matematicheskii Sbornik*, 112(2):241–251, 1966.

22 Paliath Narendran. Solving linear equations over polynomial semirings. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 466–472. IEEE, 1996.

23 Jean-Eric Pin. Tropical Semirings. In J. Gunawardena, editor, *Idempotency (Bristol, 1994)*, Publ. Newton Inst. 11, pages 50–69. Cambridge Univ. Press, Cambridge, 1998. URL: https://hal.archives-ouvertes.fr/hal-00113779.

24 Alexander Prestel. *Lectures on Formally Real Fields*, volume 1093 of *Lecture Notes in Mathematics*. Springer, 2007.

25 Alexander Prestel and Charles Delzell. *Positive Polynomials: From Hilbert's 17th Problem to Real Algebra*. Springer Science & Business Media, 2013.

26 N. S. Romanovskii. Some algorithmic problems for solvable groups. *Algebra and Logic*, 13(1):13–16, 1974.

A Omitted proofs

▶ **Proposition 3.7.** *Let v be a non-trivial real place of $\mathbb{R}(X)$ such that $\mathbb{R} \subseteq A_v$. Then v belongs to one of the two following types of real places:*

1. *For every $t \in \mathbb{R}$ there is a real place $v_t \colon \mathbb{R}(X) \to \mathbb{Z} \cup \{\infty\}$, defined by $v_t(y) = a$, where $a \in \mathbb{Z}$ is such that y can be written as $y = (X-t)^a \cdot \frac{f}{g}$, with f, g being polynomials in $\mathbb{R}[X]$ not divisible by $X - t$. The residue field $\mathbb{R}(X)_{v_t}$ is isomorphic to $\mathbb{R}$ by the natural homomorphism $y + M_{v_t} \mapsto y(t)$.*
2. *There is a real place $v_\infty \colon \mathbb{R}(X) \to \mathbb{Z} \cup \{\infty\}$, defined by $v_t(\frac{f}{g}) = \deg g - \deg f$, where f, g are polynomials in $\mathbb{R}[X]$. The residue field $\mathbb{R}(X)_{v_\infty}$ is isomorphic to $\mathbb{R}$ by the natural homomorphism $y + M_{v_\infty} \mapsto \lim_{t\to\infty} y(t)$.*

Proof. Since $\mathbb{R} \subseteq A_v$, every element $r \in \mathbb{R} \setminus \{0\}$ satisfies $v(r) \geq 0$ and $v(r^{-1}) \geq 0$. But $v(r) + v(r^{-1}) = v(1) = 0$, so $v(r) = 0$. Consider the value $v(X)$, there are two possibilities:

1. If $v(X) \geq 0$. In this case, we have $\mathbb{R} \subseteq A_v$ and $X \in A_v$, therefore $\mathbb{R}[X] \subseteq A_v$. Since M_v is a maximal (hence prime) ideal of A_v, the ideal $\mathbb{R}[X] \cap M_v$ is a prime ideal of $\mathbb{R}[X]$. Furthermore, $\mathbb{R}[X] \cap M_v$ is not zero, otherwise every element of $\mathbb{R}[X] \setminus \{0\}$ would be invertible in A_v, so $\mathbb{R}(X) \subseteq A_v$, contradicting the non-triviality of v. Since $\mathbb{R}[X]$ is a principle ideal domain, the non-zero prime ideal $\mathbb{R}[X] \cap M_v$ is generated by a single irreducible polynomial in $\mathbb{R}[X]$. Consider the two cases:
   a. The ideal $\mathbb{R}[X] \cap M_v$ is generated by a polynomial $X - t$ for some $t \in \mathbb{R}$. In this case we have $v(X - t) > 0$. Every polynomial $f \in \mathbb{R}[X]$ not divisible by $(X - t)$ can be written as $f = (X - t) \cdot F + r$ for some $F \in \mathbb{R}[X]$, $r \in \mathbb{R} \setminus \{0\}$. Since $v((X - t) \cdot F) = v(X - t) + v(F) > 0$ and $v(r) = 0$, we have $v(f) = v(r) = 0$.
   Every element $y \in \mathbb{R}(X)$ can be written as $y = (X-t)^a \cdot \frac{f}{g}$, where f, g are polynomials in $\mathbb{R}[X]$ not divisible by $(X - t)$. Then $v(y) = a \cdot v(X - t) + v(f) - v(g) = av(X - t)$. Under isomorphism of the value group Γ, we can without loss of generality suppose $v(X - t) = 1$, then we get the valuation v_t of type 1 described in the proposition. Since every element $y \in M_{v_t}$ satisfies $y(t) = 0$, we have that $y + M_{v_t} \mapsto y(t)$ is an isomorphism from the residue field to $\mathbb{R}$; it is a formally real field.
   b. The ideal $\mathbb{R}[X] \cap M_v$ is generated by a polynomial $X^2 + cX + d$ without real roots. In this case, the residue field A_v/M_v is a quadratic extension of $\mathbb{R}$, and is hence isomorphic to the field $\mathbb{C}$. However $\mathbb{C}$ is not formally real. Indeed, suppose on the contrary that $\mathbb{C}$ admits some ordering $\leq$, then since $0 < i^2 = -1$ and $0 < 1^2 = 1$, we have $0 < (-1) + 1 = 0$, a contradiction.
2. If $v(X) < 0$. In this case we have $\mathbb{R}[1/X] \subseteq A_v$ and $1/X \in M_v$. Since $\mathbb{R}[1/X] \cap M_v$ is a prime ideal of $\mathbb{R}[1/X]$ that contains $1/X$, it is generated by $1/X$. Then similar to the case 1.a., every element $y \in \mathbb{R}(X)$ can be written as $y = (1/X)^a \cdot \frac{F}{G}$, where F, G are polynomials in $\mathbb{R}[1/X]$ not divisible by $1/X$. Without loss of generality suppose $v(1/X) = 1$, we have $v(y) = a$. Rewrite $y = \frac{f}{g}$, comparing degrees, we have $a = \deg g - \deg f$. So v is the valuation v_∞ of type 2 described in the proposition. Since every element $y \in M_{v_\infty}$ satisfies $\lim_{t\to\infty} y(t) = 0$, we have that $y + M_{v_\infty} \mapsto \lim_{t\to\infty} y(t)$ is an isomorphism from the residue field to $\mathbb{R}$; it is a formally real field. ◀

▶ **Proposition 5.2.** *Given $h_1, \ldots, h_n \in \mathbb{Z}[X]$. The equation $f_1h_1 + \cdots + f_nh_n = 0$ has a solution $(f_1, \ldots, f_n)$ over $\mathbb{N}[X] \setminus \{0\}$ if and only if it has a solution over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$.*

Proof. A solution over $\mathbb{N}[X] \setminus \{0\}$ is obviously also a solution over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$. Conversely, let $f_i = \sum_{j=0}^{d_i} a_{ij}X^j, i = 1, \ldots, n$, be a solution of $f_1h_1 + \cdots + f_nh_n = 0$. Write $h_i = \sum_{j=0}^{e_i} b_{ij}X^j, i = 1, \ldots, n$, then the equation $f_1h_1 + \cdots + f_nh_n = 0$ is equivalent to the system of equations

$$\sum_{i=1}^{n} \sum_{j=0}^{d} a_{ij}b_{i,d-j} = 0, \quad d = 1, \ldots, \max_{1\leq i\leq n} (d_i + e_i). \tag{18}$$

All the coefficients b_{ij} are integers, and $b_{i,d-j} = 0$ whenever $d - j < 0$.

If $f_1h_1 + \cdots + f_nh_n = 0$ has a solution over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$, then System (18) has a solution $a_{ij}, i = 1, \ldots, n, j = 1, \ldots, d_i$ over $\mathbb{R}$, satisfying

$$a_{ij} \geq 0, \quad i = 1, \ldots, n, \quad j = 1, \ldots, d_i, \tag{19}$$

and

$$a_{i1} \neq 0 \text{ or } a_{i2} \neq 0 \text{ or } \ldots \text{ or } a_{id_i} \neq 0, \quad i = 1, \ldots, n. \tag{20}$$

This condition is a boolean combination of homogeneous linear inequalities with integer coefficients. Since the linear Systems (18), (19) and (20) have only integer coefficients, they have a solution over $\mathbb{R}$ if and only if they have a solution over $\mathbb{Q}$. Then, by their homogeneity, they have a solution over $\mathbb{Q}$ if and only if they have a solution over $\mathbb{Z}$. Hence, the Systems (18), (19), (20) have a solution over $\mathbb{Z}$, meaning $f_1h_1 + \cdots + f_nh_n = 0$ has a solution $f_i = \sum_{j=0}^{d_i} a_{ij}X^j, i = 1, \ldots, n$, over $\mathbb{N}[X] \setminus \{0\}$. ◀

▶ **Proposition 5.3.** *Given $h_1, \ldots, h_n \in \mathbb{Z}[X]$. The equation $f_1h_1 + \cdots + f_nh_n = 0$ has a solution $(f_1, \ldots, f_n)$ over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$ if and only if it has a solution over $\mathrm{U}(\mathbb{R}_{>0})$.*

Proof. Obviously a solution over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$ is a solution over $\mathrm{U}(\mathbb{R}_{>0})$.

For the other implication, we use Pólya's Theorem (Lemma 5.1). Suppose $f_1h_1 + \cdots + f_nh_n = 0$ has a solution $(f_1, \ldots, f_n)$ over $\mathrm{U}(\mathbb{R}_{>0})$. Write $f_i = X^{c_i} \cdot F_i$ where $c_i \geq 0$ and $F_i \in \mathbb{R}[X]$ is such that $X \nmid F_i$. Since $X \nmid F_i$ we have $F_i(0) \neq 0$, we claim that $F_i(0) > 0$. In fact, if $F_i(0) < 0$, then by the continuity of F_i, there exists $\varepsilon > 0$ such that $F_i(\varepsilon) < 0$, but then $f_i(\varepsilon) = \varepsilon^{c_i}F_i(\varepsilon) < 0$, contradicting the fact that $f_i \in \mathrm{U}(\mathbb{R}_{>0})$. Furthermore, one easily sees that $F_i(x) = \frac{f_i(x)}{x^{c_i}} > 0$ for all $x > 0$. So we have shown $F_i(x) > 0$ for all $x \geq 0$.

We now show that for large enough $p \in \mathbb{N}$, the polynomials $\hat{f}_i := (X+1)^p \cdot f_i$ are all in $\mathbb{R}_{\geq 0}[X]$. Let Y be a new variable, and for every i, let G_i be the homogenization of F_i using the variable Y. That is, $G_i = F_i(X/Y) \cdot Y^{\deg(F_i)}$. Since $F_i(x/y) > 0$ for all $x/y \geq 0$, we have $G_i(x, y) > 0$ for all $x \geq 0, y > 0$. Whereas for $x > 0, y = 0$, $G_i(x, y)/x^{\deg(F_i)}$ is the leading coefficient of F_i. This is non-zero and thus must be positive because $\lim_{x\to\infty} F_i(x) > 0$. Therefore $G_i(x, y) > 0$ for $x > 0, y = 0$.

We have thus shown $G_i(x, y) > 0$ for all $x \geq 0, y \geq 0, x + y > 0$. Applying Pólya's Theorem yields the existence of a $p_i \in \mathbb{N}$ such that $(X+Y)^{p_i} \cdot G_i \in \mathbb{R}_{\geq 0}[X, Y]$. Taking $Y = 1$ we dehomogenize G_i and obtain $(X+1)^{p_i} \cdot F_i \in \mathbb{R}_{\geq 0}[X]$. Let $p = \max\{p_1, \ldots, p_n\}$, then

$$\hat{f}_i = (X+1)^p \cdot f_i = X^{c_i} \cdot (X+1)^p \cdot F_i \in \mathbb{R}_{\geq 0}[X] \setminus \{0\}$$

for all i. We have thus found the solution $(\hat{f}_1, \ldots, \hat{f}_n)$ over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$ for the equation $f_1h_1 + \cdots + f_nh_n = 0$. ◀

▶ **Remark A.1.** Proposition 5.3 no longer holds if we replace $\mathrm{U}(\mathbb{R}_{>0})$ with $\mathrm{W}(\mathbb{R}_{>0}) \setminus \{0\}$. For example, take $n = 2, h_1 = 1, h_2 = -(X-1)^2$. Then $f_1 = (X-1)^2, f_2 = 1$ is a solution over $\mathrm{W}(\mathbb{R}_{>0}) \setminus \{0\}$ of the equation $f_1h_1 + f_2h_2 = 0$. However, $f_1h_1 + f_2h_2 = 0$ does not admit a solution over $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$. Indeed, any solution of $f_1 - f_2 \cdot (X-1)^2 = 0$ over $\mathbb{R}[X]$ must satisfy $f_1(1) = 0$, so f_1 cannot be in $\mathbb{R}_{\geq 0}[X] \setminus \{0\}$.

▶ **Theorem 2.4.** *Given a finite set of elements $\mathcal{G} = \{(y_1, b_1), \ldots, (y_n, b_n)\}$ in $\mathbb{Z} \wr \mathbb{Z}$, where $b_i = \pm 1$ for all i. The following are decidable:*

1. *(Group Problem) whether the semigroup $\langle \mathcal{G} \rangle$ generated by $\mathcal{G}$ is a group.*
2. *(Identity Problem) whether the neutral element I is in the semigroup $\langle \mathcal{G} \rangle$.*

Proof.

1. For the Group Problem, by Proposition 7.1 it suffices to decide whether there exists a set $S \subseteq I \times J$ satisfying $\pi_I(S) = I, \pi_J(S) = J$, such that the equation $\sum_{(i,j)\in S} f_{ij}h_{ij} = 0$ has a solution $(f_{ij})_{(i,j)\in S}$ over $\mathbb{N}[X] \setminus \{0\}$. By the homogeneity of the equation $\sum_{(i,j)\in S} f_{ij}h_{ij} = 0$, one can multiply all the Laurent polynomials h_{ij} by a power of X and suppose all $h_{ij} \in \mathbb{Z}[X]$. For every set $S \subseteq I \times J$ satisfying $\pi_I(S) = I, \pi_J(S) = J$, we can use Theorem 2.3 to decide whether $\sum_{(i,j)\in S} f_{ij}h_{ij} = 0$ has a solution over $\mathbb{N}[X] \setminus \{0\}$. This shows the decidability of the Group Problem.
2. The neutral element is in $\langle \mathcal{G} \rangle$ if and only if a non-empty subset of $\mathcal{G}$ generates a group (as a semigroup). This is because, if the product of a word $w \in \mathcal{G}^+$ is the neutral element, then every element in the set C of letters used in w can be inverted in $\langle C \rangle$, so $\langle C \rangle$ is a group. Therefore, in order to decide whether the neutral element is in $\langle \mathcal{G} \rangle$, it suffices to check for all subsets of $\mathcal{G}$ whether they generate a group. This is decidable by the above result on the Group Problem. ◀

B Algorithm for Theorem 2.2

Algorithm 1 Deciding existence of solutions over $\mathbb{N}[X] \setminus \{0\}$ of the equation $f_1h_1 + \cdots + f_nh_n = 0$.

Input: Polynomials $h_1, \ldots, h_n \in \mathbb{Z}[X]$.
Output: **True** or **False**.

(1) Compute $d \coloneqq \gcd(h_1, \ldots, h_n)$ and divide all h_i by d.
(2) Repeat the following:
  a. If $h_i(0) > 0$ for all i, or $h_i(0) < 0$ for all i, return **False**.
  b. Else if $h_i(0) \geq 0$ for all i, or $h_i(0) \leq 0$ for all i, divide all the polynomials h_i that satisfy $h_i(0) = 0$ by X.
  c. Else go to 3.
(3) Decide the truth of the existential sentence (16) in the theory of reals. If (16) is true, return **False**, otherwise return **True**.

C Comparison with the Bröcker-Prestel local-global principle

The original Bröcker-Prestel local-global principle ([24, Theorem 8.13]) can be formulated as follows.

▶ **Theorem C.1** (Bröcker-Prestel local-global principle). *Let F be a formally real field, and $h_1, \ldots, h_n$ be non-zero elements of F. If the equation $f_1h_1 + \cdots f_nh_n = 0$ has no non-trivial solution $(f_1, \ldots, f_n) \neq (0, \ldots, 0)$ over* sums of squares *of F (that is, over the set $S \coloneqq \{\sum_{i=1}^k a_i^2 \mid a_i \in F\}$), then at least one of the following hold:*

(i) *$h_1, \ldots, h_n$ are all of the same sign in some archimedean ordering of F.*
(ii) *$f_1h_1 + \cdots + f_nh_n = 0$ has no solution in the* Henselization *of some real place of F.*

For a definition of Henselizations of a formally real field, see [24, Proposition 8.1].

When applied to the field $F = \mathbb{R}(X)$, the Bröcker-Prestel local-global principle characterizes the absence of non-trivial solutions over sums of squares by condition (ii), since the field $\mathbb{R}(X)$ has no archimedean orderings. Multiplying by the common denominator and using the fact that any element in $\mathrm{W}(\mathbb{R})$ can be written as a sum of squares in $\mathbb{R}(X)$,

Theorem C.1 also characterizes the absence of non-trivial solutions over $\mathrm{W}(\mathbb{R})$. However, when considering non-trivial solutions over $\mathrm{U}(\mathbb{R})$ and $\mathrm{U}(B)$, the situation is quite different; and we now compare the proof of Theorem 2.1 to Theorem C.1.

The proof of Bröcker-Prestel's original theorem starts with the definition of the pre-semicone

$$P_1 \coloneqq \left\{ \sum_{i=1}^{n} f_i h_i, \text{ where } f_i \text{ are sum of squares of elements in } \mathbb{R}(X) \right\}.$$

Since it considers solutions over sum of squares, this definition is straightforward. The definition of P_0 is our proof of Theorem 2.1 is different and less straightforward. In our theorem, we are considering *strictly* positive polynomials on $B \subseteq \mathbb{R}$, therefore we need to replace sum of squares with polynomials in $\mathrm{U}(B)$. However, such a naive replacement does not work due to the requirement of a pre-semicone to be closed under multiplication of squares (unlike $\mathrm{W}(\mathbb{R})$, the set $\mathrm{U}(B)$ is not closed under multiplication by squares). This is why we need to add the rational function $\frac{g}{G}$ in the definition of P_0 and use the fundamental theorem of algebra to guarantee closure under addition.

Note that in order to guarantee the closure under addition of P_0, it is essential that we work in the *univariate* polynomial ring $\mathbb{R}[X]$, so that two polynomials g, g' having a common root implies $\gcd(g, g') \neq 1$. For example, this no longer holds in the bivariate polynomial ring $\mathbb{R}[X, Y]$. Therefore, even when supposing $\gcd(\frac{gG'}{dD}, \frac{g'G}{dD}) = 1$, we no longer have $\left(f_i \frac{gG'}{dD} + f_i' \frac{g'G}{dD}\right)(x, y) > 0$ in Equation (5). Thus, for the field $\mathbb{R}(X, Y)$, the closure under addition of P_0 no longer holds, a contrast with the "non-strict" version P_1.

The following step of extracting the valuation ring $A^P_{\mathbb{R}}$ from the semiordering P appeared as part of the proof of the original theorem. (The original theorem used the valuation ring $A^P_{\mathbb{Q}}$ instead, but they are in fact equivalent.) This is the main part where we drew inspiration from the original local-global principle.

After extracting the valuation ring $A^P_{\mathbb{R}}$, our proof again diverges from that of the original theorem. Our new definition of P_0 allows us to enforce strict positivity, however it also takes away some convenient properties of the pre-semicone P_1 in the original theorem. Notably, we have $h_1 \in P_1$, allowing for a quick conclusion on the positivity of h_1 in Henselizations. Whereas for P_0, we do not have $h_1 \in P_0$ due to the strict positivity of the coefficients f_i. We compensate this by the analytic approach adopted in the second half of our proof, making use of the classification of real places of $\mathbb{R}(X)$ and the continuity of functions in $\mathbb{R}[X]$. This part is absent from the proof of the original theorem, which is purely algebraic and model theoretic.

An Approximation Algorithm for Distance-Constrained Vehicle Routing on Trees

Marc Dufay
École Polytechnique and IRIF, Palaiseau, France

Claire Mathieu
CNRS Paris, France

Hang Zhou
École Polytechnique, Palaiseau, France

Abstract

In the Distance-constrained Vehicle Routing Problem (DVRP), we are given a graph with integer edge weights, a depot, a set of n terminals, and a distance constraint D. The goal is to find a minimum number of tours starting and ending at the depot such that those tours together cover all the terminals and the length of each tour is at most D.

The DVRP on *trees* is of independent interest, because it is equivalent to the "virtual machine packing" problem on trees studied by Sindelar et al. [SPAA'11]. We design a simple and natural approximation algorithm for the tree DVRP, parameterized by $\varepsilon > 0$. We show that its approximation ratio is $\alpha + \varepsilon$, where $\alpha \approx 1.691$, and in addition, that our analysis is essentially tight. The running time is polynomial in n and D. The approximation ratio improves on the ratio of 2 due to Nagarajan and Ravi [Networks'12].

The main novelty of this paper lies in the analysis of the algorithm. It relies on a reduction from the tree DVRP to the bounded space online bin packing problem via a new notion of "reduced length".

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases vehicle routing, distance constraint, approximation algorithms, trees

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.27

Funding This work was partially supported by the grant ANR-19-CE48-0016 from the French National Research Agency (ANR).

1 Introduction

The vehicle routing problem is arguably one of the most important problems in Operations Research. Books have been dedicated to vehicle routing problems, e.g., [4, 7, 16]. Yet, these problems remain challenging, both from a practical and a theoretical perspective. As observed by Li, Simchi-Levi, and Desrochers [11]:

> Typically, two types of constraints are imposed on the route traveled by any vehicle. One is the *capacity constraint* in which each vehicle cannot serve more than a given number of customers. The second is the *distance constraint*: The total distance traveled by each vehicle should not exceed a prespecified number. Depending on the system, one or both types can be binding. Usually delivery and pick-up services are characterized by capacity constraints [...], while service systems are characterized by distance constraints (see, for example, [1]). On the latter case, the system is usually required to provide a visit of a skilled worker at customer sites and thus the length of routes must be controlled because these are related to working hours.

© Marc Dufay, Claire Mathieu, and Hang Zhou;
licensed under Creative Commons License CC-BY 4.0

The focus of this paper is the distance constraint. In the *Distance-constrained Vehicle Routing Problem (DVRP)*, we are given a graph with integer edge weights, a vertex called *depot*, a set of n vertices called *terminals*, and an integer *distance constraint* D. The goal is to find a minimum number of tours starting and ending at the depot such that those tours together cover all the terminals and the length of each tour is at most D. Friggstad and Swamy [5] gave an $O(\frac{\log D}{\log\log D})$-approximation, improving upon an $O(\log D)$-approximation of Nagarajan and Ravi [13].[1] Experimental results were given in [9].

The DVRP on *trees* is of independent interest, because of its relation to the *Virtual Machine (VM) packing problem* [15, 2, 14]. In the VM packing problem, we are given a set of VMs that must be hosted on physical servers, where each VM consists of a set of pages and each physical server has a capacity of D pages. VMs running on the same physical server may share pages. The goal is to pack the VMs onto the smallest number of physical servers. Sindelar et al. [15] observed:

> Using memory traces for a mixture of diverse OSes, architectures, and software libraries, we find that *a tree model* can capture up to 67% of inter-VM sharing from these traces.

Sindelar et al. [15] gave a 3-approximation for the VM packing problem on trees, and also suggested as future work "*A key direction is tightening the approximation bounds*".

It is easy to see that the VM packing problem on trees is equivalent to the DVRP on trees. Thus the algorithm of Sindelar et al. [15] is a 3-approximation for the tree DVRP. Nagarajan and Ravi [13] improved the ratio of the tree DVRP to 2. When the distance bound is allowed to be violated by an ε fraction, Becker and Paul [3] designed a bicriteria PTAS. We observe that the tree DVRP is strongly NP-hard (Appendix A).

In this work, we design a simple and natural approximation algorithm for the tree DVRP, parameterized by $\epsilon > 0$, see Algorithm 1. The main novelty lies in the analysis of Algorithm 1. Our main result (Theorem 2) shows that the approximation ratio of Algorithm 1 is $\alpha + O(\epsilon)$, where $\alpha \approx 1.691$ is defined in Definition 1. The running time is polynomial in n and D. Interestingly, the ratio α is best possible for Algorithm 1 (Theorem 3).

▶ **Definition 1.** *Let* $(u_k)_{k\geq 1}$ *denote the following sequence:*

$$u_k = \begin{cases} 1, & k = 1, \\ u_{k-1}(u_{k-1}+1), & k \geq 2. \end{cases}$$

$$\text{Let } \alpha := \sum_{k=1}^{+\infty} \frac{1}{u_k} = 1.69103\ldots.$$

▶ **Theorem 2.** *For any constant* $\varepsilon > 0$*, Algorithm 1 is an* $(\alpha + O(\epsilon))$*-approximation for the* Distance-constrained Vehicle Routing Problem (DVRP) *on trees. Its running time is* $O\left(n^2 \cdot \left(D/\epsilon^2\right)^{O(1/\epsilon^2)}\right)$.

▶ Remark. It is common in vehicle routing to parameterize the running time of an algorithm by the value of a constraint. For example, in capacitated vehicle routing with splittable demands, the running time of the algorithms in [8, 12] is parameterized by the *tour capacity*.

[1] More precisely, Nagarajan and Ravi [13] designed a bicriteria $\left(O(\log \frac{1}{\varepsilon}), 1+\varepsilon\right)$-approximation, which could be turned into an $O(\log D)$-approximation by setting $D = \frac{1}{\varepsilon}$.

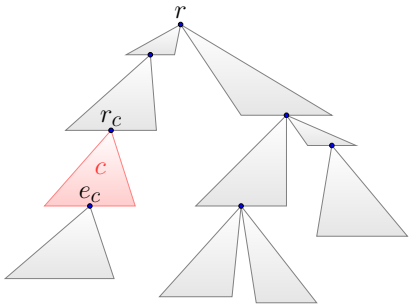


Figure 1 Component decomposition. Extracted from [12].

▶ **Theorem 3.** *For any constant $\varepsilon > 0$, Algorithm 1 is at best an α-approximation for the* Distance-constrained Vehicle Routing Problem (DVRP) *on trees.*

It is an open question to achieve a better-than-α approximation for the DVRP on trees. From Theorem 3, this would require new insights in the algorithmic design.

1.1 Algorithm

Let $\varepsilon > 0$. Our algorithm for the DVRP is Algorithm 1. It consists of two phases, using Algorithm 2 and Algorithm 3 with $\Gamma = 1/\varepsilon^2$:

Phase 1: The tree is partitioned into *components* (Lemma 5 and Algorithm 2), where each component can be covered with a bounded number of tours.

Phase 2: Each component is taken as an independent instance for the DVRP, which is solved using a polynomial time dynamic program (Lemma 7 and Algorithm 3); the solution for the whole tree is the union of the solutions for individual components.

Algorithm 1 Approximation algorithm for the DVRP on trees. Parameter $\varepsilon > 0$.

Input: A tree T rooted at r, a set of terminals U, a distance constraint D
Output: number of tours in a feasible solution to cover all terminals in U

1: $\Gamma \leftarrow 1/\epsilon^2$
2: Partition the tree T into a set $\mathcal{C}$ of components ▷ Algorithm 2
3: **for** each component $c \in \mathcal{C}$ **do**
4: $r_c \leftarrow$ root vertex of component c ▷ defined in Lemma 5
5: $D_c \leftarrow D$ minus twice the distance between r and r_c
6: $U_c \leftarrow$ set of terminals in U that belong to c
7: $n_c \leftarrow$ minimum number of tours for the subproblem (c, U_c, D_c) ▷ Algorithm 3
8: **return** $\sum_{c \in \mathcal{C}} n_c$

▶ Remark. As written, Algorithm 1 returns the number of tours, not the tours themselves. If desired, by adding auxiliary information to the dynamic program of Algorithm 3 in a standard manner, it is possible to retrieve a feasible solution whose number of tours matches the value returned by Algorithm 1.

1.2 Overview of the Analysis

This section gives a high-level description of main new ideas in this paper.

The analysis of Algorithm 1 relies on a reduction (Theorem 14) from the DVRP on trees to the *bounded space online bin packing* (Definition 12). In the *bounded space online bin packing*, the items arrive in an online fashion, and in addition, at any point in the packing process, only a fixed number of partially-filled bins may be *active*, i.e., open to further items. Once a bin is closed it must remain so and can no longer be active.

What does bin packing have to do with DVRP on trees? The relation lies in a new notion introduced in this paper, that of *reduced lengths* (Definition 8). If we consider a bin in bin packing, its item sizes sum to at most 1. Similarly, if we consider a tour in DVRP on trees, the reduced lengths of its subtours in components sum to at most 1 (Lemma 9).

To show the reduction (Theorem 14), we start from an instance of the tree DVRP and we construct an instance of the bounded space online bin packing as follows. We consider the tours in an optimal solution and decompose each tour into a set of subtours, one for each component it visits. For each subtour, we consider its *reduced length*, which is defined with respect to the component. We then construct an online sequence of those reduced lengths such that reduced lengths of the subtours in the same component are *consecutive* in the sequence. Then we consider a solution to the bounded space online bin packing on that sequence. Intuitively, the bound on the number of open bins implies that bins in that solution tend to contain only reduced lengths of the subtours from the same component. That is a desirable property because, if a bin contains only reduced lengths of subtours in the same component, then those subtours can be combined into a single tour (Lemma 10). Using the above intuition, we show that the performance of Algorithm 1 for the tree DVRP is up to some negligible additive factor at least as good as the best corresponding bounded space bin-packing problem.

From the reduction (Theorem 14) and since the bounded space online bin packing admits an algorithm with worst-case performance ratio α due to Lee and Lee [10], we conclude that Algorithm 1 is an $(\alpha + O(\varepsilon))$ approximation for the DVRP on trees (Theorem 2).

There are several technical details along the way. For example, subtours that end in a component and subtours that traverse a component cannot be combined in the same way, which requires additional care in the definition of reduced lengths, see Figure 2.

Finally, we show that the analysis in Theorem 2 on the approximation ratio of Algorithm 1 is essentially tight by providing a matching lower bound (Theorem 3).

1.3 Organization of the Paper

In Section 2, we give the formal problem definition, preliminary results, and previous techniques. In Section 3, we define and analyze *reduced lengths*. In Section 4, we prove Theorem 2 by establishing the reduction (Theorem 14). In Section 5, we prove Theorem 3.

2 Preliminaries

2.1 Formal Problem Definition, Notations, and Assumptions

Let T be a rooted tree (V, E) with non-negative integer edge weights $w(u, v)$ for all $(u, v) \in E$. Consider a tour (resp. subtour) $t = (v_0, v_1, v_2, \ldots, v_m)$ for some $m \in \mathbb{N}$ such that $v_0 = v_m$. The *length* of t, denoted by $\text{length}(t)$, is defined to be $\sum_{i=1}^{m} w(e_i)$, where e_i is the edge between v_{i-1} and v_i.

▶ **Definition 4** (tree DVRP). *An instance* (T, U, D) *of the* Distance-constrained Vehicle Routing Problem (DVRP) *on* trees *consists of*

- *a rooted* tree $T = (V, E)$ *with non-negative integer edge weights, where the root* $r \in V$ *of the tree is the* depot*;*
- *a set* $U \subseteq V$ *of* n *vertices of the tree* T*, called* terminals*;*
- *a positive integer* distance constraint D*.*

A feasible solution is a set of tours such that

- *each tour starts and ends at* r*;*
- *each terminal is visited by at least one tour;*
- *each tour has length at most* D*.*

The goal is to minimize the total number of tours in a feasible solution.

Let OPT denote an optimal solution to the tree DVRP. Let opt denote the number of tours in OPT.

For any vertices $u, v \in V$, let $\text{dist}(u, v)$ denote the *distance* between u and v in the tree T.

Up to a preprocessing step, we can assume that each vertex has at most two children and that the terminals are the same as the leaves of the tree, see, e.g., [12]. Furthermore, we assume that each non-leaf vertex has exactly two children.[2] We assume that for each tour in the solution, each edge on that tour is traversed exactly twice, once in each direction. We assume w.l.o.g. that $\varepsilon > 0$ is upper bounded by a sufficiently small constant.

2.2 Decomposition into Components

We decompose the tree into components in Lemma 5.

▶ **Lemma 5** ([12]). *Let* $\Gamma \geq 1$ *be a fixed integer. There is an algorithm* $\textsc{Decompose}^{(\Gamma)}$ *(Algorithm 2) that runs in time* $O(n^2 \cdot (2\Gamma)^\Gamma \cdot D^{3\Gamma})$ *and computes a partition of the edges of the tree* T *into a set* $\mathcal{C}$ *of* components *(see Figure 1), such that all of the following properties are satisfied:*

1. *Every component* $c \in \mathcal{C}$ *is a connected subgraph of* T*; the* root *vertex of the component* c*, denoted by* r_c*, is the vertex in* c *that is closest to the depot.*
2. *We say that a component* $c \in \mathcal{C}$ *is a* leaf *component if all descendants of* r_c *in tree* T *are in* c*, and an* internal *component otherwise. An internal component* c *shares vertices with other components at two vertices only: at vertex* r_c*, and at one other vertex, called the* exit *vertex of the component* c*, and denoted by* e_c*.*
3. *The terminals of each component can be covered by at most* Γ *tours. We say that a component is* big *if at least* $\Gamma/2$ *tours are needed to cover its terminals. Each leaf component is big.*
4. *If the number of components in* $\mathcal{C}$ *is strictly greater than 1, then there exists a map from all components to big components, such that each big component has at most three pre-images.*

The component decomposition was previously given in [12], inspired by Becker and Paul [3]. Algorithm 2 is a small adaptation from [12], and is given in Appendix B. The proof of Lemma 5 is almost identical to that in [12], hence omitted.

[2] Indeed, if a vertex u has only one child v, there are two cases. In the first case, u is the root. Then we remove u and its incident edge, let v be the depot, and update D by $D - 2w(u, v)$. In the second case, u has a parent p. Then we remove u and its incident edges, and we add an edge between p and v with weight $w(p, v) := w(p, u) + w(u, v)$.

▶ **Definition 6** (subtours and their categories, [12]). *Let $c \in \mathcal{C}$ be any component. A* subtour *in component c is a walk inside c that starts and ends at r_c and visits at least one terminal. For any subtour s in component c, we say that the* category *of s is* passing *if c is an internal component and the exit vertex e_c belongs to s, and* ending *otherwise.*

2.3 Solving Instances with a Bounded Number of Tours

For an instance of the tree DVRP admitting a solution of a bounded number of tours, an optimal solution can be computed in polynomial time using a simple dynamic program (Algorithm 3), in Lemma 7.

▶ **Lemma 7.** *Let $\Gamma \geq 1$ be a fixed integer. There is an algorithm $\textsc{Solve}^{(\Gamma)}$ (Algorithm 3) that, given an instance $(\tilde{T}, \tilde{U}, \tilde{D})$ of the DVRP on trees, computes*

$$\min\{|S| : S \text{ is a feasible set of tours and } |S| \leq \Gamma\}.$$

Thus $\textsc{Solve}^{(\Gamma)}$ returns $+\infty$ if the instance does not admit any solution of at most Γ tours. The running time of $\textsc{Solve}^{(\Gamma)}$ is $O(\tilde{n} \cdot (2\Gamma)^{\Gamma} \cdot \tilde{D}^{3\Gamma})$, where $\tilde{n}$ is the number of terminals in $\tilde{T}$.

Algorithm 3 and the proof of Lemma 7 are in Appendix C.

3 Reduced Lengths

In this section, we introduce a novel concept of *reduced lengths* (Definition 8).

▶ **Definition 8** (reduced lengths, see Figure 2). *Let $c \in \mathcal{C}$ be any component. For any subtour s in component c, we define the* reduced length $\bar{\ell}(s)$ *of subtour s as follows:*

- *If s is an* ending subtour*,*

$$\bar{\ell}(s) := \frac{\text{length}(s)}{D - 2 \cdot \text{dist}(r, r_c)};$$

- *If s is a* passing subtour*,*

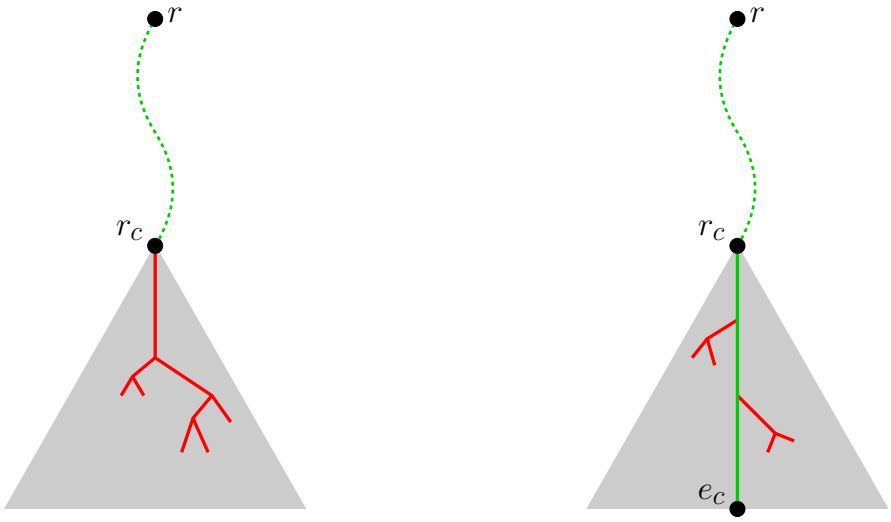
$$\bar{\ell}(s) := \frac{\text{length}(s) - 2 \cdot \text{dist}(r_c, e_c)}{D - 2 \cdot \text{dist}(r, e_c)}.$$

We distinguish the two categories of the subtours in the definition of reduced lengths (Definition 8) so that the properties in Lemmas 9 and 10 are satisfied.

▶ **Lemma 9.** *Let t be a tour. Let $c_1, \ldots, c_k$ be the components containing terminals visited by t. For each $i \in [1, k]$, let s_i denote the subtour of t in c_i. Then $\sum_{i=1}^{k} \bar{\ell}(s_i) \leq 1$.*

Proof. Let $c^* \in \{c_1, \ldots, c_k\}$ be a component such that $\text{dist}(r, r_{c^*})$ is maximized. Let p be the path from r to r_{c^*}. For each $i \in [1, k]$, let $E_i \subseteq E$ denote a subset of edges defined as follows: if s_i is an ending subtour, then E_i consists of the edges in s_i; and if s_i is a passing subtour, then E_i consists of the edges in s_i that do not belong to the r_{c_i}-to-e_{c_i} path. By construction, the edges in $p, E_1, \ldots, E_k$ are disjoint. Let $w(E_i)$ denote $\sum_{e \in E_i} w(e)$. Since the length of t is at most D, we have:

$$2 \cdot \text{dist}(r, r_{c^*}) + \sum_{i=1}^{k} 2 \cdot w(E_i) \leq D,$$



(a) Ending subtour s. **(b)** Passing subtour s.

Figure 2 Illustration for the new notion of the *reduced length* of a subtour s in a component c. The component c is represented by the gray triangle. There are two cases depending on whether s is an ending subtour (Figure 2a) or a passing subtour (Figure 2b). In both cases, the subtour s is represented by the solid segments. The r-to-r_c connection (in both directions) is represented by the dashed curve. The reduced length $\bar{\ell}(s)$ of the subtour s is defined by $\frac{\text{length(red)}}{D - \text{length(green)}}$, which is proportional to the red part of the path. See Definition 8.

and equivalently,

$$\sum_{i=1}^{k} \frac{2 \cdot w(E_i)}{D - 2 \cdot \text{dist}(r, r_{c^*})} \leq 1. \tag{1}$$

For each $i \in [1, k]$, by the definition of E_i, we have

$$2 \cdot w(E_i) = \begin{cases} \text{length}(s_i), & \text{if } s_i \text{ is an ending subtour} \\ \text{length}(s_i) - 2 \cdot \text{dist}(r_{c_i}, e_{c_i}), & \text{if } s_i \text{ is a passing subtour.} \end{cases} \tag{2}$$

By the choice of c^*, we have

$$\text{dist}(r, r_{c^*}) \geq \begin{cases} \text{dist}(r, r_{c_i}), & \text{if } s_i \text{ is an ending subtour} \\ \text{dist}(r, e_{c_i}), & \text{if } s_i \text{ is a passing subtour.} \end{cases} \tag{3}$$

From Equations (2) and (3) and Definition 8, for all $i \in [1, k]$, we have

$$\bar{\ell}(s_i) \leq \frac{2 \cdot w(E_i)}{D - 2 \cdot \text{dist}(r, r_{c^*})}.$$

Combining with Equation (1), the claim follows. ◀

▶ **Lemma 10.** *Let c be any component. Let $s_1, \ldots, s_m$, for some $m \geq 1$, be any subtours in c that are of the same category and such that $\sum_{i=1}^{m} \bar{\ell}(s_i) \leq 1$. Then all terminals from all subtours $s_1, \ldots, s_m$ can be visited by a tour of length at most D.*

Proof. We construct a tour t visiting all terminals from all subtours $s_1, \ldots, s_m$, and we show that its length is at most D. Since $s_1, \ldots, s_m$ are of the same category, there are two cases depending on their category.

- If $s_1, \ldots, s_m$ are ending subtours, then t consists of the r-to-r_c connection (in both directions) and of the subtour s_i for each $i \in [1, m]$. Therefore,

$$\begin{aligned}\text{length}(t) &\leq 2 \cdot \text{dist}(r, r_c) + \sum_{i=1}^{m} \text{length}(s_i) \\ &= 2 \cdot \text{dist}(r, r_c) + (D - 2 \cdot \text{dist}(r, r_c)) \sum_{i=1}^{m} \bar{\ell}(s_i) && \text{(by Definition 8)} \\ &\leq D && \text{(since } \sum_{i=1}^{m} \bar{\ell}(s_i) \leq 1\text{).}\end{aligned}$$

- If $s_1, \ldots, s_m$ are passing subtours, then t consists of the r-to-e_c connection (in both directions) and of the subtour s_i without the r_c-to-e_c connection (in both directions) for each $i \in [1, m]$. Therefore,

$$\begin{aligned}\text{length}(t) &\leq 2 \cdot \text{dist}(r, e_c) + \sum_{i=1}^{m} (\text{length}(s_i) - 2 \cdot \text{dist}(r_c, e_c)) \\ &= 2 \cdot \text{dist}(r, e_c) + (D - 2 \cdot \text{dist}(r, e_c)) \sum_{i=1}^{m} \bar{\ell}(s_i) && \text{(by Definition 8)} \\ &\leq D && \text{(since } \sum_{i=1}^{m} \bar{\ell}(s_i) \leq 1\text{).}\end{aligned}$$

The claim follows in both cases. ◀

▶ **Lemma 11.** *Assuming $\Gamma \geq 20$, the number of components in $\mathcal{C}$ is at most $(15/\Gamma) \cdot \text{opt}$.*

Proof. Let $\mathcal{C}_b \subseteq \mathcal{C}$ denote the set of big components in $\mathcal{C}$. Consider a big component $c \in \mathcal{C}_b$. Let $s_{c,1}, ..., s_{c,m_c}$ denote all subtours in c in the global optimal solution, for some $m_c \in \mathbb{N}$. Let $\bar{\ell}_{c,1}, \ldots, \bar{\ell}_{c,m_c}$ denote their reduced lengths. Those subtours can be viewed as a feasible solution to the local problem for the terminals in c only. We partition those subtours into parts, such that the subtours in the same part are of the same category, and in addition, for each part except possibly two parts, the reduced lengths of the subtours in that part sum to a value in $(1/2, 1]$. For each part, we replace the subtours by a new subtour visiting the terminals covered by all the subtours in that part. This creates a new feasible local solution for the terminals of c, and its number of subtours is at most $2\sum_{i=1}^{m_c} \bar{\ell}_{c,i} + 2$. By definition of big components, covering the terminals of c requires at least $\Gamma/2$ tours. Thus:

$$2\sum_{i=1}^{m_c} \bar{\ell}_{c,i} + 2 \geq \Gamma/2.$$

Therefore,

$$\sum_{i=1}^{m_c} \bar{\ell}_{c,i} \geq \Gamma/4 - 1 \geq \Gamma/5,$$

since $\Gamma \geq 20$. Hence

$$\sum_{c \in \mathcal{C}} \sum_{i=1}^{m_c} \bar{\ell}_{c,i} \geq |\mathcal{C}_b| \cdot \Gamma/5 \geq |\mathcal{C}| \cdot \Gamma/15,$$

where the last inequality follows from Property 4 of Lemma 5.

On the other hand, consider each tour t in OPT. Let k_t denote the number of components containing terminals visited by tour t. For each $i \in [1, k_t]$, let $\bar{\ell}'_{t,i}$ denote the reduced length of the i^{th} subtour of t. By Lemma 9 we have:

$$\sum_{t \in \text{OPT}} \sum_{i=1}^{k_t} \bar{\ell}'_{t,i} \leq \sum_{t \in \text{OPT}} 1 = \text{opt}.$$

By re-ordering this sum we get

$$\sum_{t \in \text{OPT}} \sum_{i=1}^{k_t} \bar{\ell}'_{t,i} = \sum_{c \in \mathcal{C}} \sum_{i=1}^{m_c} \bar{\ell}_{c,i}.$$

Therefore, $\text{opt} \geq |\mathcal{C}| \cdot \Gamma/15$. The claim follows. ◀

4 Proof of Theorem 2

To prove Theorem 2, we use a reduction from the DVRP on trees to the *bounded space online bin-packing*.

4.1 Bounded Space Online Bin Packing

▶ **Definition 12.** *In the* bounded space online bin-packing *problem, we are given a positive integer M and an online sequence of item sizes $(a_1, a_2, ..., a_n) \in [0, 1]^n$, and we want to pack those items into bins of size* 1*. At any time, the following operations are allowed: (1) opening a bin; (2) closing an bin; (3) assigning the current item to some* open *bin. We require that at any time there are at most M open bins.*

The goal is to minimize the total number of bins used.

Let opt_{BP} denote the number of bins in an optimal solution to the bin packing in the *offline* setting. The following theorem due to Lee and Lee [10] is a direct corollary of Theorem 2 from [10] and Eq. (3.6) from [10].

▶ **Theorem 13** ([10])**.** *Let M be any positive integer. Let $k \in \mathbb{N}$ be such that $u_k < M \leq u_{k+1}$. There exists a solution to the bounded space online bin-packing*

in which the total number of bins used is at most

$$\left(\sum_{i=1}^{k} \frac{1}{u_i} + \frac{M}{(M-1)u_{k+1}} \right) \cdot \text{opt}_{\text{BP}} + (M-1).$$

4.2 Reduction

In this subsection, we prove the following Theorem 14.

▶ **Theorem 14.** *Assume that the bounded space online bin-packing problem admits a solution using at most $\beta_M \cdot \mathrm{opt}_{\mathrm{BP}} + \gamma_M$ bins, where β_M and γ_M are parameters depending on the space bound M, and $\mathrm{opt}_{\mathrm{BP}}$ is defined in Section 4.1. Let $\varepsilon > 0$. Then Algorithm 1 uses at most $(\beta_M + 30\varepsilon^2 \cdot M) \cdot \mathrm{opt} + \gamma_M$ tours for the tree DVRP.*

Consider an instance of the DVRP on a tree. Let m denote the number of components, and let $\mathcal{C} = \{c_1, c_2, ..., c_m\}$ denote the set of components. For each component $c_i \in \mathcal{C}$, let $\bar{\ell}_{i,1}, \bar{\ell}_{i,2}, ..., \bar{\ell}_{i,e_i}$ denote the reduced lengths of the ending subtours in c_i from OPT, and $\bar{\ell}'_{i,1}, \bar{\ell}'_{i,2}, ..., \bar{\ell}'_{i,p_i}$ denote the reduced lengths of the passing subtours in c_i from OPT. We define the following instance of the bounded space online bin-packing:

$$L = (\bar{\ell}_{1,1}, \bar{\ell}_{1,2}, ..., \bar{\ell}_{1,e_1}, \bar{\ell}'_{1,1}, \bar{\ell}'_{1,2}, ..., \bar{\ell}'_{1,p_1}, ..., \bar{\ell}_{m,1}, \bar{\ell}_{m,2}, ..., \bar{\ell}_{m,e_m}, \bar{\ell}'_{m,1}, \bar{\ell}'_{m,2}, ..., \bar{\ell}'_{m,p_m}).$$

Let B_1 denote a solution to this instance satisfying the assumption of the claim, i.e.,

$$|B_1| \leq \beta_M \cdot \mathrm{opt}_{\mathrm{BP}} + \gamma_M. \tag{4}$$

For each bin b in B_1, we partition its contents so that each part contains the reduced lengths of the subtours that come from the same component and are in the same category, and we replace b by a collection of bins, one for each part of the partition containing at least one subtour of b. This defines a bin-packing B_2.

Next, we define a solution S to the tree DVRP corresponding to B_2. For each bin b of B_2, we consider the subtours that correspond to the reduced lengths in b, and we create one tour in S, which is the minimum tour visiting all terminals covered by those subtours. Observe that those subtours are in the same component and of the same category and, in addition, their reduced lengths sum to at most 1. By Lemma 10, the created tour is within the distance constraint D. Therefore, S is a feasible solution to the tree DVRP.

By construction, each tour in S visits terminals from only one component. Therefore, the output of Algorithm 1 is at most $|S|$, so is at most $|B_2|$. It remains to analyze $|B_2|$.

▶ **Lemma 15.** $|B_2| \leq |B_1| + (30/\Gamma)M \cdot \mathrm{opt}$.

Proof. Let $J \subseteq [1, |L| - 1]$ denote the set of integers $j \leq |L| - 1$ such that the j^{th} element in L and the $(j+1)^{\text{th}}$ element in L are the reduced lengths of two subtours in different components or of different categories. From the construction of L and since there are m components and two categories, we have

$$|J| \leq 2m. \tag{5}$$

Consider a bin $b \in B_1$. The number of bins in B_2 generated by b is the number of pairs (c, x) where $c \in \mathcal{C}$ and $x \in \{\text{passing, ending}\}$, such that b contains a reduced length of a subtour in component c and of category x. Let $\min(b)$ (resp. $\max(b)$) denote the minimum (resp. maximum) integer $j \in [1, |L|]$ such that the j^{th} element in L belongs to b. Let p_b denote the number of elements $j \in J$ such that $j \in [\min(b), \max(b)]$. The number of bins in B_2 generated by b is at most $1 + p_b$. Summing over all bins of B_1 gives

$$|B_2| \leq \sum_{b \in B_1} (1 + p_b) = |B_1| + \sum_{b \in B_1} p_b. \tag{6}$$

Observe that

$$\sum_{b \in B_1} p_b \leq \sum_{j \in J} \#\big(\text{bins } b \in B_1 \text{ such that } j \in [\min(b), \max(b)]\big).$$

For each $j \in J$, if a bin $b \in B_1$ is such that $j \in [\min(b), \max(b)]$, then b is *open* at the time of j. Since B_1 is a solution to the bounded space online bin packing, the number of open bins at the time of j is at most M, thus the number of bins $b \in B_1$ such that $j \in [\min(b), \max(b)]$ is at most M. Therefore,

$$\sum_{b \in B_1} p_b \leq |J| \cdot M \leq 2m \cdot M \leq (30/\Gamma)M \cdot \text{opt}, \tag{7}$$

where the second inequality follows from Equation (5) and the last inequality follows from Lemma 11.

The claim follows from Equations (6) and (7). ◀

Combining Lemma 15, Equation (4) and using $\Gamma = 1/\varepsilon^2$, we have

$$|B_2| \leq \beta_M \cdot \text{opt}_{\text{BP}} + \gamma_M + 30\varepsilon^2 M \cdot \text{opt}.$$

Next, we show that opt_{BP} is at most opt. For each tour t in an optimal solution OPT to the tree DVRP, if t visits components $c_1, ..., c_k$ with subtours' reduced lengths $\bar{\ell}_1, \bar{\ell}_2, ..., \bar{\ell}_k$, then using Lemma 9 we have $\bar{\ell}_1 + \bar{\ell}_2 + ... + \bar{\ell}_k \leq 1$. So it is possible to put the reduced lengths $\bar{\ell}_1, \bar{\ell}_2, ..., \bar{\ell}_k$ in a single bin of size 1. This leads to a feasible solution to the bin packing in the offline setting that uses opt bins. Thus $\text{opt}_{\text{BP}} \leq \text{opt}$. Hence

$$|B_2| \leq (\beta_M + 30\varepsilon^2 M) \cdot \text{opt} + \gamma_M.$$

This completes the proof of Theorem 14.

4.3 Proof of Theorem 2 Using the Reduction (Theorem 14)

First, consider the case when $\text{opt} < 1/\varepsilon^2$. Since $\Gamma = 1/\varepsilon^2$, Algorithm 2 returns a single component, let it be c. Then Algorithm 3 computes an optimal solution in c, thus the solution returned by Algorithm 1 is optimal.

Next, consider the case when $\text{opt} \geq 1/\varepsilon^2$. Let $M = 1/\varepsilon$. Let k be defined in Theorem 13. By Theorem 13, there exists a solution to the bounded space online bin-packing problem using at most $\beta_M \cdot \text{opt}_{\text{BP}} + \gamma_M$ tours, where

$$\beta_M := \left(\sum_{i=1}^{k} \frac{1}{u_i} + \frac{M}{(M-1)u_{k+1}} \right) \leq \alpha + 2\varepsilon, \text{ since } u_{k+1} \geq M,$$

$$\gamma_M := M - 1 < 1/\varepsilon.$$

Thus by the reduction (Theorem 14), the number of tours used by Algorithm 1 is at most

$$(\beta_M + 30\varepsilon^2 \cdot M) \cdot \text{opt} + \gamma_M < (\alpha + 2\varepsilon + 30\varepsilon) \cdot \text{opt} + 1/\varepsilon \leq (\alpha + 33\varepsilon) \cdot \text{opt},$$

where the last inequality follows since $\text{opt} \geq 1/\varepsilon^2$.

By Lemmas 5 and 7, the running time of Algorithm 1 is $O\left(n^2 \cdot \left(D/\varepsilon^2\right)^{O(1/\varepsilon^2)}\right)$.

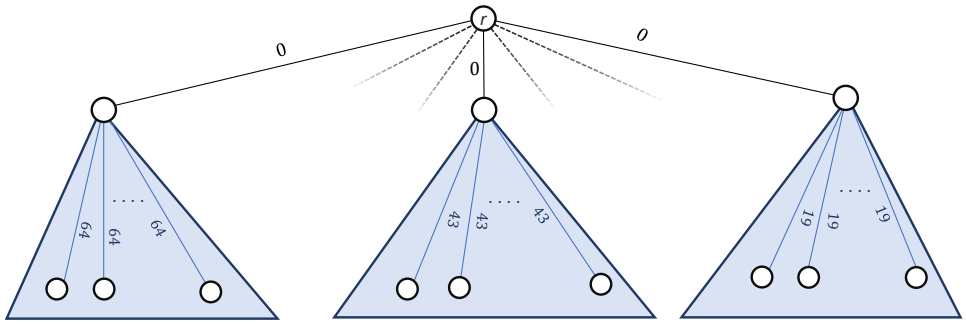


Figure 3 Instance $\mathcal{I}_k$ for $k = 3$. From Definition 1, $u_1 = 1$, $u_2 = 2$, $u_3 = 6$, $u_4 = 42$. The distance constraint $D = 2 \cdot k \cdot u_{k+1} = 252$. In a type-1 component, the edge weight $x_1 = 64$. In a type-2 component, the edge weight $x_2 = 43$. In a type-3 component, the edge weight $x_3 = 19$. Consider a tour t that visits a terminal in a type-1 component, a terminal in a type-2 component, and a terminal in a type-3 component. The length of t is $2(x_1 + x_2 + x_3) = D$.

5 Proof of Theorem 3

For each positive integer k, we construct an instance $\mathcal{I}_k$ of the tree DVRP as follows.

The tree in the instance $\mathcal{I}_k$ consists of a root vertex r and a set of components. The root of each component is connected to r by an edge of weight 0. The components are of k *types*. For each $i \in [1, k]$, a type-i component consists of $1 + \Gamma \cdot u_i$ vertices: One vertex is the root of the component, and the remaining $\Gamma \cdot u_i$ vertices are the terminals, each connected to the root of the component by an edge of weight $x_i := k \cdot u_{k+1}/(u_i + 1) + 1$. There are u_k/u_i components of type i for each $i \in [1, k]$. See Figure 3. Observe that $u_{k+1}/(u_i + 1)$ and u_k/u_i are both integers according to Definition 1. We set the distance constraint $D := 2k \cdot u_{k+1}$.

We claim that there exists a feasible solution to $\mathcal{I}_k$ using $\Gamma \cdot u_k$ tours. Consider a set of tours S such that each tour $t \in S$ visits exactly k terminals: for each $i \in [1, k]$, tour t visits exactly one terminal from all components of type i. For any $i \in [1, k]$, there are $\Gamma \cdot u_i \cdot u_k/u_i = \Gamma \cdot u_k$ terminals in all components of type i. Thus S consists of $\Gamma \cdot u_k$ tours. Next, we show that each tour $t \in S$ is within the distance constraint D. We have

$$\text{length}(t) = \sum_{i=1}^{k} 2x_i = 2k \cdot u_{k+1} \sum_{i=1}^{k} \frac{1}{u_i + 1} + 2k. \tag{8}$$

Using Definition 1, it is easy to show the following fact by induction.

▶ **Fact 16.** *For any positive integer k, we have*

$$\frac{1}{u_{k+1}} = 1 - \sum_{i=1}^{k} \frac{1}{u_i + 1}.$$

From Equation (8) and Fact 16, we have

$$\text{length}(t) = 2k \cdot u_{k+1} \left(1 - \frac{1}{u_{k+1}}\right) + 2k = D.$$

Therefore, S is a feasible solution. Hence

$$\text{opt} \leq \Gamma \cdot u_k. \tag{9}$$

Next, we analyze the solution to $\mathcal{I}_k$ computed by Algorithm 1. Recall that Algorithm 1 computes an optimal solution in each component independently. Let c be any component. Let $i \in [1, k]$ be the type of c. We observe that a tour is able to cover u_i terminals in c but is unable to cover $u_i + 1$ terminals in c. This is because, the cost to cover u_i terminals in c is

$$u_i \cdot 2x_i = D - \frac{2k \cdot u_{k+1}}{u_i + 1} + 2u_i \leq D,$$

where the inequality follows from Definition 1, and the cost to cover $u_i + 1$ terminals in c is

$$(u_i + 1) \cdot 2x_i = D + 2(u_i + 1) > D.$$

Since there are $\Gamma \cdot u_i$ terminals in c, the minimum number of tours to cover the terminals in c is Γ.

For each $i \in [1, k]$, the number of components of type i is u_k/u_i. Thus the number of tours returned by Algorithm 1 is $\sum_{i=1}^{k}(u_k/u_i) \cdot \Gamma = \Gamma \cdot u_k \sum_{i=1}^{k} 1/u_i$.

Combined with Equation (9), we conclude that the approximation ratio of Algorithm 1 on $\mathcal{I}_k$ is at least $\sum_{i=1}^{k} 1/u_i$, which tends to α when k tends to ∞ by Definition 1.

This completes the proof of Theorem 3.

References

1. Arjang A. Assad. Modeling and implementation issues in vehicle routing. In *Vehicle routing: Methods and studies*, pages 7–45, 1988.
2. Sean Barker, Timothy Wood, Prashant Shenoy, and Ramesh Sitaraman. An empirical study of memory sharing in virtual machines. In *USENIX Annual Technical Conference*, pages 273–284, 2012.
3. Amariah Becker and Alice Paul. A framework for vehicle routing approximation schemes in trees. In *Workshop on Algorithms and Data Structures*, pages 112–125. Springer, 2019.
4. Teodor G. Crainic and Gilbert Laporte. *Fleet management and logistics*. Springer Science & Business Media, 2012.
5. Zachary Friggstad and Chaitanya Swamy. Approximation algorithms for regret-bounded vehicle routing and applications to distance-constrained vehicle routing. In *Proceedings of the forty-sixth annual ACM Symposium on Theory of Computing (STOC)*, pages 744–753, 2014.
6. Michael R. Garey and David S. Johnson. *Computers and Intractability: a guide to the theory of NP-Completeness*. Freeman, New York, 1985.
7. Bruce Golden, S. Raghavan, and Edward Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*. Springer, 2008.
8. Aditya Jayaprakash and Mohammad R. Salavatipour. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 877–893, 2022.
9. Gilbert Laporte, Martin Desrochers, and Yves Nobert. Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172, 1984.
10. Chan C. Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.
11. Chung-Lun Li, David Simchi-Levi, and Martin Desrochers. On the distance constrained vehicle routing problem. *Operations Research*, 40(4):790–799, 1992.

12 Claire Mathieu and Hang Zhou. A PTAS for capacitated vehicle routing on trees. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 95:1–95:20, 2022.

13 Viswanath Nagarajan and R Ravi. Approximation algorithms for distance constrained vehicle routing problems. *Networks*, 59(2):209–214, 2012.

14 Safraz Rampersaud and Daniel Grosu. Sharing-aware online virtual machine packing in heterogeneous resource clouds. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):2046–2059, 2016.

15 Michael Sindelar, Ramesh K. Sitaraman, and Prashant Shenoy. Sharing-aware algorithms for virtual machine colocation. In *Proceedings of the twenty-third annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 367–378, 2011.

16 Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. SIAM, 2002.

A NP-Hardness

▶ **Lemma 17.** *The Tree DVRP is strongly NP-hard.*

Proof. We reduce the bin packing problem to the tree DVRP. Consider an instance of the bin packing problem with an integer bin capacity M and n items of sizes $a_1, \ldots, a_n$, where $a_i \in [0, M]$ for each $i \in [1, n]$. The bin packing problem looks for a partition of the n items into the minimum number of bins such that in each bin, the sum of the item sizes is at most M. We construct an instance of the tree DVRP as follows. There is a depot and n terminals. For each $i \in [1, n]$, there is an edge between the depot and the i^{th} terminal with weight a_i. Let $D = 2M$. It is easy to see that a solution to the bin packing instance is equivalent to a soution to the tree DVRP instance. Since the bin packing problem is strongly NP-hard [6], the tree DVRP is strongly NP-hard. ◀

B Component Decomposition Algorithm

For each vertex v of the tree, let $T(v)$ denote the subtree rooted at v. For any subgraph H of the tree, let U_H denote the set of terminals in U that belong to the subgraph H.

The algorithm is given in Algorithm 2.

Algorithm 2 Decomposition algorithm $\textsc{Decompose}^{(\Gamma)}$ parameterized by Γ (see Lemma 5).

Input A tree T rooted at r, a distance constraint D
Output A decomposition of T into components

1: {Leaf components} := $\{T(v) : v$ least deep vertex s.t. $\textsc{Solve}^{(\Gamma)}(T(v), D - 2 \cdot \text{dist}(r, v), U_{T(v)}) \leq \Gamma$
2: $T' \leftarrow$ subtree spanning $\{r\} \cup \{$roots of leaf components$\}$
3: **for** each maximal downward v_1-to-v_2 path in T' whose internal vertices have only one child in T' **do**
4: Let v_1' be the child of v_1 on the v_1-to-v_2 path.
5: **while** $\textsc{Solve}^{(\Gamma)}(T(v) \setminus T(v_2), D - 2 \cdot \text{dist}(r, v), U_{T(v) \setminus T(v_2)}) \leq \Gamma$ **do**
6: $v \leftarrow$ least deep vertex on the v_1'-to-v_2 path such that
7: $\textsc{Solve}^{(\Gamma)}(T(v) \setminus T(v_2), D - 2 \cdot \text{dist}(r, v), U_{T(v) \setminus T(v_2)}) \leq \Gamma$
8: Define internal component $(T(v) \setminus T(v_2))$ with exit vertex v_2
9: $v_2 \leftarrow v$
10: Define internal component $(v_1, v_1') \cup (T(v_1') \setminus T(v_2))$ with exit vertex v_2

Running time

The number of calls on $\textsc{Solve}^{(\Gamma)}$ is $O(n)$. Each call takes time $O(n \cdot (2\Gamma)^{\Gamma} \cdot D^{3\Gamma})$ by Lemma 7. Thus the overall running time of Algorithm 2 is $O(n^2 \cdot (2\Gamma)^{\Gamma} \cdot D^{3\Gamma})$.

C Proof of Theorem 7

Let $\tilde{T} = (\tilde{V}, \tilde{E})$ be a tree with root $\tilde{r}$. Let $\tilde{n}$ denote the number of vertices in $\tilde{T}$. Let $\tilde{U} \subseteq \tilde{V}$ be the set of terminals. The goal is to cover the terminals in $\tilde{U}$ using a minimum number of tours of length at most $\tilde{D}$ each.

Let Γ be a positive integer. We design a dynamic program that computes an optimal solution of at most Γ tours if such a solution exists. See Algorithm 3.

Algorithm 3 Dynamic program $\textsc{Solve}^{(\Gamma)}$ parameterized by Γ (see Lemma 7).

Input A tree $\tilde{T}$ rooted at $\tilde{r}$, a set of terminals $\tilde{U}$, a distance constraint $\tilde{D}$
Output $\min\{|S| : S \text{ is a feasible set of tours and } |S| \leq \Gamma\}$

Preprocess the tree $\tilde{T}$ so that each leaf is a terminal and each non-leaf vertex of $\tilde{T}$ has exactly two children ▷ Section 2.1
for each configuration (v, A) **do**
    $valid(v, A) = false$
for each terminal v in $\tilde{T}$ **do** ▷ Case 1
    for each list A such that $\ell(A) \in [1, \Gamma]$ and every element in A equals 0 **do**
        $valid(v, A) = true$
    for each non-terminal v of $\tilde{T}$ in bottom-up order **do** ▷ Case 2
        Let v_1 and v_2 denote the two children of v
        for each lists A, A_1, A_2 such that (v_1, A_1) and (v_2, A_2) are valid configurations **do**
            if (v_1, A_1), (v_2, A_2), and (v, A) are *compatible* **then** ▷ Definition 19
                $valid(v, A) \leftarrow true$
return $\min\{\ell(A) : (\tilde{r}, A) \text{ is valid}\}$

To begin with, using the preprocessing step in Section 2.1, we transform the tree $\tilde{T}$ so that the terminals are the same as the leaves in $\tilde{T}$ and every non-leaf vertex in $\tilde{T}$ has exactly two children.

We compute values at *configurations* (Definition 18), which are solutions restricted to a subtree of $\tilde{T}$.

▶ **Definition 18** (configurations). *A* configuration (v, A) *is defined by a vertex* $v \in \tilde{T}$ *and a list* A *of* $\ell(A)$ *integers* $(s_1, s_2, \ldots, s_{\ell(A)})$ *such that*

- $\ell(A) \leq \Gamma$*;*
- *for each* $i \in [1, \ell(A)]$*,* s_i *is an integer in* $[0, \tilde{D}]$*.*

We say that a configuration (v, A) is *valid* if it is possible to cover the terminals in the subtree of $\tilde{T}$ rooted at v with a collection of $\ell(A)$ subtours, such that each subtour starts and ends at v and the i-th subtour has length s_i. Note that $\ell(A)$ is equal to the total number of subtours in the collection. Thus the objective is to find a valid configuration $(\tilde{r}, A)$ with $\ell(A)$ minimum.

Let v be any vertex in $\tilde{T}$. We decide whether the configuration (v, A) is valid according to one of the two cases.

Case 1: v is a leaf vertex in $\tilde{T}$

Then v is a terminal. The configuration (v, A) is *valid* if and only if $\ell(A) \in [1, \Gamma]$ and every element in the list A equals 0.

Case 2: v is a non-leaf vertex in $\tilde{T}$

Let v_1 and v_2 be the two children of v in $\tilde{T}$.

▶ **Definition 19** (compatibility). *Let w_1 (resp. w_2) denote the weight of the edge between v and v_1 (resp. v_2). We say that the configurations (v_1, A_1), (v_2, A_2), and (v, A) are* compatible *if there is a partition $\mathcal{P}$ of $A_1 \cup A_2$ into parts, each part consisting of one or two elements such that at most one element is from A_1 (resp. A_2), and a one-to-one correspondence between every part in $\mathcal{P}$ and every element in A such that:*

- *a part in $\mathcal{P}$ consisting of one element $s^{(1)} \in A_1$ corresponds to an element s in A if and only if $s^{(1)} + 2w_1 = s$;*
- *a part in $\mathcal{P}$ consisting of one element $s^{(2)} \in A_2$ corresponds to an element s in A if and only if $s^{(2)} + 2w_2 = s$;*
- *a part in $\mathcal{P}$ consisting of two elements $s^{(1)}$ and $s^{(2)}$ corresponds to an element s in A if and only if $s = s^{(1)} + 2w_1 + s^{(2)} + 2w_2$.*

The configuration (v, A) is *valid* if and only if there exist a valid configuration (v_1, A_1) and a valid configuration (v_2, A_2) that are compatible with (v, A).

Running time

For each vertex $v \in \tilde{T}$, since $\ell(A) \leq \Gamma$, the number of configurations (v, A) is at most $\tilde{n} \cdot \tilde{D}^{\Gamma}$. For fixed $(v_1, A_1), (v_2, A_2)$, and (v, A), to check compatibility, there are at most $(2\Gamma)^{\Gamma}$ partitions of $A_1 \cup A_2$ into parts. Thus the overall running time of Lemma 7 is $O(\tilde{n} \cdot (2\Gamma)^{\Gamma} \cdot \tilde{D}^{3\Gamma})$.

Representation of Short Distances in Structurally Sparse Graphs

Zdeněk Dvořák
Computer Science Institute, Charles University, Prague, Czech Republic

Abstract

A partial orientation $\vec{H}$ of a graph G is a *weak r-guidance system* if for any two vertices at distance at most r in G, there exists a shortest path P between them such that $\vec{H}$ directs all but one edge in P towards this edge. In case that $\vec{H}$ has bounded maximum outdegree Δ, this gives an efficient representation of shortest paths of length at most r in G: For any pair of vertices, we can either determine the distance between them or decide the distance is more than r, and in the former case, find a shortest path between them, in time $O(\Delta^r)$. We show that graphs from many natural graph classes admit such weak guidance systems, and study the algorithmic aspects of this notion. We also apply the notion to obtain approximation algorithms for distance variants of the independence and domination number in graph classes that admit weak guidance systems of bounded maximum outdegree.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases distances, structurally sparse graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.28

Related Version *Full Version*: `https://arxiv.org/abs/2204.09113`

Funding *Zdeněk Dvořák*: Supported by the ERC-CZ project LL2005 (Algorithms and complexity within and beyond bounded expansion) of the Ministry of Education of Czech Republic. Revised and extended with support of the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 810115).

1 Introduction

We consider the following general question: Given an undirected unweighted graph G, can short distances in G be represented efficiently? More precisely, the setting that interests us is as follows:

- G is known to belong to some class $\mathcal{G}$ of well-structured graphs (e.g., planar graphs, graphs of clique-width at most 6, ...).
- We are only interested in distances up to some fixed upper bound r.
- We are allowed to preprocess G in polynomial time; let D denote the resulting data structure.
- The data structure D should enable us to efficiently answer the queries of the following form:
  - Are two input vertices u and v at distance at most r in G?

  In case that the answer is positive, we may also want to determine the distance between u and v, and return a shortest path between them.

Note that we consider both $\mathcal{G}$ and r to be fixed parameters. There are several criteria to consider:

- The time complexity of the preprocessing.
- The time complexity of the queries.
- The space complexity (the size of D).


© Zdeněk Dvořák;
licensed under Creative Commons License CC-BY 4.0


Of course, there are some trade-offs between these criteria. E.g., D could store distances between all pairs of vertices, resulting in a relatively slow preprocessing time and space complexity $\Theta(|V(G)|^2)$, but constant query time. In this paper we consider a solution which still achieves constant query time (depending only on $\mathcal{G}$ and r), but is memory efficient in the sense that storing D takes up about as much space as the graph G itself. To achieve this, D will only consist of an orientation of G.

An *orientation* of an undirected graph G is a directed graph $\vec{H}$ such that for every $(u,v) \in E(\vec{H})$, we have $uv \in E(G)$, and for every $uv \in E(G)$, at least one of (u,v) and (v,u) is a directed edge of $\vec{H}$. Note that $\vec{H}$ can contain both (u,v) and (v,u), i.e., we allow an edge of G to be directed in both ways at the same time. Let $B_{\vec{H}}(v,a)$ denote the set of vertices reachable in $\vec{H}$ from v by a directed path of length at most a. An *r-guidance system* is an orientation $\vec{H}$ such that for any vertices $u, v \in V(G)$ at distance $\ell \leq r$ in G, there exist non-negative integers a and b such that $a + b = \ell$ and $B_{\vec{H}}(u,a) \cap B_{\vec{H}}(v,b) \neq \emptyset$; i.e., there is a shortest path between u and v in G whose edges are directed in $\vec{H}$ towards one of its vertices. Note that if $\vec{H}$ has maximum outdegree at most c, all such paths can be enumerated in time $O(c^r)$, and if c is small, this enables us to find a shortest path between a given pair of vertices (or verify that their distance is greater than r) efficiently.

The guidance systems were (without explicitly naming them) introduced by Kowalik and Kurowski [11], who proved that they can be used to represent short distances in planar graphs, and more generally for every F, in any graph avoiding F as a topological minor. As observed in [7], essentially the same argument shows that graphs from even more general graph classes, namely all classes with *bounded expansion* and more generally all *nowhere-dense* classes, admit guidance systems of bounded maximum outdegree. To state the result precisely, we need to introduce several definitions.

For a non-negative integer s, a graph H is an *s-shallow minor* of a graph G if H is obtained from a subgraph of G by contracting pairwise-disjoint subgraphs, each of radius at most s. For a class $\mathcal{G}$, let $\nabla_s\mathcal{G}$ denote the class of all graphs H that appear as s-shallow minors in graphs from $\mathcal{G}$. A class $\mathcal{G}$ of graphs has *bounded expansion* if for every $s \geq 0$ there exists d_s such that every graph in $\nabla_s\mathcal{G}$ has average degree at most d_s. Even less restrictively, a class $\mathcal{G}$ is *nowhere-dense* if for every $s \geq 0$ there exists d_s such that $K_{d_s} \notin \nabla_s\mathcal{G}$. Examples of classes of graphs with bounded expansion include planar graphs and more generally all proper minor-closed classes, graphs with bounded maximum degree and more generally all proper classes closed under topological minors, graphs drawn in the plane with $O(1)$ crossings on each edge, and many other classes of sparse graphs; see [13] for more details.

▶ **Theorem 1** (Dvořák and Lahiri [7]). *Let $\mathcal{G}$ be a class of graphs and r a positive integer.*

- *If $\mathcal{G}$ has bounded expansion, then there exists c such that every graph $G \in \mathcal{G}$ has an r-guidance system of maximum outdegree at most c. Moreover, such an r-guidance system can be found in time $O(|V(G)|)$.*
- *If $\mathcal{G}$ is nowhere-dense, then for every $\varepsilon > 0$, there exists c such that every graph $G \in \mathcal{G}$ has an r-guidance system of maximum outdegree at most $c|V(G)|^\varepsilon$. Moreover, such an r-guidance system can be found in time $O(|V(G)|^{1+\varepsilon})$.*

A graph with an orientation of maximum outdegree at most c necessarily has maximum average degree at most $2c$, and thus it is $(2c+1)$-degenerate. Hence, guidance systems of bounded maximum outdegree can only exist in sparse graphs. This brings us to the main topic of our paper: **Does there exist a variant of the notion useful for dense graphs?**

Note that representing distance one by a guidance system forces us to orient all edges. If we relax the notion to only represent distances 2, 3, ..., r, this may not be necessary. A *partial orientation* of a graph G is a spanning directed subgraph of an orientation of G

(i.e., we allow some edges not to be oriented in either direction). An r^+-*guidance system* is a partial orientation $\vec{H}$ of a graph G such that for any vertices $u, v \in V(G)$ at distance ℓ in G, where $2 \leq \ell \leq r$, there exist non-negative integers a and b such that $a + b = \ell$ and $B_{\vec{H}}(u, a) \cap B_{\vec{H}}(v, b) \neq \emptyset$. Let us give a (trivial) example showing that there are dense graphs admitting r^+-guidance systems.

▶ **Example 2.** Let G be a graph containing a universal vertex u, and let $\vec{H}$ be the partial orientation obtained by directing all edges incident with u towards u. Observe that for any positive integer r, $\vec{H}$ is an r^+-guidance system in G of maximum outdegree one.

However, there are some quite simple graphs that do not admit r^+-guidance systems of bounded outdegree. For a graph G and a positive integer k, let G^k denote the *k-distance power* of G, that is, the graph with vertex set $V(G)$ and two vertices adjacent if and only if the distance between them in G is at most k.

▶ **Example 3.** Let T be the graph obtained from $K_{1,n}$ by subdividing every edge exactly twice, let X be the set of its leaves, and let Y be the set of neighbors of the central vertex of degree n. Let $G = T^2$. Note that Y induces a clique in G, and any two vertices of X are joined by a unique path of length three using exactly one edge of this clique. This implies that in any 3^+-guidance system for G, every edge of the clique on Y must be directed in at least one direction, and thus some vertex of Y has outdegree at least $(n-1)/2$.

This example highlights the fact that in dense graphs, we cannot afford to represent the shortest paths by having all of their edges oriented. This motivates us to generalize the guidance systems as follows, introducing the main notion of interest for this paper.

▶ **Definition 4.** *A* weak r-guidance system *is a partial orientation $\vec{H}$ of G such that for any distinct vertices $u, v \in V(G)$ at distance $\ell \leq r$ in G, there exist non-negative integers a and b such that $a + b = \ell - 1$ and G contains an edge between $B_{\vec{H}}(u, a)$ and $B_{\vec{H}}(v, b)$; that is, there exists a shortest path between u and v in G such that all but one edge e of this path is directed in $\vec{H}$ towards this exceptional edge e (which may or may not be directed).*

In particular, an r-guidance system (or an r^+-guidance system) is also a weak r-guidance system. Note that if the graph G is represented so that we can in constant time test whether two vertices are adjacent, then a weak r-guidance system of maximum outdegree c makes it possible to find a shortest path between a given pair of vertices (or verify that their distance is greater than r) in time $O(c^{r-1})$.

The goal of this paper is to develop the theory of weak guidance systems; we show that several interesting graph classes admit weak guidance systems of small maximum outdegree (constant, or logarithmic in the number of vertices), address the algorithmic question of finding weak guidance systems efficiently, and on the negative side, we give examples of simple graph classes that do not admit weak guidance systems of small maximum outdegree.

Guidance systems and related notions such as weak coloring numbers have many applications in algorithmic design, for example in efficient practical algorithms for determining statistics of small subgraphs [16] and design of approximation algorithms [4, 6, 7]. We expect weak guidance systems to be similarly useful; to illustrate this, we describe an application in approximation of distance variants of the independence and domination number, generalizing the results of [4].

1.1 Summary of our results

- We start by describing basic properties of weak guidance systems, including the fact that they behave well under the distance power operation considered in Example 3.

- A standard way of generalizing results for sparse graphs (e.g., for classes with bounded expansion) is to consider graphs definable in them by first-order logic formulas. We show that this approach works for weak guidance systems in Theorem 8, which generalizes Theorem 1 to the dense setting in this way.
- The aforementioned results guaranteeing the existence of weak guidance systems of bounded maximum outdegree do not provide polynomial-time algorithms to find such a weak guidance system. In Corollary 16, we provide an approximation algorithm for this problem that for an n-vertex graph which admits a weak guidance system of maximum outdegree c returns one of maximum outdegree $O(c \log n)$.
- In Theorem 19, we improve this bound to $O(c \log c)$ under an additional assumption that certain set systems have bounded VC-dimension. This in particular gives an efficient algorithmic version of Theorem 8 (Corollary 21).
- On the negative side, in Section 2.3 we show that several natural graph classes do not admit weak guidance systems of bounded maximum outdegree, specifically graphs of girth at least five and large average degree, split graphs, and graphs of bounded clique-width.
- In Section 3, we show an application of weak guidance systems in design of approximation algorithms for distance independence and domination number (under an additional assumption that the considered graph class is stable, which we show to be necessary). In particular, this gives a constant-factor approximation algorithm for any graphs that are first-order definable in classes with bounded expansion.

2 Theory of weak guidance systems

In this section, we describe the theoretical results on weak guidance systems in detail. Some of the proofs are defered to the Appendix; the claims marked with (†) have simple proofs which we do not present due to space constraints. Before we start, let us note that weak guidance systems enable us to circumvent the difficulty from Example 3.

▶ **Lemma 5** (†). *Let G be a graph and let $k \geq 1$ and $c \geq 2$ be integers. For any positive integer r, if G has a weak kr-guidance system $\vec{H}$ of maximum outdegree at most c, then G^k has a weak r-guidance system $\vec{F}$ of maximum outdegree at most $2c^k$.*

Weak guidance systems are qualitatively different from guidance systems only in dense graphs, as in degenerate graphs, a weak guidance system can be completed to a guidance system by directing the rest of the edges while preserving the bounded maximum outdegree.

▶ **Observation 6.** *If G admits a weak r-guidance system of maximum outdegree c and G is t-degenerate, then G also admits an r-guidance system of maximum outdegree at most $c + t$.*

Finally, we give the following description of weak r-guidance systems, which we use often in the rest of the paper. For vertices u and v of a graph G at distance ℓ, let $G(u \to v)$ be the set of neighbors of u at distance $\ell - 1$ from v; i.e., $G(u \to v)$ consists of all possible second vertices of shortest paths from u to v.

▶ **Observation 7.** *A partial orientation $\vec{H}$ of a graph G is a weak r-guidance system if and only if the following claim holds for all $u, v \in V(G)$ at distance ℓ in G, where $2 \leq \ell \leq r$:*
($\star$) Either u has an outneighbor in $G(u \to v)$, or v has an outneighbor in $G(v \to u)$.

2.1 Weak guidance systems in structurally sparse graphs

The standard way of generalizing the concepts of bounded expansion and nowhere-density to dense graphs is through the notion of *first-order transductions*, see e.g. [8, 9, 2, 12]. For a positive integer k and a graph G, let kG denote the disjoint union of k copies of G. A *transduction* T consists of

- a positive integer k
- a binary predicate symbol M and unary predicate symbols $U_1, \ldots, U_s$, and
- first-order formulas $\omega(x)$ and $\epsilon(x,y)$ with free variables x (resp. x and y) using these predicate symbols and the binary predicate symbol E.

For graphs H and G, we write $H \in T(G)$ if there exist sets $C_1, \ldots, C_s \subseteq V(kG)$ such that $V(H)$ consists exactly of the vertices $v \in V(kG)$ satisfying

$$kG, U_1 := C_1, \ldots, U_s := C_s \models \omega(v)$$

and $E(H)$ consists exactly of the pairs $u, v \in V(H)$ such that

$$kG, U_1 := C_1, \ldots, U_s := C_s \models \epsilon(u,v),$$

where the predicate symbol E is interpreted as adjacency in kG and M is interpreted as the equivalence between the k copies of each vertex.

That is, a transduction allows us to blow up the graph by replicating each vertex a bounded number of times, then non-deterministically color some vertices (via the predicates $U_1, \ldots, U_s$), and finally define the vertices and edges of the new graph by a first-order formula. As an example, if T is the transduction with $k = 1$, $s = 0$, $\omega(x) = \text{true}$ and

$$\epsilon(x,y) = (x \neq y) \wedge (\exists z)(z = x \vee E(x,z)) \wedge E(z,y),$$

then $H \in T(G)$ if and only if $H = G^2$. Hence, the transduction operation generalizes the graph power operations we considered in Lemma 5.

For a class of graphs $\mathcal{G}'$ and a transduction T, let $T(\mathcal{G}')$ denote the class of all graphs G such that $G \in T(G')$ for some $G' \in \mathcal{G}'$. We say that a class of graphs $\mathcal{G}$ has *structurally bounded expansion* (resp., is *structurally nowhere-dense*) if $\mathcal{G} \subseteq T(\mathcal{G}')$ for a transduction T and a graph class $\mathcal{G}'$ of bounded expansion (resp., being nowhere-dense). As our first result, we show that weak guidance systems behave as one would expect for these standard generalizations of the notions of sparsity to dense graphs.

▶ **Theorem 8.** *Let $\mathcal{G}$ be a class of graphs and let r be a positive integer.*

- *If $\mathcal{G}$ has structurally bounded expansion, then for some positive integer c, every graph in $\mathcal{G}$ has a weak r-guidance system of maximum outdegree at most c.*
- *If $\mathcal{G}$ is structurally nowhere-dense and $\varepsilon > 0$, then for some positive integer c, every graph in $\mathcal{G}$ has a weak r-guidance system of maximum outdegree at most $c|V(G)|^{\varepsilon}$.*

In preparation for the proof of Theorem 8, let us consider the graph classes with bounded *shrub-depth*. The notion of shrub-depth was defined by Ganian et al. [10] using the concept of *tree models*. For a positive integer m, an *m-signature* is a function $S : \mathbb{Z}^+ \to 2^{[m]\times[m]}$ assigning a symmetric relation $S(i)$ to each $i > 0$. For a positive integer d, an *(m,d)-tree model* of a graph G is a triple (T, φ, S), where

- T is a rooted tree with leaf set $V(G)$ and such that the length of every root-leaf path is d,
- $\varphi : V(G) \to [m]$ assigns one of m labels to each leaf,
- S is an m-signature, and
- for every $u, v \in V(G)$, if $2i$ is the distance between u and v in T (i.e., if i is the distance from u and v to their nearest common ancestor in T), then $uv \in E(G)$ if and only if $(\varphi(u), \varphi(v)) \in S(i)$.

A class $\mathcal{G}$ of graphs has *shrub-depth at most d* if for some positive integer m, every graph in $\mathcal{G}$ has an (m,d)-model.

▶ **Lemma 9.** *For every class $\mathcal{G}$ of graphs of bounded shrub-depth and every positive integer r, there exists a positive integer c such that every graph from $\mathcal{G}$ has a weak r-guidance system of maximum outdegree at most c.*

Proof. Let m and d be positive integers such that every graph $G \in \mathcal{G}$ has an (m,d)-tree model (T,φ,S). Let $c = r^3 m^r (d+1)^{r^2} d$.

For a positive integer k, a *k-type* is a pair (f,g) of functions $f : [k] \to [m]$ and $g : [k]^2 \to \{0\} \cup [d]$. The *type* of a k-tuple $(v_1,\ldots,v_k)$ of vertices of G is the k-type (f,g) such that $f(i) = \varphi(v_i)$ for $i \in [k]$ and $g(i,j)$ is half of the distance between v_i and v_j in T. For each vertex $x \in V(T)$, each positive integer $k \leq r$, and each k-type t, if there exist a k-tuple $(v_1,\ldots,v_k)$ of leaves of T with ancestor x and of type t, fix such a k-tuple $Q(x,t) = (v_1,\ldots,v_k)$ arbitrarily and let $A(x,t) = \{v_1,\ldots,v_k\}$; otherwise, let $A(x,t) = \emptyset$. For each non-leaf vertex $y \in V(T)$, if y has more than r children x such that $A(x,t) \neq \emptyset$, then let $R(y,t)$ be a set of $r+1$ of them chosen arbitrarily; otherwise let $R(y,t)$ be the set of all children x of y such that $A(x,t) \neq \emptyset$. Let $B(y,t) = \bigcup_{x \in R(y,t)} A(x,t)$, and let $B(y)$ be the union of $B(y,t)$ over all k-types t with $k \leq r$.

Let $\vec{H}$ be the partial orientation of G containing exactly the edges (u,v) such that $uv \in E(G)$ and $v \in B(y)$ for some ancestor y of u in T. Clearly, $\vec{H}$ has maximum outdegree at most c. Let us now argue that $\vec{H}$ is a weak r-guidance system.

Consider any vertices $u,v \in V(G)$ at distance ℓ in G, where $2 \leq \ell \leq r$, and let $P = u_0 u_1 \ldots u_\ell$, where $u_0 = u$ and $u_\ell = v$, be a shortest path from u to v in G. We will show that the condition $(\star)$ from Observation 7 is satisfied for u and v. Let y be the nearest common ancestor of u and u_1 in T, let X be the set of children of y that have a descendant belonging to $V(P)$, and let x_1 be the child of y whose descendant is u_1. Suppose first that v is not a descendant of x_1. Let Q be the tuple of vertices of P that are descendants of x_1 (in any order) and let t be its type. Since $|X| \leq r+1$ and $A(x_1,t) \neq \emptyset$, there exists $x_1' \in R(y,t) \setminus (X \setminus \{x_1\})$. Let $Q' = Q(x_1',t)$ and let P' be obtained from P by replacing the vertices of Q by the vertices of Q'. Observe that since Q and Q' have the same type and the same common ancestors with the other vertices of P, P' is also a shortest path from u to v in G. Moreover, the construction of $\vec{H}$ implies that the first edge of P' is directed away from u, establishing the validity of the condition $(\star)$ from Observation 7.

Hence, suppose that v is a descendant of x_1. In particular, this implies that y is also the nearest common ancestor of u and v. Let x_2 be the child of y whose descendant is u. By symmetry, we can assume that $u_{\ell-1}$ is a descendant of x_2 as well. Let $Q_1 = (u_1,u_2,\ldots,u_k)$ be the maximal initial segment of $P-u$ consisting of descendants of x_1; we have $k < \ell-1$. Let t_1 be the type of Q_1. Since $|X| \leq r+1$ and $A(x_1,t_1) \neq \emptyset$, there exists $x_1'' \in R(y,t_1) \setminus (X \setminus \{x_1\})$. Let $Q_1' = Q(x_1',t_1)$ and let P_1' be obtained from P by replacing the vertices of Q_1 by the vertices of Q_1'. Observe that since Q_1 and Q_1' have the same type and the same common ancestors with u and u_{k+1}, P_1' is also a shortest path from u to v in G. Moreover, the construction of $\vec{H}$ implies that the first edge of P_1' is directed away from u, establishing the validity of the condition $(\star)$ from Observation 7.

We conclude that $\vec{H}$ is a weak r-guidance system. ◀

Crucially, the notions of structurally bounded expansion and structural nowhere-density can be characterized in terms of *bounded shrub-depth covers*. A *cover* of a graph G is a system of subsets of $V(G)$. Let a be a positive integer. A cover $\mathcal{C}$ of G is *a-generic* if for every subset $A \subseteq V(G)$ of size at most a, there exists $C \in \mathcal{C}$ such that $A \subseteq C$. An *a-generic bounded shrub-depth cover assignment* for a graph class $\mathcal{G}$ is a function $\mathcal{C}$ that to each graph $G \in \mathcal{G}$ assigns an a-generic cover $\mathcal{C}(G)$ such that the class

$$\mathcal{C}(\mathcal{G}) = \{G[C] : G \in \mathcal{G}, C \in \mathcal{C}(G)\}$$

has bounded shrub-depth.

▶ **Theorem 10** (Gajarský et al. [9] and Dreier et al. [3]). *Let $\mathcal{G}$ be a class of graphs and let a be a positive integer.*

- *If $\mathcal{G}$ has structurally bounded expansion, then for some positive integer k, $\mathcal{G}$ has an a-generic bounded shrub-depth cover assignment $\mathcal{C}$ such that $|\mathcal{C}(G)| \leq k$ for every $G \in \mathcal{G}$.*
- *If $\mathcal{G}$ is structurally nowhere-dense and $\varepsilon > 0$, then for some positive integer k, $\mathcal{G}$ has an a-generic bounded shrub-depth cover assignment $\mathcal{C}$ such that $|\mathcal{C}(G)| \leq k|V(G)|^\varepsilon$ for every $G \in \mathcal{G}$.*

Together with Lemma 9, this gives the main result of this section.

Proof of Theorem 8. Let $\mathcal{C}$ be an $(r+1)$-generic bounded shrub-depth cover assignment and k the corresponding constant from Theorem 10. Let c_0 be the constant from Lemma 9 for the class $\mathcal{C}(\mathcal{G})$. Let $c = kc_0$.

For any graph $G \in \mathcal{G}$, let $\vec{H}$ be the union of the weak r-guidance systems of the subgraphs $G[C]$ for $C \in \mathcal{C}(G)$ obtained using Lemma 9. Clearly, the maximum outdegree of $\vec{H}$ is at most c if $\mathcal{G}$ has structurally bounded expansion and at most $c|V(G)|^\varepsilon$ if $\mathcal{G}$ is structurally nowhere-dense. Moreover, consider any vertices u and v at distance at most r in G, and let P be a shortest path between them. Since the cover $\mathcal{C}(G)$ is $(r+1)$-generic, there exists $C \in \mathcal{C}(G)$ such that $G[C]$ contains P. Since $\vec{H}$ restricted to C is a weak r-guidance system in $G[C]$, there exists a shortest path between u and v in $G[C]$ (and thus also in G) directed by $\vec{H}$ towards one of its edges. We conclude that $\vec{H}$ is a weak r-guidance system in G. ◀

Let us remark that r-guidance systems can be used to characterize bounded expansion and nowhere-density.

▶ **Lemma 11.** *Let $\mathcal{G}$ be a class of graphs closed under induced subgraphs.*

- *If there exists $c : \mathbb{Z}^+ \to \mathbb{Z}^+$ such that for every positive integer r, every $G \in \mathcal{G}$ has an r-guidance system of maximum outdegree at most $c(r)$, then $\mathcal{G}$ has bounded expansion.*
- *If there exists $c : \mathbb{Z}^+ \times \mathbb{R}^+ \to \mathbb{Z}^+$ such that for every positive integer r and for every $\varepsilon > 0$, every $G \in \mathcal{G}$ has an r-guidance system of maximum outdegree at most $c(r, \varepsilon)|V(G)|^\varepsilon$, then $\mathcal{G}$ is nowhere-dense.*

Note that the assumption of being closed under induced subgraphs is needed, as seen by Example 2: This example together with Observation 6 shows that the class of graphs formed from cliques by subdividing each edge once and adding a universal vertex afterwards admits an r-guidance system of maximum outdegree at most 4 for every $r \geq 1$; but this class is not nowhere-dense.

It is tempting to ask whether weak r-guidance systems similarly characterize structurally bounded expansion or structural nowhere-density. However, this is not the case. We define a *weak ∞-guidance system* to be a partial orientation that is a weak r-guidance system for every positive integer r. *Interval graphs* are the intersection graphs of sets of open intervals in the real line.

▶ **Example 12.** Consider any interval graph G. Let $\vec{H}$ be the partial orientation of G obtained as follows. For each $u \in V(G)$, let v_1 and v_2 be the neighbors of u such that the right endpoint of the interval of v_1 is maximum among all neighbors of u, and the left endpoint of the interval of v_2 is minimum among them. Include in $\vec{H}$ the edges (u, v_1) and (u, v_2). Then $\vec{H}$ is a weak ∞-guidance system in G of maximum outdegree at most two.

The class of interval graphs is closed under induced subgraphs, but it is well-known not to be structurally nowhere-dense.

2.2 Algorithmic aspects

The proof of Theorem 8 is based on the fact that structurally sparse graphs are known to admit bounded shrub-depth covers, see Theorem 10. However, it is currently not known how to obtain such covers efficiently. Consequently, Theorem 8 does not give an efficient algorithm to obtain weak guidance systems. A similar remark applies for Lemma 5, in case we are not given the weak guidance system $\vec{H}$ but only the graph G^k as the input.

In this section, we address this issue, giving a polynomial-time algorithm that given an n-vertex graph returns a weak guidance system whose maximum outdegree is worse than optimal only by an $O(\log n)$ factor, and an improved approximation algorithm in case certain relevant set systems have bounded VC-dimension.

First, let us introduce one more relaxation of the guidance system notion. A *fractional orientation* of a graph G is a function p that assigns a non-negative real number $p(u,v)$ to each pair (u,v) of adjacent vertices of G. The *outdegree* $d_p^+(u)$ of a vertex u in the fractional orientation p is $\sum_{v:uv\in E(G)} p(u,v)$. We say that p is a *fractional r-guidance system* if for every $u,v \in V(G)$ at distance ℓ, where $2 \leq \ell \leq r$, we have

$$\sum_{y\in G(u\to v)} p(u,y) + \sum_{y\in G(v\to u)} p(v,y) \geq 1. \tag{1}$$

By Observation 7, weak guidance systems can naturally be interpreted as fractional guidance systems.

▶ **Observation 13.** *Suppose $\vec{H}$ is a weak r-guidance system in a graph G, of maximum outdegree c. Let us define $p(u,v) = 1$ for every $(u,v) \in E(\vec{H})$ and $p(u,v) = 0$ for every $uv \in E(G)$ such that $(u,v) \notin E(\vec{H})$. Then p is a fractional r-guidance system of maximum outdegree c.*

Moreover, an optimal fractional guidance system can be constructed through linear programming.

▶ **Lemma 14** (†). *If a graph G has a weak r-guidance system of maximum outdegree c_0, we can find a fractional r-guidance system of maximum outdegree at most c_0 in G in polynomial time.*

A fractional r-guidance system p can be directly used to test presence of shortest paths, with a small probability of error, by constructing a pair of random walks between the two given vertices, with the probability distribution derived from p in the natural way; see the Appendix for details. More interestingly, we can turn a fractional r-guidance system to a weak r-guidance system with a logarithmic loss in the maximum degree. The basic idea is that a random selection of an outgoing edge at each vertex from the probability distribution given by p makes sure that in expectation the condition $(\star)$ from Observation 7 is satisfied by a constant fraction of pairs of vertices, and iterating such sampling $\Theta(\log n)$ times is sufficient for $(\star)$ to hold for all pairs, which shows that the resulting partial orientation is a weak r-guidance system.

▶ **Lemma 15.** *Let c be a positive real number, let n be a positive integer, and let $m = \lceil 4c \log n \rceil$. Suppose p is a fractional r-guidance system in a graph G, with maximum outdegree at most c. There exists an algorithm that in polynomial time returns a weak r-guidance system $\vec{H}$ in G with maximum outdegree at most m.*

Combining Lemmas 14 and 15, we obtain the following claim.

▶ **Corollary 16.** *There exists an algorithm that, for an input n-vertex graph G that admits a weak r-guidance system of maximum outdegree at most c, outputs in polynomial time a weak r-guidance system of maximum outdegree $O(c \log n)$.*

Let us remark that the logarithmic loss in Lemma 15 cannot be avoided in general. For positive integers a and k, let $m = k2^{k+1}$ and let $G_{a,k}$ be the random graph obtained as follows. We start with a random bipartite graph with parts L of size a and R of size ma, with each vertex of L being adjacent to each vertex of R independently with probability $1/2$. We then divide R into m parts $R_1, \ldots, R_m$ of size a arbitrarily, and for $i = 1, \ldots, m$, we add a vertex x_i adjacent to all vertices of R_i.

▶ **Lemma 17.** *There exists an integer a_0 such that for every $a \geq a_0$ and $k \leq \log a$, with positive probability*

- *$G_{a,k}$ has a fractional 2-guidance system with maximum outdegree at most 3, and*
- *$G_{a,k}$ does not have a weak 2-guidance system with maximum outdegree at most k.*

Note that if we set $k = \lfloor \log a \rfloor$, we have

$$n = |V(G_{a,k})| \leq (m+1)(a+1) \leq \left(k2^{k+1}+1\right) \cdot (\exp(k+1)+1) \leq \exp(O(k)),$$

and thus Lemma 17 gives examples of graphs with an arbitrarily large number n of vertices and a fractional 2-guidance system of maximum outdegree at most 3 such that every weak 2-guidance system has maximum outdegree $\Omega(\log n)$.

However, we can do better in case the VC-dimension of relevant systems is bounded. Recall that a system $\mathcal{S}$ of subsets of a set X *shatters* a set $A \subseteq X$ if $\{A \cap S : S \in \mathcal{S}\}$ contains all subsets of A, and that the *VC-dimension* of $\mathcal{S}$ is the size of the largest subset of X shattered by $\mathcal{S}$. For a graph G, integer $r \geq 2$, and vertex $u \in V(G)$, let $\mathrm{VC}(G, r, u)$ denote the VC-dimension of the system

$$\{G(u \to v) : v \in V(G), 2 \leq d_G(u, v) \leq r\},$$

and let $\mathrm{VC}(G, r) = \max_{u \in V(G)} \mathrm{VC}(G, r, u)$.

The key property of systems with bounded VC-dimension is that they admit efficient (randomized) approximation for smallest hitting set in terms of the size of the smallest fractional hitting set (a *hitting set* for $\mathcal{S}$ is a subset of X intersecting all elements of $\mathcal{S}$, and a *fractional hitting set* is a function $w : X \to \mathbb{R}_0^+$ such that, defining $w(A) = \sum_{x \in A} w(x)$ for each subset A of X, each element $S \in \mathcal{S}$ satisfies $w(S) \geq 1$; the size of the fractional hitting set w is $w(X)$). For the following standard result, see e.g. [14].

▶ **Theorem 18.** *There exists a polynomial-time randomized algorithm that, given a system $\mathcal{S}$ of subsets of a set X of VC-dimension at most d and a fractional hitting set w of size s, with probability at least $1/2$ returns a hitting set for $\mathcal{S}$ of size $O(ds \log s)$.*

We now apply this fact to give an improved algorithm to obtain weak guidance systems from fractional ones when $\mathrm{VC}(G, r)$ is bounded.

▶ **Theorem 19.** *There exists a polynomial-time randomized algorithm that, for an input n-vertex graph G that admits a weak r-guidance system of maximum outdegree at most c, with probability at least $1/2$ outputs a weak r-guidance system of maximum outdegree $O(\mathrm{VC}(G, r) \cdot c \log c)$.*

Proof. Let p be a fractional r-guidance system of maximum outdegree at most c in G found using Lemma 14. For each $u \in V(G)$, let R_u be the set of vertices $v \in V(G)$ such that $2 \leq d_G(u,v) \leq r$ and

$$\sum_{z \in G(u \to v)} p(u,z) \geq 1/2.$$

Since p is a fractional r-guidance system, for each $u, v \in V(G)$ such that $2 \leq d_G(u,v) \leq r$, we have $v \in R_u$ or $u \in R_v$.

Let $\mathcal{S}_u$ be the system $\{G(u \to v) : v \in R_u\}$ of subsets of the set $N_G(u)$ of neighbors of u. For $z \in N_G(u)$, let us define $w(z) = 2p(u,z)$. By the choice of R_u, we have $w(S) \geq 1$ for each $S \in \mathcal{S}_u$, and thus w is a fractional hitting set for $\mathcal{S}_u$. Moreover, $w(N_G(u)) \leq 2c$, since the maximum outdegree of p is at most c. The VC-dimension of $\mathcal{S}_u$ is at most $\mathrm{VC}(G,r,u) \leq \mathrm{VC}(G,r)$, and thus we can by Theorem 18 find a hitting set $H_u \subseteq N_G(u)$ for $\mathcal{S}_u$ of size $O(\mathrm{VC}(G,r) \cdot c \log c)$; note that we iterate the algorithm $\Omega(|V(G)|)$ times to make the probability of error less than $\frac{1}{2|V(G)|}$, and thus we find a valid hitting set for all $u \in V(G)$ with probability at least $1/2$.

Let us now define a partial orientation $\vec{G}$ of G by, for each $u \in V(G)$, directing the edges from u to H_u. Clearly, $\vec{G}$ has maximum outdegree $O(\mathrm{VC}(G,r) \cdot c \log c)$. Moreover, consider any $u, v \in V(G)$ such that $2 \leq d_G(u,v) \leq r$. By symmetry, we can assume that $v \in R_u$, and thus H_u intersects the set $G(u \to v) \in \mathcal{S}_u$. Hence, u has an outneighbor in $G(u \to v)$. By Observation 7, we conclude that $\vec{G}$ is a weak r-guidance system for G. ◀

In particular, this is useful for structurally nowhere-dense classes (and especially for classes with structurally bounded expansion), as follows from the fact that first-order definable sets in graphs from these classes have bounded VC-dimension [1, 15].

▶ **Lemma 20.** *For every structurally nowhere-dense class $\mathcal{G}$ of graphs and every integer $r \geq 2$, there exists a constant d such that $\mathrm{VC}(G,r) \leq d$ for every graph $G \in \mathcal{G}$.*

Hence, Theorem 19 gives the following algorithmic form of Theorem 8.

▶ **Corollary 21.** *Let $\mathcal{G}$ be a class of graphs and let r be a positive integer.*

- *If $\mathcal{G}$ has structurally bounded expansion, then there exists c and a randomized algorithm that for an input n-vertex graph $G \in \mathcal{G}$ outputs in polynomial time with probability at least $1/2$ a weak r-guidance system of maximum outdegree at most c.*
- *If $\mathcal{G}$ is structurally nowhere-dense and $\varepsilon > 0$, then there exists c and a randomized algorithm that for an input n-vertex graph $G \in \mathcal{G}$ outputs in polynomial time with probability at least $1/2$ a weak r-guidance system of maximum outdegree at most cn^{ε}.*

2.3 Graph classes without bounded outdegree weak guidance systems

To better understand obstructions to the existence of weak r-guidance systems of bounded maximum outdegree, it is natural to consider the dual of the linear program from the proof of Lemma 14, which can be reformulated as follows. For $uz \in E(G)$, let $R_r(u,z)$ be the set of vertices $v \in V(G)$ such that the distance between u and v is between 2 and r and z lies on a shortest path from u to v in G; i.e., $z \in G(u \to v)$.

▶ **Lemma 22** (†). *Let G be a graph and let r be a positive integer. Let c be the solution to the following optimization problem:*

$$y_{uv} \geq 0 \qquad \text{for every } u,v \in V(G) \text{ at distance between 2 and } r$$

$$x_u = \max_{z:uz\in E(G)} \sum_{v\in R_r(u,z)} y_{uv} \qquad \text{for every } u \in V(G)$$

$$\text{maximize } \frac{\sum_{uv:2\leq d_G(u,v)\leq r} y_{uv}}{\sum_{v\in V(G)} x_v}$$

Then every fractional or weak r-guidance system in G has maximum outdegree at least c.

As an example, this easily shows that no good weak guidance systems exist for graphs of girth at least five and large maximum average degree (the *maximum average degree* of a graph is the maximum of the average degrees of its subgraphs).

▶ **Lemma 23.** *Let G be a graph of girth at least five and maximum average degree $d \geq 2$. Every fractional or weak 2-guidance system in G has maximum outdegree at least $d/2$.*

Proof. Let $Z \subseteq V(G)$ be a smallest set such that $G[Z]$ has average degree d. Since $d \geq 2$, every vertex of $G[Z]$ has degree at least two, since deleting vertices of degree at most one would not decrease the average degree.

Since G has girth at least 5, any vertices $u,v \in Z$ at distance two in $G[Z]$ have a unique common neighbor $z \in Z$; we define

$$y_{uv} = \frac{1}{\deg_{G[Z]} z - 1}.$$

For any pair $u,v \in V(G)$ of vertices at distance two in G such that $\{u,v\} \not\subseteq Z$ or the common neighbor of u and v does not belong to Z, we define $y_{uv} = 0$. For any edge uz of G, if $\{u,z\} \subseteq Z$, then we have $|R_2(u,z) \cap Z| = \deg_{G[Z]} z - 1$, and thus

$$\sum_{v\in R_2(u,z)} y_{uv} = 1;$$

while if $\{u,z\} \not\subseteq Z$, then

$$\sum_{v\in R_2(u,z)} y_{uv} = 0.$$

Therefore,

$$x_u = \max_{z:uz\in E(G)} \sum_{v\in R_2(u,z)} y_{uv} = 1$$

for $u \in Z$ and $x_u = 0$ for $u \in V(G) \setminus Z$, and

$$\frac{\sum_{uv:d_G(u,v)=2} y_{uv}}{\sum_{u\in V(G)} x_u} = \frac{\frac{1}{2}\cdot\sum_{u\in Z}\sum_{z:uz\in E(G[Z])}\sum_{v\in R_2(u,z)} y_{uv}}{|Z|} = \frac{|E(G[Z])|}{|Z|} = d/2.$$

The claim now follows from Lemma 22. ◀

This shows that weak guidance systems can be qualitatively different from guidance systems only in graphs of girth at most four.

▶ **Corollary 24.** *Let G be a graph of girth at least five. For any $r \geq 2$, if G admits a weak r-guidance system of maximum outdegree at most c, then G also admits an r-guidance system of maximum outdegree at most $3c$.*

Proof. By Lemma 23, G has maximum average degree at most $2c$, and thus G is $2c$-degenerate. The claim then follows by Observation 6. ◀

Next, we consider the class of *split graphs* (a graph G is a split graph if there exists a partition (A, B) of its vertex set where A is a clique and B is an independent set). An easy construction based on the incidence graphs of finite projective planes shows that split graphs do not admit weak guidance systems of bounded maximum outdegree.

▶ **Lemma 25.** *For every n such that n is a power of a prime, there exists a split graph G_n with $2(n^2 + n + 1)$ vertices such that every fractional or weak 2-guidance system in G has maximum outdegree at least $(n + 1)/2$.*

Let us remark that split graphs are a special case of *chordal graphs* (graphs with no induced cycle of length at least four), and thus chordal graphs do not in general admit weak guidance systems of bounded maximum outdegree.

A *k-labeled graph* is a graph where each vertex is assigned a label from $[k]$ (several vertices can have the same label, and not all labels must be used). A k-labeled graph G is *constructible* if G has only one vertex or

- G is the disjoint union of at least two constructible k-labeled graphs, or
- G is obtained from a constructible k-labeled graph G' by, for some $i, j \in [k]$, changing all labels i to j, or
- G is obtained from a constructible k-labeled graph G' by, for some $i, j \in [k]$, adding all edges between vertices with labels i and j.

We say a graph has *clique-width* at most k if we can assign labels to its vertices so that the resulting k-labeled graph is constructible. For graphs of bounded clique-width, we again obtain a superconstant lower bound, though substantially smaller than in the case of split graphs.

▶ **Lemma 26.** *There exist arbitrarily large graphs G of clique-width at most 6 such that any weak 2-guidance system in G has maximum outdegree at least $\Omega(\log |V(G)| / \log \log |V(G)|)$.*

Note this is in contrast to Lemma 9, where we prove that graphs of bounded shrubdepth (a natural subclass of graphs of bounded clique-width) admit weak guidance systems with bounded maximum outdegree. On the positive side, we can show that graphs of bounded clique-width admit weak guidance systems of logarithmic outdegree.

Let us start by a useful observation. Suppose (A, B) is a partition of the vertex set of a graph G. For $u, v \in V(G)$, we write $u \equiv_{(A,B)} v$ if either $u, v \in A$ and u and v have the same neighbors in B, or $u, v \in B$ and u and v have the same neighbors in A.

▶ **Lemma 27.** *Let r be a positive integer or ∞. Suppose (A, B) is a partition of the vertex set of a graph G and $\equiv_{(A,B)}$ has k equivalence classes. If $G[A]$ and $G[B]$ have a weak r-guidance system of maximum outdegree at most c, then G has a weak r-guidance system of maximum outdegree at most $c + k$.*

Proof. Let $\vec{H}_A$ and $\vec{H}_B$ be weak r-guidance systems of maximum outdegree at most c in $G[A]$ and $G[B]$, respectively. Let $\vec{H}$ consist of $\vec{H}_A \cup \vec{H}_B$ and the following edges: For each $u \in V(G)$ and each equivalence class C of $\equiv_{(A,B)}$ intersecting the component of G containing u, choose a vertex u'_C in C nearest to u in G and a vertex $u_C \in G(u \to u'_C)$ arbitrarily, and add the edge (u, u_C). Clearly, $\vec{H}$ has maximum outdegree at most $c + k$.

Consider now any vertices $u, v \in V(G)$ at distance ℓ, where $2 \leq \ell \leq r$, and let P be a shortest path between u and v in G. If an edge of P incident with u or v belongs to $G[A] \cup G[B]$, switch the names of vertices u and v if necessary so that such an edge is incident with u. By symmetry, we can assume $u \in A$. If $P \subseteq G[A]$, then by Observation 7, $\vec{H}_A$ (and thus also $\vec{H}$) contains an edge directed from u to $G[A](u \to v) \subseteq G(u \to v)$ or an edge directed from v to $G[A](v \to u) \subseteq G(v \to u)$. Hence, suppose that $P \not\subseteq G[A]$.

If the first edge of P is contained in $G[A]$, then let P' be the longest initial segment of P contained in $G[A]$. If the first edge of P is not contained in $G[A]$, then let P' be the longest initial segment of P contained in $G[B \cup \{u\}]$. Let C be the equivalence class of $\equiv_{(A,B)}$ containing the last vertex z of P'. Note that $z \neq v$: In the first case, this is because P is not contained in $G[A]$. In the second case, this is because $|E(P)| = \ell \geq 2$ and the choice of the names of the vertices u and v implies that the last edge of P is not contained in $G[B]$. Since u'_C is a nearest vertex from u in C, u'_C is at distance at most $|E(P')|$ from u in G. Moreover, u'_C is in the same equivalence class of $\equiv_{(A,B)}$ as z, and thus u'_C is adjacent to the vertex following z in P. Hence, $u_C \in G(u \to v)$ and $\vec{H}$ contains the edge (u, u_C).

Observation 7 then implies that $\vec{H}$ is a weak r-guidance system in G. ◀

We combine this with the following well-known fact about clique-width.

▶ **Observation 28.** *If G is a graph with n vertices and clique-width at most k, then there exists a partition (A, B) of vertices of G such that $|A|, |B| \leq \frac{2}{3}n$ and $\equiv_{(A,B)}$ has at most $2k$ equivalence classes.*

Since any induced subgraph of a graph of clique-width at most k also has clique-width at most k, the desired bound follows.

▶ **Corollary 29.** *For every $k \geq 0$, every n-vertex graph of clique-width at most k has a partial orientation $\vec{H}$ of maximum outdegree $O(k \log n)$ such that $\vec{H}$ is a weak ∞-guidance system.*

3 Application: Approximation of distance domination and independence number

For a positive integer r, a set S of vertices of a graph G is *r-dominating* if every vertex of G is at distance at most r from S, and *r-independent* if distinct vertices of S are at distance greater than r from one another. Let $\gamma_r(G)$ denote the smallest size of an r-dominating set in G, and $\alpha_r(G)$ the largest size of an r-independent set in G. Observe that if D is an r-dominating and A a $2r$-independent set in G, then every vertex of D is at distance at most r from at most one vertex of A, and since every vertex of A is at distance at most r from D, we have $|A| \leq |D|$. Consequently, $\alpha_{2r}(G) \leq \gamma_r(G)$.

In general, the converse inequality does not hold and it is not even possible to bound $\gamma_r(G)$ by a function of $\alpha_{2r}(G)$; however, Dvořák [4] proved that if G is from a class of graphs with bounded expansion, then an approximate converse holds, i.e., $\gamma_r(G) = O(\alpha_{2r}(G))$. A small variation of the argument gives the following stronger claim.

▶ **Lemma 30.** *For all positive integers c and r, there exists a linear-time algorithm that, given a graph G together with its $2r$-guidance system of maximum outdegree less than c, returns an r-dominating set D and a $2r$-independent set A in G such that $|D| \leq c^2|A|$.*

Note that this implies that $\gamma_r(G) \leq |D| \leq c^2\gamma_r(G)$ and $\frac{1}{c^2}\alpha_{2r}(G) \leq |A| \leq \alpha_{2r}(G)$, and thus this gives a linear-time algorithm to approximate both the r-domination and the $2r$-independence number of G within the constant factor c^2.

The goal of this section is to show that a similar result holds for classes of graphs that admit weak guidance systems. However, the presence of a weak $2r$-guidance system of bounded outdegree is not by itself sufficient to ensure this.

▶ **Example 31.** Let $\vec{K}$ be a random orientation of the clique with vertex set $\{1,\dots,n\}$ (for each edge, choose direction uniformly independently at random). Let G be the graph obtained from $\vec{K}$ as follows: We have $V(G) = \{v_1,\dots,v_n,u_1,\dots,u_n,z\}$, where for each $i \in \{1,\dots,n\}$, u_i is adjacent to z, v_i, and all vertices v_j such that $(i,j) \in E(\vec{K})$. Let $\vec{H}$ be the partial orientation of G where for $i \in \{1,\dots,n\}$, the edge v_iu_i for $i \in \{1,\dots,n\}$ is directed towards u_i, and the edge u_iz is directed towards z. Note that for any distinct $i,j \in \{1,\dots,n\}$, we have $(i,j) \in E(\vec{K})$ or $(j,i) \in E(\vec{K})$, and thus the path $v_iu_iv_j$ or $v_ju_jv_i$ has the first edge directed towards its middle vertex. Consequently, $\vec{H}$ is a weak 2-guidance system for G of maximum outdegree one. Moreover, any 2-independent set in G contains at most one of the vertices $\{v_1,\dots,v_n,z\}$ and at most one of the vertices $\{u_1,\dots,u_n\}$, and thus $\alpha_2(G) \leq 2$. On the other hand, we have $\gamma_1(G) = \Omega(\log n)$: By replacing each vertex v_i by u_i in an optimal dominating set and possibly adding z, we obtain a dominating set D of size at most $\gamma_1(G)+1$ containing none of the vertices v_1, …, v_n, and to dominate these vertices, observe that with high probability D needs to contain $\Omega(\log n)$ of the vertices u_1, …, u_n.

We can solve this issue by adding another condition. An (r,k)-*halfgraph* in a graph G is a sequence u_1, …, u_k, v_1, …, v_k of vertices of G such that for every $i,j \in \{1,\dots,k\}$,

- if $j < i$, then the distance between u_i and v_j in G is greater than r, and
- if $j \geq i$, then the distance between u_i and v_j in G is exactly r.

We say that a graph is (r,k)-*stable* if it does not contain any (r,k)-halfgraph.

▶ **Theorem 32.** *For all positive integers r, k, and $c \geq 2$, there exists a constant b and a linear-time algorithm that, given an (r,k)-stable graph G together with its weak $2r$-guidance system $\vec{H}$ of maximum outdegree at most c, returns an r-dominating set D and a $2r$-independent set A in G such that $|D| \leq b|A|$.*

Proof. Let D and A' be the sets of vertices of G obtained as follows. We initialize $D := \emptyset$ and $A' := \emptyset$. As long as D is not an r-dominating set, we choose a vertex x at distance greater than r from D arbitrarily, we add x to A', and we add x and all vertices reachable in $\vec{H}$ from x by directed paths of length at most r to D. At the end, D is an r-dominating set and $|D| \leq c^{r+1}|A'|$.

Let $\prec$ be the linear ordering on vertices of A' such that $x \prec y$ when x was added to A' before y. The algorithm above enforces the following property (†): If $x \prec y$, then every vertex reachable from x by a directed path in $\vec{H}$ of length at most r is at distance greater than r from y.

Let $\sigma(1) = 0$ and for $p = 2,\dots,k$, let $\sigma(p) = c^{2r+1}(\sigma(p-1)+1)$. The set A' is not necessarily $2r$-independent, however it has the following property: If $S \subseteq A'$ consists of vertices pairwise at distance at most $2r$ from one another, then $|S| \leq \sigma(k+1)$. To prove this, we will show a stronger claim. For a positive integer $p \leq k+1$, a *p-halfgraph extension of S* is a sequence u_p, …, u_k, v_p, …, v_k of vertices of G such that for $i = p,\dots,k$,

- **(i)** $u_i \in A'$, $u_i \prec u_{i+1}$ if $i < k$, and $s \prec u_i$ for every $s \in S$.
- **(ii)** $\vec{H}$ contains a directed path from u_i to v_i of length exactly r,
- **(iii)** the distance between u_i and v_j in G is exactly r for every $j \in \{i,\dots,k\}$, and
- **(iv)** the distance between v_i and s is exactly r for every $s \in S$.

We will prove by induction on p that if there exists a p-halfgraph extension of S, then $|S| \leq \sigma(p)$; $|S| \leq \sigma(k+1)$ then follows, since an empty sequence trivially forms a $(k+1)$-halfgraph extension of S. For $p = 1$, note that if $1 \leq j < i \leq k$, then $u_j \prec u_i$ by (i), and (ii) and (†) imply that the distance between v_j and u_i in G is greater than r. Together with (iii), this implies that G contains an (r,k)-halfgraph, which is a contradiction. That is, the case $p = 1$ can never occur and the conclusion $|S| \leq \sigma(1)$ holds trivially.

Suppose now that $p \geq 2$ and that the claim holds for $p-1$. If $S = \emptyset$, then $|S| \leq \sigma(p)$ holds. Otherwise, let u_{p-1} be the last vertex of S in the ordering $\prec$. Since the distance between any vertices of S is at most $2r$ and $\vec{H}$ is a weak $2r$-guidance system, for each $s \in S \setminus \{u_{p-1}\}$, there exists a shortest path P_s in G between u_{p-1} and s directed in $\vec{H}$ towards one of its edges. Let Q_s be the longest initial segment of P_s directed away from u_{p-1}. By the choice of u_{p-1}, we have $s \prec u_{p-1}$, and thus (†) implies that the part of P_s directed away from s has length at most $r-1$, and consequently $|E(Q_s)| \geq r$.

For any directed path Q in $\vec{H}$ starting in u_{p-1} of length between r and $2r$, let S_Q be the set of vertices $s \in S \setminus \{u_{p-1}\}$ such that $Q_s = Q$. The preceding argument shows that S is the union of the sets S_Q over all such paths, and thus we can fix Q such that $|S_Q| \geq |S|/c^{2r+1}$. If $|S_Q| \leq 1$, then $|S| \leq 2^{2r+1} \leq \sigma(p)$, as required. Hence, suppose that $|S_Q| \geq 2$. Let v_{p-1} be the final vertex of Q and let s_Q be the first vertex of S_Q in the ordering $\prec$. Consider any vertex $s' \in S_Q \setminus \{s_Q\}$. Note that G contains a path of length at most $2r - |E(Q)| \leq r$ from s_Q to v_{p-1} with all but possibly the last edge directed away from s_Q in $\vec{H}$, and since $s_Q \prec s'$ by the choice of s_Q, (†) implies that s' is at distance at least r from v_{p-1}. Since s' is also at distance at most $2r$ from u_{p-1} through a shortest path whose initial segment is Q, s' is at distance at most $2r - |E(Q)| \leq r$ from v_{p-1}. We conclude that $|E(Q)| = r$ and all vertices of $S_Q \setminus \{s_Q\}$ are at distance exactly r from v_{p-1}. Therefore, $u_{p-1}, \ldots, u_k, v_{p-1}, \ldots, v_k$ is a $(p-1)$-halfgraph extension of $S_Q \setminus \{s_Q\}$, and $|S_Q \setminus \{s_Q\}| \leq \sigma(p-1)$ by the induction hypothesis. But then $|S| \leq c^{2r+1}|S_Q| \leq c^{2r+1}(\sigma(p-1)+1) = \sigma(p)$.

Let F be the auxiliary graph with $V(F) = A'$ and with distinct vertices $u, v \in A'$ adjacent if the distance between them in G is at most $2r$. We claim that each vertex of F has at most $c^{2r+1}\sigma(k+1)$ neighbors that precede it in the ordering $\prec$. Indeed, let N be the set of such neighbors of a vertex $u \in A'$, and for each directed path Q in $\vec{H}$ starting in u of length between r and $2r$, let N_Q consist of the vertices $v \in N$ such that Q is the maximal initial directed segment of a shortest path from u to v in G which is directed towards one of its edges by $\vec{H}$. As in the preceding part of the proof, note that (†) and the fact that $\vec{H}$ is a weak $2r$-guidance system implies that N is the union of the sets N_Q over such paths, and thus we can fix such a path Q for which $|N_Q| \geq |N|/c^{2r+1}$. However, the vertices of N_Q are at distance at most $2r - |E(Q)| \leq r$ from the final vertex of Q, and thus they are pairwise at distance at most $2r$ from one another. Consequently, $|N_Q| \leq \sigma(k+1)$, and $|N| \leq c^{2r+1}\sigma(k+1)$.

We conclude that F is $c^{2r+1}\sigma(k+1)$-degenerate, and thus it is $(c^{2r+1}\sigma(k+1)+1)$-colorable and has an independent set A of size at least

$$\frac{|A'|}{c^{2r+1}\sigma(k+1)+1} \geq \frac{|D|}{c^{r+1}(c^{2r+1}\sigma(k+1)+1)}.$$

By the construction of F, A is a $2r$-independent set in G. Therefore, the theorem holds with $b = c^{r+1}(c^{2r+1}\sigma(k+1)+1)$. ◀

By the results of Adler and Adler [1], for any structurally nowhere-dense graph class $\mathcal{G}$ and every r, there exists k so that all graphs in $\mathcal{G}$ are (r,k)-stable. In combination with Corollary 21, we have the following consequence.

▶ **Corollary 33.** *For any class $\mathcal{G}$ with structurally bounded expansion and for any positive integer r, there exists a constant b and a polynomial-time randomized algorithm that, given a graph $G \in \mathcal{G}$ with probability at least $1/2$ returns an r-dominating set D and a $2r$-independent set A in G such that $|D| \leq b|A|$.*

4 Conclusions

As we have shown, many interesting graph classes admit weak guidance systems of bounded maximum outdegree, including

- interval graphs,
- classes with structurally bounded expansion, and
- distance powers of graphs with bounded outdegree weak guidance systems.

However, we do not have an exact characterization of the graph classes with this property.

▶ **Problem 34.** *Characterize hereditary graph classes $\mathcal{G}$ such that for every positive integer r, every graph from $\mathcal{G}$ admits a weak r-guidance system of bounded maximum outdegree.*

We have also exhibited several graph classes that only admit weak guidance systems whose outdegree grows slowly with the number of vertices of the graph, in particular

- structurally nowhere-dense classes, and
- graphs of bounded clique-width.

Again, we do not have a good description of the graph classes with this property.

▶ **Problem 35.** *Characterize hereditary graph classes $\mathcal{G}$ such that for every positive integer r, every graph $G \in \mathcal{G}$ admits a weak r-guidance system of maximum outdegree at most $|V(G)|^{o(1)}$.*

In sparse graphs, guidance systems and related notions (such as generalized coloring numbers) have various algorithmic and structural applications. We suspect that similar applications can be found for weak guidance systems as well, generalizing them to dense graphs; we demonstrated this on the example of approximation algorithms for distance domination and independence number.

References

1. H. Adler and I. Adler. Interpreting nowhere dense graph classes as a classical notion of model theory. *European Journal of Combinatorics*, 36:322–330, 2014.
2. J. Dreier. Lacon-and shrub-decompositions: a new characterization of first-order transductions of bounded expansion classes. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021.
3. J. Dreier, J. Gajarský, S. Kiefer, M. Pilipczuk, and Sz. Toruńczyk. Treelike decompositions for transductions of sparse graphs. *arXiv*, 2022. `arXiv:2201.11082`.
4. Z. Dvořák. Constant-factor approximation of domination number in sparse graphs. *European Journal of Combinatorics*, 34:833–840, 2013.
5. Z. Dvořák. Induced subdivisions and bounded expansion. *European Journal of Combinatorics*, 69:143–148, 2018.
6. Z. Dvořák. On distance r-dominating and $2r$-independent sets in sparse graphs. *J. Graph Theory*, 91(2):162–173, 2019.
7. Z. Dvořák and A. Lahiri. Approximation schemes for bounded distance problems on fractionally treewidth-fragile graphs. In *29th Annual European Symposium on Algorithms (ESA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

8 J. Gajarský, P. Hliněný, J. Obdržálek, D. Lokshtanov, and MS Ramanujan. A new perspective on FO model checking of dense graph classes. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–23, 2020.

9 J. Gajarský, S. Kreutzer, J. Nešetřil, P. Ossona De Mendez, M. Pilipczuk, S. Siebertz, and Sz. Toruńczyk. First-order interpretations of bounded expansion classes. *ACM Transactions on Computational Logic (TOCL)*, 21(4):1–41, 2020.

10 R. Ganian, P. Hliněný, J. Nešetřil, J. Obdržálek, and P. Ossona de Mendez. Shrub-depth: Capturing Height of Dense Graphs. *Logical Methods in Computer Science*, 15, 2019.

11 Ł. Kowalik and M. Kurowski. Oracles for bounded length shortest paths in planar graphs. *ACM Trans. Algorithms*, 2:335–363, 2006.

12 J. Nešetřil, P. Ossona de Mendez, M. Pilipczuk, R. Rabinovich, and S. Siebertz. Rankwidth meets stability. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2014–2033. SIAM, 2021.

13 J. Nešetřil and P. Ossona de Mendez. *Sparsity (Graphs, Structures, and Algorithms)*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

14 J. Pach and P. K. Agarwal. *Combinatorial geometry*. John Wiley & Sons, 2011.

15 M. Pilipczuk, S. Siebertz, and Sz. Toruńczyk. On the number of types in sparse graphs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS'18, pages 799–808. ACM, 2018.

16 F. Reidl and B. D. Sullivan. A color-avoiding approach to subgraph counting in bounded expansion classes. *arXiv preprint*, 2020. `arXiv:2001.05236`.

A Appendix

Let us start by giving the short proof that r-guidance systems can be used to characterize bounded expansion and nowhere-density.

Proof of Lemma 11. Suppose for a contradiction that $\mathcal{G}$ is not nowhere-dense. By assumptions, for every $\varepsilon > 0$, every graph $G \in \mathcal{G}$ has an orientation with maximum outdegree at most $c(1,\varepsilon)|V(G)|^\varepsilon$, and thus the maximum average degree of subgraphs of G is at most $2c(1,\varepsilon)|V(G)|^\varepsilon$. By [5, Theorem 6], there exists $r \geq 2$, a graph $G \in \mathcal{G}$, and a graph H of average degree $d > 2c(r,\varepsilon)|V(G)|^\varepsilon$ such that G contains the graph H' obtained from H by subdividing each edge exactly $r-1$ times as an induced subgraph. Since $\mathcal{G}$ is closed under induced subgraphs, we can assume $G = H'$. Suppose $\vec{H}$ is an r-guidance system in G. Then for every $uv \in E(H)$, the corresponding path P_{uv} of length r in G contains an edge directed away from u or from v, and thus the average outdegree of the vertices of H in G is at least $|E(H)|/|V(H)| = d/2 > c(r,\varepsilon)|V(G)|^\varepsilon$. This contradicts the assumptions.

The argument for the bounded expansion case is analogous, using [5, Theorem 5] instead of [5, Theorem 6]. ◀

B Algorithmic aspects

Fractional r-guidance systems can be directly used to test presence of shortest paths, with a small probability of error. Let p be a fractional r-guidance system in a graph G. If u is a non-isolated vertex of G, then by a *p-random neighbor of u*, we mean a neighbor of u selected at random, with the probability that a neighbor v is selected being $p(u,v)/d_p^+(u)$; if $d_p^+(u) = 0$, the probability is $1/\deg u$, instead. For distinct vertices u and v and a positive integer r, a *random (p,r)-exploration* between u and v is a random pair of walks (P_u, P_v) from u and v selected as follows:

- If $uv \in E(G)$, then $P_u = uv$ and $P_v = v$.
- Otherwise, if $r = 1$ or u or v is an isolated vertex, then $P_u = u$ and $P_v = v$.
- Otherwise, let $x \in \{u, v\}$ be selected uniformly at random, and let y be a p-random neighbor of x;
  - if $x = u$, then select a random $(p, r-1)$-exploration (P_y, P_v) between y and v and let P_u be the concatenation of uy and P_y, and
  - if $x = v$, then select a random $(p, r-1)$-exploration (P_u, P_y) between u and y and let P_v be the concatenation of vy and P_y.

▶ **Observation 36.** *Suppose p is a fractional r-guidance system in a graph G, of maximum outdegree c. Let u and v be distinct vertices of G at distance at most r, and let (P_u, P_v) be a random (p, r)-exploration between u and v. The probability that $P_u \cup P_v$ is a shortest path between u and v in G is at least $(4c)^{-(r-1)}$.*

Note that for Observation 36 to be practically useful, we would need a representation of p that enables us to choose a p-random neighbor efficiently; in that case, we could iterate $k(4c)^{r-1}$ times the procedure from Observation 36 to find the shortest path between u and v (or decide that the distance between them is greater than r) with error probability at most e^{-k}. Next, let us show how to turn a fractional guidance system into a (slightly worse) weak guidance system.

Proof of Lemma 15. Let us say that pair $\{u, v\}$ of vertices is *dissatisfied* by a partial orientation $\vec{F}$ if the distance ℓ between u and v satisfies $2 \leq \ell \leq r$ and $\vec{F}$ contains neither an edge from u to $G(u \to v)$ nor an edge from v to $G(v \to u)$. By Observation 7, $\vec{F}$ is a weak r-guidance system if and only if there are no dissatisfied pairs.

Let X be any set of pairs of vertices of G at distance between 2 and r. Let $\vec{F}$ be a random partial orientation of G obtained by, for each non-isolated vertex z of G, choosing a random p-neighbor z' and adding the edge (z, z'). Clearly, $\vec{F}$ has maximum outdegree at most one. Moreover, consider any $\{u, v\} \in X$. By (1) and symmetry, we can assume that

$$\sum_{y \in G(u \to v)} p(u, y) \geq 1/2.$$

Hence, the probability that $u' \in G(u \to v)$ (and thus $\{u, v\}$ is not dissatisfied in $\vec{F}$) is at least $\frac{1}{2c}$. By the linearity of expectation, the expected number of dissatisfied pairs in X is at most $\left(1 - \frac{1}{2c}\right)|X|$.

Moreover, we can use the method of conditional probabilities to derandomize this procedure and to deterministically construct a partial orientation $\vec{F}$ of G of maximum outdegree at most one such that the number of pairs in X dissatisfied by $\vec{F}$ is at most $\left(1 - \frac{1}{2c}\right)|X|$. Indeed, we can select the outneighbors one by one, always maintaining the invariant (initially satisfied by the computation from the previous paragraph) that the expected number of pairs in X dissatisfied by the orientation obtained by choosing the remaining outneighbors as random p-neighbors is at most $\left(1 - \frac{1}{2c}\right)|X|$. To do so, when processing a vertex u, we only need to be able to compute this expected number after each possible choice of the outneighbor of u, which is straightforward due to the linearity of expectation.

Now, to obtain $\vec{H}$, we let X_0 be the set of all pairs of vertices whose distance is between 2 and r in G. Then, for $i = 1, \ldots, m$, we use the procedure described in the previous paragraph to find a partial orientation $\vec{F}_i$ of maximum outdegree at most one so that the set X_i of pairs from X_{i-1} dissatisfied by $\vec{F}_i$ has size at most $\left(1 - \frac{1}{2c}\right)|X_{i-1}|$. Note that

$$|X_m| \leq \left(1 - \tfrac{1}{2c}\right)^m |X_0| \leq \frac{|X_0|}{n^2} < 1,$$

and thus $X_m = \emptyset$. Consequently, no pair is dissatisfied by

$$\vec{H} = \bigcup_{i=1}^{m} \vec{F}_i,$$

and thus $\vec{H}$ is the desired weak r-guidance system in G. ◀

Let us now show that the logarithmic loss cannot be avoided in general.

Proof of Lemma 17. Let us use the notation from the definition of the graph $G_{a,k}$. Note that

- for $i = 1, \ldots, m$ and $v \in L$, the expected number of neighbors of v in R_i is $a/2$, and by Chernoff inequality, the probability that v has less than $a/3$ neighbors in R_i is less than $\exp(-a/36)$.
- for distinct vertices $u, v \in R$, the expected number of common neighbors of u and v in L is $a/4$, and by Chernoff inequality, the probability that u and v have less than $a/5$ common neighbors in L is less than $\exp(-a/200)$,
- for distinct $u, v \in L$, the probability that u and v have less than $a/5$ common neighbors in R_1 is also less than $\exp(-a/200)$, and
- for $i \in 1, \ldots, m$ and a k-tuple K of vertices of R_i, the expected number of vertices of L with no neighbor in K is $2^{-k}a$, and by Chernoff inequality, the probability that the number of such vertices is at most $2^{-k-1}a$ is at most $\exp\bigl(-2^{-k-3}a\bigr)$.

Hence, the probability that any of these events occurs is less than

$$ma \cdot \exp(-a/36) + (m^2+1)a^2 \cdot \exp(-a/200) + ma^k \cdot \exp\bigl(-2^{-k-3}a\bigr) < 1$$

if a is sufficiently large (and using the assumption that $k \leq \log a$; note that the basis of the logarithm is e, and thus $2^k \leq a^{\log 2} \ll a$). Hence, with positive probability,

- for $i = 1, \ldots, m$, each vertex $v \in L$ has at least $a/3$ neighbors in R_i,
- any distinct vertices $u, v \in R$ have at least $a/5$ common neighbors in L,
- any distinct vertices $u, v \in L$ have at least $a/5$ common neighbors in R_1, and
- for $i \in 1, \ldots, m$ and for every k-tuple K of vertices of R_i, more than $2^{-k-1}a$ vertices of L have no neighbor in K.

Let us define a fractional orientation p of $G_{a,k}$ as follows:

- For $i = 1, \ldots, m$ and $v \in R_i$, we set $p(x_i, v) = 3/a$,
- for each adjacent $u \in R$ and $z \in L$, we set $p(u, z) = 2.5/a$, and
- for each adjacent $z \in L$ and $u \in R_1$, we set $p(z, u) = 2.5/a$;

p is 0 everywhere else. Note that this fractional orientation has maximum outdegree at most 3, since $\deg x_i = |R_i| = a$, the number of neighbors of $u \in R$ in L is at most $|L| = a$, and the number of neighbors of $z \in L$ in R_1 is at most $|R_1| = a$. Consider now any vertices $x, y \in V(G_{a,k})$ at distance exactly two from one another. Note that $G_{a,k}$ is bipartite, and thus either $x, y \in R$, or $x, y \in V(G_{a,k}) \setminus R$. There are the following cases:

- One of x and y belongs to $\{x_1, \ldots, x_m\}$, say $x = x_i$. Then y necessarily belongs to L, and y has at least $a/3$ neighbors in R_i. Hence, $|G_{a,k}(x \to y)| \geq a/3$ and

$$\sum_{z \in G_{a,k}(x \to y)} p(x, z) \geq a/3 \cdot 3/a = 1.$$

- Both x and y belong to L. Since x and y have at least $a/5$ common neighbors in R_1, we have $|G_{a,k}(x \to y) \cap R_1| = |G_{a,k}(y \to x) \cap R_1| \geq a/5$, and

$$\sum_{z \in G_{a,k}(x \to y)} p(x,z) + \sum_{z \in G_{a,k}(y \to x)} p(y,z) \geq 2 \cdot a/5 \cdot 2.5/a = 1.$$

- Similarly, if $x, y \in R$, then x and y have at least $a/5$ common neighbors in L, and

$$\sum_{z \in G_{a,k}(x \to y)} p(x,z) + \sum_{z \in G_{a,k}(y \to x)} p(y,z) \geq 2 \cdot a/5 \cdot 2.5/a = 1.$$

Therefore, p is a fractional 2-guidance system for $G_{a,k}$.

Consider now any partial orientation $\vec{H}$ of $G_{a,k}$ with maximum outdegree at most k. Then each vertex $v \in L$ has an outneighbor in R_i for at most k choices of i, and thus there exists $i \in \{1, \ldots, m\}$ such that at least $(1 - k/m)a = \left(1 - 2^{-k-1}\right)a$ vertices of L have no outneighbor in R_i. Let K be a k-tuple of vertices of R_i containing all outneighbors of x_i. More than $2^{-k-1}a$ vertices of L have no neighbor in K, and thus there exists a vertex $v \in L$ with no outneighbor in R_i and no neighbor in K. However, x_i and v are at distance 2, yet neither x_i nor v has an outneighbor in $G_{a,k}(x_i \to v) = G_{a,k}(v \to x_i) \subseteq R_i \setminus K$. Hence $\vec{H}$ is not a weak 2-guidance system. Consequently, every weak 2-guidance system for $G_{a,k}$ must have maximum outdegree greater than k. ◀

For a first-order formula $\psi(\vec{x}, \vec{y})$ with two groups $\vec{x}$ and $\vec{y}$ of free variables, a graph G, and a $|\vec{x}|$-tuple $\vec{u}$ of vertices of G, let $S_{\psi,G}(\vec{u})$ be the set of $|\vec{y}|$-tuples $\vec{v}$ of vertices of G such that $G \models \psi(\vec{u}, \vec{v})$, and let $\mathcal{S}_{\psi,G}$ be the system

$$\{S_{\psi,G}(\vec{u}) : \vec{u} \in V(G)^{|\vec{x}|}\}$$

of sets of $|\vec{y}|$-tuples of vertices of G. The following bound follows from the results of Adler and Adler [1], see also [15] for a more precise bounds and the discussion of the possibility to introduce vertex and edge colors (unary and binary predicates from the statement of the theorem).

▶ **Theorem 37.** *For every nowhere-dense graph class $\mathcal{G}$ and a first-order formula $\psi(\vec{x}, \vec{y})$ using unary predicate symbols U_1, …, U_s and binary predicate symbols E_1, …, E_t, there exists a constant d such that the following claim holds. Consider any graph $G \in \mathcal{G}$, and interpret U_i for $i \in \{1, \ldots, s\}$ as a subset of $V(G)$ and E_j for $j \in \{1, \ldots, t\}$ as a subset of $E(G)$. Then the system $\mathcal{S}_{\psi,G}$ has VC-dimension at most d.*

This easily gives the bound on $\mathrm{VC}(G, r)$ for structurally nowhere-dense classes.

Proof of Lemma 20. Since $\mathcal{G}$ is structurally nowhere-dense, there exists a nowhere-dense class $\mathcal{G}_0$ and a transduction $T = (k, M, U_1, \ldots, U_s, \omega, \epsilon)$ such that for each $G \in \mathcal{G}$ there exists a graph $H \in \mathcal{G}_0$ such that $G \in T(H)$; let C_1^G, …, C_s^G be the corresponding subsets of $V(H)$ used to interpret U_1, …, U_s.

For $H \in \mathcal{G}_0$, let $(kH)'$ be the graph obtained from the disjoint union of k copies of G by adding a clique on each k-tuple of vertices corresponding to the same vertex of H, and let M_H be the set of the edges of these cliques. Also, let E_H be the set of edges of kH. Let $\mathcal{G}_1 = \{(kH)' : H \in \mathcal{G}_0\}$. Since $(kH)'$ is a subgraph of the lexicographic product of H with a clique of bounded size and $\mathcal{G}_0$ is nowhere-dense, the class $\mathcal{G}_1$ is nowhere-dense as well [13].

Note that there exists a first-order formula $\psi_r(x_1, x_2, y)$ with three free variables such that for each $u, v \in V(G)$ satisfying $2 \leq d_G(u, v) \leq r$ and $z \in V(G)$, $G \models \psi_r(u, v, z)$ if and only if $z \in G(u \to v)$. Let ψ'_r be the formula obtained from ψ_r by restricting the quantification to vertices satisfying ω and replacing each usage of the adjacency predicate by ϵ. Clearly, if $G \in T(H)$, then

$$G \models \psi_r(u, v, z) \text{ iff } (kH)', U_1 := C_1^G, \ldots, U_s := C_s^G, E := E_H, M := M_H \models \psi'_r(u, v, z).$$

Therefore, with the interpretation of the unary and binary symbols as above, $\mathcal{S}_{\psi_r,G}$ is a subset of $\{S \cap V(G) : S \in \mathcal{S}_{\psi'_r,(kH)'}\}$, and thus the VC-dimension of $\mathcal{S}_{\psi_r,G}$ is at most as large as the VC-dimension of $\mathcal{S}_{\psi'_r,(kH)'}$. Since $(kH)' \in \mathcal{G}_1$ and $\mathcal{G}_1$ is nowhere-dense, Theorem 37 implies that this VC-dimension is bounded. ◀

C Graph classes without bounded outdegree weak guidance systems

Let us now prove the lower bound for split graphs.

Proof of Lemma 25. It is well-known that whenever n is a power of prime, there exists a finite projective plane B of order n, i.e., a system of n^2+n+1 subsets of the set $A = [n^2+n+1]$ with the property that

- **(i)** $|p_1 \cap p_2| = 1$ for every distinct $p_1, p_2 \in B$ and
- **(ii)** every element of A belongs to exactly $n+1$ sets from B.

Let G_n be the graph with vertex set $A \cup B$, vertices in A forming a clique, vertices in B forming an independent set, and vertices $z \in A$ and $p \in B$ adjacent iff $z \in p$. Note that distinct vertices of B are at distance two in G_n by (i), and that for each $p \in B$ and $z \in p$, $|R_2(p, z) \cap B| = n$ by (ii). Therefore, defining $y_{p_1p_2} = 1$ for any distinct $p_1, p_2 \in B$ and $y_{uv} = 0$ for any other pair u, v of vertices of G_n, we have

$$x_p = \max_{z:z\in p} \sum_{p' \in R_2(p,z)} y_{pp'} = n$$

for $p \in B$ and $x_z = 0$ for $z \in A$. Therefore,

$$\frac{\sum_{uv:d_{G_n}(u,v)=2} y_{uv}}{\sum_{u\in V(G_n)} x_u} = \frac{\binom{|B|}{2}}{|B|n} = \frac{|B|-1}{2n} = \frac{n+1}{2}.$$

The claim now follows from Lemma 22. ◀

Finally, let us prove the following claim, which clearly implies Lemma 26.

▶ **Lemma 38.** *For every $d \geq 0$ and $a \geq \max(2, 2d-1)$, there exists a constructible 6-labeled graph $H_{d,a}$ with half its vertices labeled 1 and half its vertices labeled 2, such that*

- **(i)** *$|V(H_{d,a})| \leq 8a^d - 6$ and*
- **(ii)** *for every partial orientation $\vec{G}$ of $H_{d,a}$ of maximum outdegree less than d, there exist vertices u and v of labels 1 and 2, respectively, at distance exactly two, such that for every common neighbor x of u and v, we have $(u, x), (v, x) \notin E(\vec{G})$.*

Proof. For $d = 0$, we can let $H_{0,a} = K_2$ with one vertex labeled 1 and the other vertex labeled 2. Suppose we already constructed $H_{d-1,a}$, and let us show how to inductively obtain $H_{d,a}$. First, let $H'_{d-1,a}$ be the graph obtained from $H_{d-1,a}$ by adding vertices v_3 and v_4 with labels 3 and 4 and adding all edges between vertices with labels 1 and 4 and between vertices with labels 2 and 3. Next, we form the disjoint union of a copies of $H'_{d-1,a}$. Then we add

two vertices v_5 and v_6 with labels 5 and 6, and all edges between vertices with labels i and $i+2$ for $i \in \{3,4\}$. Finally, we relabel vertices with labels 3 and 5 to label 1 and vertices with labels 4 and 6 to label 2.

The construction uses only 6 labels, and thus $H_{d,a}$ is a constructible 6-labeled graph. Moreover,

$$|V(H_{d,a})| = a(|V(H_{d-1,a}) + 2) + 2 \leq a(8a^{d-1} - 4) + 2 \leq 8a^d - 6,$$

where the last inequality holds since $a \geq 2$. Consider any partial orientation $\vec{G}$ of $H_{d,a}$ of maximum outdegree less than d. Since v_5 and v_6 have outdegree less than d, for one of the $a \geq 2d-1$ copies of $H'_{d-1,a}$ in $H_{d,a}$, denoted by F', we have $(v_i, v) \notin \vec{G}$ for every $i \in \{5,6\}$ and $v \in V(F')$. Let F be the copy of $H_{d-1,a}$ in F'. Suppose that for any two vertices u and v of F of labels 1 and 2, respectively, at distance exactly two in $H_{d,a}$, there exists a common neighbor x of u and v in $H_{d,a}$ such that $(u,x) \in E(\vec{G})$ or $(v,x) \in E(\vec{G})$. The construction of $H'_{d-1,a}$ and $H_{d,a}$ ensures that such a common neighbor x necessarily belongs to F, as we did not add any vertex adjacent both to vertices with label 1 and with label 2. Hence, by the induction hypothesis, the restriction of $\vec{G}$ to F has maximum outdegree at least $d-1$. Let u be a vertex of F with at least $d-1$ outneighbors in $\vec{G}$ belonging to F. By symmetry, we can assume u has label 1. Since $\vec{G}$ has maximum outdegree less than d, we have $(u, v_4) \notin E(\vec{G})$. Moreover, by the choice of F', we have $(v_6, v_4) \notin E(\vec{G})$. Note that v_6 has label 2 in $H_{d,a}$ and the copy of v_4 in F is the only common neighbor of u and v_6 in $H_{d,a}$. This shows that $H_{d,a}$ satisfies the property (ii). ◀

Exact Matching: Algorithms and Related Problems

Nicolas El Maalouly
Department of Computer Science, ETH Zürich, Switzerland

Abstract

In 1982, Papadimitriou and Yannakakis introduced the *Exact Matching* (EM) problem where given an edge colored graph, with colors red and blue, and an integer k, the goal is to decide whether or not the graph contains a perfect matching with exactly k red edges. Although they conjectured it to be **NP**-complete, soon after it was shown to be solvable in randomized polynomial time in the seminal work of Mulmuley et al., placing it in the complexity class **RP**. Since then, all attempts at finding a deterministic algorithm for EM have failed, thus leaving it as one of the few natural combinatorial problems in **RP** but not known to be contained in **P**, and making it an interesting instance for testing the hypothesis $\mathbf{RP} = \mathbf{P}$. Progress has been lacking even on very restrictive classes of graphs despite the problem being quite well known as evidenced by the number of works citing it.

In this paper we aim to gain more insight into EM by studying a new optimization problem we call *Top-k Perfect Matching* (TkPM) which we show to be polynomially equivalent to EM. By virtue of being an optimization problem, it is more natural to approximate TkPM so we provide approximation algorithms for it. Some of the approximation algorithms rely on a relaxation of EM on bipartite graphs where the output is required to be a perfect matching with a number of red edges differing from k by at most $k/2$, which is of independent interest and generalizes to the *Exact Weight Perfect Matching* (EWPM) problem. We also consider parameterized algorithms and show that TkPM can be solved in FPT time parameterized by k and the independence number of the graph. This result again relies on new tools developed for EM which are also of independent interest.

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

Keywords and phrases Perfect Matching, Exact Matching, Approximation algorithms, Independence number, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.29

Related Version *Full Version*: `https://arxiv.org/abs/2203.13899`

Acknowledgements I want to thank Ola Svensson for introducing me to the Exact Matching problem, for defining the Top-k Perfect Matching problem and for helpful discussions and feedback.

1 Introduction

Deciding whether randomization adds power to sequential algorithms is an central problem in complexity theory. The main question there is whether $\mathbf{P} = \mathbf{RP}$ which remains a big open problem and is tied to other important questions in the field [22]. Only few natural problems are known to be in **RP** while no deterministic algorithms are known for them. Exact Matching (EM), defined in 1982 by Papadimitriou and Yannakakis [27], is one such problem.

Exact Matching (EM)
Input: A graph G, with each edges colored red or blue, and integer k.
Task: Decide whether there exists a perfect matching M in G with exactly k red edges.

© Nicolas El Maalouly;
licensed under Creative Commons License CC-BY 4.0

At the time of its introduction it was conjectured to be **NP**-complete. Only a few years later, however, it was shown to be in **RP** by Mulmuley, Vazirani and Vazirani [26], which makes it unlikely to be **NP**-hard. In fact, in was even shown to be in **RNC** which is defined as the class of decision problems allowing an algorithm running in polylogarithmic time[1] (i.e., $O(\log n^c)$ for some constant $c > 0$) using polynomially many parallel processors, while having additional access to randomness (we refer the interested reader to [6] Chapter 12 for a formal definition). Derandomizing matching problems from this complexity class is also a big open problem [30]. This makes EM even more interesting since randomness allows it to be efficiently parallelizable while it even remains difficult to solve sequentially without such access to randomness.

The interest in EM is evidenced by the numerous works that cite it. These include works on the parallel computation complexity of the matching problem [30], planarizing gadgets for perfect matchings [20], multicriteria optimization [19], matroid intersection [7], DNA sequencing [5], binary linear equation systems with small hamming weight [2], recoverable robust assignment [15] in addition to generalizations of the problem with multiple color constraints [4, 24, 25, 29]. Despite that, deciding whether EM is in **P** has remained an open problem for almost four decades and little progress has been made even for very restricted classes of graphs, thus highlighting the surprising difficulty of the problem.

1.1 Prior Work

Restricted Graph Classes. When it comes to restricted classes of graphs, results go in two directions. The first is the sparse graphs regime where in the extreme case we have trees for which EM can be solved by a simple dynamic program (DP). This can also be generalized to bounded tree-width graphs. Such a DP has not been explicitly given in the literature but would be easy to construct by keeping track of how every edge in a bag is matched (not yet matched, matched outside the bag or matched inside the bag) as well as the total number of red edges in the matching so far. It is easy to see that the number of possible states is at most $O(3^{tw} \cdot n)$ (where n comes from the possible number of red edges in the matching) resulting in an FPT algorithm parameterized by the tree width of the graph. Continuing with sparse graph classes, EM is also known to be solvable for planar graphs [32] by relying on the existence of Pfaffian orientations to derandomize the **RNC** algorithm. The same techniques used for this derandomization also allow for computing the matching generating function (see [18] Chapter 1 for a definition) and can be generalized to other graph classes such as $K_{3,3}$-minor free graphs and graphs embeddable on a surface of bounded genus [16]. Computing the matching generating function was recently shown to be #**P**-hard already for K_8-minor free graphs [9] so these results do not generalize much further and are restricted to very sparse graphs.

The second direction is dense graph classes. Here it is known that EM is in **P** for complete and complete bipartite graphs, i.e., graphs of independence number $\alpha = 1$ and bipartite graphs of bipartite independence number[2] $\beta = 1$. In fact, these results are already non-trivial and at least four different articles have appeared on resolving them [23, 31, 17, 21]. Very recently, however, El Maalouly and Steiner [13] pushed the boundary of positive results further by showing that EM is in **P** for all graphs of bounded independence number and all bipartite graphs of bounded bipartite independence number.

---

[1] in the following, n denotes the number of vertices of the input graph

[2] The *bipartite independence number* of a bipartite graph G equipped with a bipartition of its vertices is defined as the largest number β such that G contains a *balanced independent set* of size 2β, i.e., an independent set using exactly β vertices from each side of the bipartition.

Generalizations. As mentioned above, prior work also considered a generalization of the problem to multiple color constraints, known as Bounded Color Matching (BCM).

Bounded Color Matching (BCM)
Input: A weighted and edge-colored (with colors $c_1, ..., c_l$) graph G and integers $k_1, ..., k_l$.
Task: Find a maximum weight matching M in G with at most k_i edges of color i for all $i \in \{1, ..., l\}$.

BCM is known to be NP-hard [28]. Mastrolilli and Stamoulis [25] provide bi-criteria approximation schemes which give an approximately maximum matching with small constraint violations. Stamoulis [29] also gives a 1/2-approximation for the objective with no constraint violations. No prior work considered bounds on the constraint violations while requiring an optimal objective, i.e., a perfect matching if the graph is unweighted. For EM, however, Yuster [32] proved that given an instance of the problem, one can decide in polynomial time that either G contains no perfect matching with exactly k red edges, or one can compute an *almost* perfect matching (i.e., of size at least $\lfloor \frac{n}{2} \rfloor - 1$) containing k red edges. This means that the techniques used in the bi-criteria approximation of BCM do not provide much further insight into solving EM since they relax the perfect matching requirement.

Another way to generalize the problem is to have a weighted instead of edge-colored graph, and require the output perfect matching to have an exact weight.

Exact Weight Perfect Matching (EWPM)
Input: A weighted graph G and integer W.
Task: Find a perfect matching M in G with $w(M) = W$.

EWPM can be reduced to EM if the edge weights are polynomial in the input size but is known to be NP-hard for exponential weights [20]. This makes approximation algorithms that aim to minimize the constraint violation even more desirable for EWPM.

1.2 Our contribution

Exact Matching. We provide an algorithm for a relaxed version of EM on bipartite graphs where we require the output to be a perfect matching and allow for a constraint violation that is a constant fraction of k, 0.5 in this case.

▶ **Theorem 1.** *There exists a deterministic polynomial time algorithm that, given a "Yes" instance of EM on a bipartite graph, outputs a perfect matching M with $0.5k \leq |R(M)| \leq 1.5k$, where $|R(M)|$ is the number of red edges in M.*

This can also be seen as an attempt to approximate the EM problem without relaxing the perfect matching constraint. This type of approximation is the first of its kind for EM. Note that in light of the above mentioned result by Yuster [32] (i.e., an algorithm that outputs an *almost* perfect matching containing k red edges) one would think that it should not be too difficult to find an algorithm for the relaxed version of EM with a constraint violation of only one red edge. However, the perfect matching requirement seems to be intrinsic to the difficulty of the problem (given the simplicity of Yuster's algorithm) and many attempts at improving the constraint violation of Theorem 1 have failed so far. We also show that the approximation algorithm works for the more general problem of EWPM, only loosing $1/poly(n)$ in the approximation factor for exponential weights.

▶ **Corollary 2.** ($\star$) *There exists a deterministic polynomial time algorithm that, given a "Yes" instance of EWPM on a bipartite graph with input weights bounded by a polynomial (resp. exponential) function of the input size, outputs a perfect matching M with $0.5W \leq w(M) \leq 1.5W$ (resp. $(0.5 - 1/poly(n))W \leq w(M) \leq (1.5 + 1/poly(n))W$).*

We also introduce a new way of finding alternating cycles with certain color and weight properties in FPT time parameterized by the number of edges in the cycles.

▶ **Proposition 3.** *Let $G = (V, E, w)$ be an edge colored and weighted graph with edge colors red and blue, and let M and M' be two perfect matchings in G and $\mathcal{C} = M \Delta M'$ s.t. $|E(\mathcal{C})| \leq L$ for some integer L. Then there exists an algorithm running in time $f(L)poly(n)$ (for $f(L) = L^{O(L)}$) that, given G and M as input, outputs a perfect matching M'' in G with $w(M'') \geq w(M')$ and $|R(M'')| = |R(M')|$.*

This allows us to get an FPT algorithm parameterized by the circumference of the graph.

▶ **Theorem 4.** *There exists a deterministic FPT algorithm, parameterized by the circumference[3] of the graph, for the Exact Matching problem in general graphs.*

Top-k Perfect Matching. The above studied problems suffer from the fact that they are not optimization problems (due to the exactness constraint which requires the optimization of more than one objective) and are thus less natural to approximate. For this reason, we study a new matching problem called Top-k Perfect Matching.

Top-k Perfect Matching (TkPM)
Input: A weighted graph G and integer k.
Task: Find a perfect matching in G maximizing the top-k weight function.

Here the top-k weight function is defined as the sum of the weights of the k highest weight edges in the matching. To our knowledge, this problem has not yet been considered in the literature, but similar types of optimization objectives have been used for other problems such as k-clustering and load balancing [8]. We show that this problem can also be reduced to EM (in deterministic polynomial time) when the edge weights are polynomially bounded in the input size.

▶ **Theorem 5.** *$TkPM \leq_p EM$ for polynomially bounded weights.*[4]

This puts TkPM with polynomial weights in the class **RP** and it remains open whether or not it is in **P**, thus making it another natural problem in this category. Interestingly, a recent result shows that EM can in turn be reduced to TkPM, making the two problems polynomially equivalent.

▶ **Lemma 6** (from [14])**.** *$EM \leq_p TkPM$ for polynomially bounded weights.*

This means that progress on TkPM not only provides further insight into EM, but could also help solve it directly. As previously mentioned, the main advantage of TkPM over the other studied variants of EM is that it is an optimization problem, i.e., we are maximizing a single objective function. This makes it more suitable for approximation and we provide approximation algorithms for it.

▶ **Theorem 7.** ($\star$) *There exists a deterministic polynomial time 0.5-approximation algorithm for TkPM.*

▶ **Theorem 8.** *There exists a deterministic polynomial time $(0.8 - 1/poly(n))$-approximation algorithm for TkPM on bipartite graphs.*

---

[3] The circumference of a graph is the length of any longest cycle in the graph.

[4] For two problems A and B, $A \leq_p B$ means that A is reducible to B in deterministic polynomial time and $A \equiv_p B$ implies both $A \leq_p B$ and $B \leq_p A$.

It is interesting to note that the main tool used for the proof of Theorem 1 (i.e., Proposition 11) was originally developed to prove Theorem 8. This shows how the study of TkPM can indeed provide insight into the EM problem.

Finally, the techniques we developed for FPT algorithms for EM so far only resulted in an FPT algorithm parameterized by the circumference of the graph. The circumference, however, is usually quite large and not very good as a parameter, so to better illustrate the use of these techniques, we combine them with techniques from [13] to show the existence of an FPT algorithm for TkPM parameterized by k and α (the independence number of the input graph), and an FPT algorithm for TkPM on bipartite graphs parameterized by k and β (the bipartite independence number of the input graph).

▶ **Theorem 9.** *There exists a deterministic algorithm for TkPM running in time $f(k,\alpha)poly(n)$ where $f(k,\alpha) = (k4^{\alpha})^{O(k4^{\alpha})}$ and α is the independence number of the input graph.*

▶ **Theorem 10.** ($\star$) *There exists a deterministic algorithm for TkPM on bipartite graphs running in time $f(k,\beta)poly(n)$ where $f(k,\beta) = (k\beta)^{O(k\beta)}$ and β is the bipartite independence number of the input graph.*

1.3 Organization of the paper

The remainder of this paper is organized as follows: In Section 2 we present the basic definitions and conventions we use throughout the paper. In Section 3 and Section 4 we study EM and TkPM respectively, both from the perspectives of approximation and parameterized algorithms. Finally in Section 5 we conclude the paper and provide some open problems.

2 Preliminaries

Due to space restrictions, proofs of statements marked ($\star$) have been deferred to the full version of this paper [12]. All graphs considered are simple. For a red/blue edge colored graph G and G' a subgraph of G, we define $R(G')$ (resp. $B(G')$) to be the set of red (resp. blue) edges in G' and $w(G')$ to be the sum of the weights of edges in G'. Undirected cycles are considered to have an arbitrary orientation. For a cycle C and $u, v \in C$, $C[u,v]$ is defined as the path from u to v along C (in the fixed but arbitrarily chosen orientation if C is undirected). Given a matching M, C is called M-alternating if for any two adjacent edges in C, one of them is in M and the other is not. An e edge is called a matching edge if $e \in M$ and a non-matching edge if $e \notin M$.

We always assume that for problems on weighted graphs, the input weights are given as positive integers and their encoding size is part of the input (i.e., they can be at most exponential in the input size if they are encoded in binary). We use w to refer to the set of weights in a weighted graph $G = (V, E, w)$. We always consider a strict ordering on the edges in which the edges are ordered by decreasing weight with ties broken arbitrarily (but the ordering is fixed for a given graph and weight function). The top-k weight function $w^k(E)$ for a set of edges E is defined as the sum of the first k edges from E in the edge ordering of the graph, i.e., $w^k(E) = \sum_{i\in\{1\dots k\}} w(E(i))$ where $E(i)$ is the i-th edge from E in the edge ordering of the graph.

3 Exact Matching

3.1 Approximation Algorithms

In this section, we aim to prove Theorem 1 by developing a deterministic polynomial time algorithm for EM where we require the output to be a perfect matching (abbreviated PM) and allow for a constraint violation that is a constant fraction of k. More precisely we require the output PM to have between $0.5k$ and $1.5k$ red edges. The main tool we use is the following proposition which allows us to increase the number of red edges of a PM without adding too many such edges.

▶ **Proposition 11.** *Let $G := (V, E)$ be an edge weighted directed graph containing a directed cycle C with $w(C) > 0$ and C contains at most k edges having strictly positive weight. There exists a deterministic polynomial time algorithm that, given G, finds a directed cycle in G with the same properties as C.*

Proof of Proposition 11. For simplicity, we will flip the sign of all weights so that we are looking for a negative cycle which can be found by a shortest path algorithm. In the following we will use the Bellman-Ford algorithm which relies on a dynamic program (DP) to compute the distance between any two nodes in the graph [3]. By adding an extra constraint variable to the DP, we are also able to compute the shortest path weights for paths that fulfill some bound on the constraint. More formally we start with the normal update rule for the Bellman-Ford algorithm:

$$d(s, v) = \min_{u \in V}\{d(s, u) + \bar{w}(u, v) | (u, v) \in E\}$$

where $\bar{w}(u, v) = -w(e)$ (i.e., we flip the sign of the weights) for $e = (u, v)$ and $d(s, v)$ is the distance from s to v where the length of an edge is given by its weight $\bar{w}$ (note that every vertex is considered to have a self loop of weight 0). We modify it to include the constraint variable (with an extra dimension in the table entries of the DP to account for it):

$$d(s, v, c) = \min_{u \in V}\{d(s, u, c - \mathbb{1}_{\bar{w}(u,v)<0}) + \bar{w}(u, v) | (u, v) \in E\}$$

where $\mathbb{1}$ is the indicator variable which takes value 1 if the condition is true and 0 otherwise, so the constraint variable is decreased every time the path uses a red edge. The entries $d(s, v, c)$ are initialized to ∞ for all $s, v \in V(G)$ and $c \in \{-1, 0, 1, 2, ..., k\}$, except for the entries of the form $d(s, s, c)$, for all $s \in V(G)$ and $c \in \{0, 1, 2, ..., k\}$, which are initialized to 0. This way, after running the update rule on the DP until convergence or until some entry of the form $d(s, s, c)$ becomes negative (i.e., a strictly negative cycle is detected), the value of $d(s, v, c)$ corresponds to the weight of the shortest path from s to v, containing at most c red edges, if such a path exists and is ∞ otherwise, unless there is a negative cycle. Note that the table entries can be computed iteratively, starting with entries of the form $d(s, v, 0)$ (the computation is the same as the regular Bellman-ford algorithm but with the new update rule) then increasing c by 1 every time. Finally observe that if a strictly negative cycle C containing at most k edges of strictly negative weight (i.e., positive in the original edge weight before the sign flip) exists, at least one of the entries of the form $d(s, s, c)$ for $s \in C$ and $0 \leq c \leq k$ will become negative (since the shortest path from s to itself should have negative length). Such a cycle is guaranteed by the conditions of the proposition and computing it can be done by a standard modification of the DP that keeps track of the last used edge for each updated entry. The output cycle is guaranteed to be strictly positive and have at most k strictly positive weight edges (in the original graph before the sign flip). Note that the running time of the DP is polynomial in the number of table entries, which in turn is polynomial in the size of the input graph. ◀

By repeatedly applying Proposition 11 we are able to find a PM fulfilling the requirements of Theorem 1.

Proof of Theorem 1. Let M be a PM containing a minimum number of red edges (should be at most k since we have a "Yes" instance). Note that M can be computed in polynomial time by simply using a maximum weight perfect matching algorithm [11], with -1 weights assigned to red edges and 0 weights assigned to blue edges. If $|R(M)| \geq 0.5k$ we are done, so suppose $|R(M)| < 0.5k$. We define the directed graph G' in the following way. We start with the bipartite input graph $G = (A \cup B, E)$ and orient the edges as follows: edges in M are oriented from A to B and edges not in M are oriented from B to A. This way we are guaranteed that any directed cycle in the resulting graph is an M-alternating cycle. We also define edge weights as follows: blue edges get weight 0, red edges in M get weight -1 and red edges not in M get weight $+1$. This way we have that for any M-alternating cycle C, $M' := M\Delta C$ is a perfect matching with $|R(M')| = |R(M)| + w(C)$. Note that $M\Delta M'$ is a set of disjoint cycles that are both M-alternating and M'-alternating.

Let M^* be a solution to the EM instance, i.e., $|R(M^*)| = k$ (which must exist since we are given a "Yes" instance). Observe that $w(M\Delta M^*) = |R(M^*)| - |R(M)| > 0$ so there must be a cycle $C \in M\Delta M^*$ s.t. $w(C) > 0$. Also note that M^* contains exactly k red edges so $M\Delta M^*$ contains at most k red edges not in M (i.e., edges of strictly positive weight). Finally note that the cycle C is a directed cycle (since it is alternating). So we can use Proposition 11 on the resulting graph to find a cycle C' with $w(C') > 0$ containing at most k edges of strictly positive weight. Note that $w(C') \leq k$ since edges have weight at most $+1$. Now we let $M' := M\Delta C'$ (this is possible since C' being a directed cycle implies that it must be an M-alternating cycle). Note that $|R(M)| < |R(M)| + w(C') \leq |R(M)| + k < 1.5k$ and $|R(M')| = |R(M\Delta C')| = |R(M)| + w(C')$. So if $|R(M')| \geq 0.5k$ the algorithm stops and outputs $M\Delta C'$, otherwise we repeat the above procedure, with M' replacing M, until $|R(M')| \geq 0.5k$. The running time is polynomial since the above procedure runs in polynomial time (by Proposition 11) and it is repeated at most k times. ◀

3.2 FPT Algorithms

In this section we start by proving Proposition 3 which provides a new tool for finding alternating cycles with color and weight constraints in FPT time parameterized by the size of the cycles.

Proof of Proposition 3. Our goal is to find a set of M-alternating disjoint cycles $\mathcal{C}'$ in G with the same number of matching (i.e., edges in M) and non-matching (i.e., edges not in M) red edges as $\mathcal{C}$ and weights that are at least as big, i.e., for every $C \in \mathcal{C}$ there must be a $C' \in \mathcal{C}'$ such that C' has the same number of matching and non-matching red edges as C and $w(C') \geq w(C)$ (and vice versa, i.e., there is a one to one correspondence between the cycles in $\mathcal{C}$ and the cycles in $\mathcal{C}'$). This way we construct $M'' := M\Delta\mathcal{C}'$ s.t. $w(M'') = w(M\Delta\mathcal{C}') \geq w(M\Delta\mathcal{C}) = w(M')$ and $|R(M'')| = |R(M\Delta\mathcal{C}')| = |R(M\Delta\mathcal{C})| = |R(M')|$.

Color Coding. The main tool for finding such a set of cycles is color coding [1]. The idea is to color all vertices at random with L colors. The probability that all vertices of $\mathcal{C}$ get different colors is only a function of L. This can also be achieved deterministically using a perfect hash family of size bounded by $L^{O(L)}poly(n)$, which can be guaranteed to contain at least one coloring for which all vertices of $\mathcal{C}$ have different colors (see [10] Chapter 5 for more details on derandomizing color coding).

Separating the Cycles. Observe that for every cycle in $C \in \mathcal{C}$, the following can also be achieved in $L^{O(L)}poly(n)$ time.

- Guess the set of colors $colors(C)$ of its vertices and their exact order.
- Guess its number of matching (i.e., in M) and non-matching (i.e., not in M) red edges.

Let G_C be the graph induced on the vertices of G with a color from the set $colors(C)$. Observe that C is contained in G_C and that the subgraphs G_C for $C \in \mathcal{C}$ are all disjoint. So we can look for each cycle separately.

Orienting the Cycles. Since we know the colors of the vertices of $\mathcal{C}$, we can define a bipartition (A, B) of G by splitting the set of colors into two equal parts and letting A (resp. B) be the vertices having a color from the first (resp. second), s.t. the cycles in $\mathcal{C}$ are alternating with respect to the bipartition (note that this is indeed possible since the cycles in $\mathcal{C}$ are M-alternating so they have even length). By deleting all edges with endpoints in the same part, we get a bipartite graph which contains $\mathcal{C}$. Now we can define the following orientation for the edges: edges in M are oriented from A to B and edges not in M are oriented from B to A. This way we are guaranteed that any directed cycle in the resulting graph is an alternating cycle.

From Cycles to Colorful Paths. For this part and the next, we look into one cycle $C \in \mathcal{C}$ and its corresponding subgraph G_C. Let $(c_1, c_2, ...) := colors(C)$. We first guess the edge of C with start vertex from color class $c_{|colors(C)|}$ and end vertex from color class c_1 (this can be done in polynomial time by trying all possibilities). Then we delete all edges from G_C except for the edges going from a vertex of color c_i to a vertex of color c_{i+1} for $i \in \{1, 2, ..., |colors(C)| - 1\}$. Observe that G_C is now acyclic and the remaining edges of C form a directed path from s to t in G_C.

Finding the Paths. For simplicity, we will flip the sign of all weights so that we are looking for paths of minimum weight which can be found by a shortest path algorithm. Similarly to the proof of Theorem 1 we use a modified Bellman-Ford algorithm, so we will focus on the main difference, i.e., the update rule. By adding extra constraint variables to the DP, we are also able to compute the shortest path weights for paths that fulfill an exact constraint. Note that this is only possible since the graph is acyclic (otherwise the algorithm can output non-simple paths). More formally the update rule for the Bellman-Ford algorithm is the following:

$$d(s, v, rm, rn) = \min_{u \in V}\{d(s, u, rm - \mathbb{1}_{(u,v)\in R_m}, rn - \mathbb{1}_{(u,v)\in R_n}) + \bar{w}(u, v) | (u, v) \in E\}$$

where $G_C = (V, E)$, $\bar{w}(u, v) = -w(e)$ for $e = (u, v)$, R_m is the set of matching red edges in G_C and R_n is the set of non-matching red edges in G_C. We are interested in the value $d(s, t, |(C\backslash(t, s)) \cap R_m|, |(C\backslash(t, s)) \cap R_n|)$ where $|(C\backslash(t, s)) \cap R_m|$ is the number of matching red edges in $C\backslash(t, s)$ and $|(C\backslash(t, s)) \cap R_n|$ is the number of non-matching red edges in $C\backslash(t, s)$. The DP runs in polynomial time since the number of table entries is polynomial and allows us to find a simple path $path(C)$ (since the graph is acyclic) from s to t of minimum weight, i.e., $\bar{w}(path(C)) \leq \bar{w}(C\backslash(t, s))$ which implies $w(path(C)) \geq w(C\backslash(t, s))$, and with the same number of matching and non-matching red edges as $(C\backslash(t, s))$.

Constructing the set $\mathcal{C}'$. Finally for $C \in \mathcal{C}$, let $cycle(C) := path(C) \cup (t, s)$ with $path(C)$ computed using the above DP on the processed graph G_C. Let $\mathcal{C}' := \{cycle(C) | C \in \mathcal{C}\}$. Observe that $cycle(C)$ is a cycle with the same number of matching and non-matching red edges as C, $w(cycle(C)) \geq w(C)$ and all cycles in $\mathcal{C}'$ are M-alternating and disjoint. So $\mathcal{C}'$ fulfills all the required properties.

Running Time. Observe that all the above steps can be run in $L^{O(L)}poly(n)$ time, so the total running time is of the same order. ◀

The above proposition is the key to proving Theorem 4 which gives an FPT algorithm for EM parameterized by the circumference of the graph. But first we need the following lemma which ensures that we can always make progress using a set of alternating cycles of size bounded by a function of their individual lengths.

▶ **Lemma 12.** *Given a "Yes" instance of EM and a PM M, if $|R(M)| < k$ then there exists a set of disjoint M-alternating cycles $\mathcal{C}$ s.t. $|E(\mathcal{C})| \leq 2c^4$, where c is circumference of the graph, and $|R(M)| < |R(M\Delta\mathcal{C})| \leq k$.*

Proof of Theorem 4. Let M be a PM containing a minimum number of red edges (should be at most k). Note that M can be computed in polynomial time by simply using a maximum weight perfect matching algorithm with -1 weights assigned to red edges and 0 weights assigned to blue edges. From Lemma 12 we know that there exists a set of disjoint M-alternating cycles $\mathcal{C}$ s.t. $|E(\mathcal{C})| \leq 2c^3$ and $|R(M)| < |R(M\Delta\mathcal{C})| \leq k$. Let $M' := M\Delta\mathcal{C}$. Now by using Proposition 3 we can find a PM M'' with $|R(M'')| = |R(M')|$ (here we do not need to assign any weights to edges so the weight function used to apply the proposition can simply be uniform) so $|R(M)| < |R(M'')| \leq k$. We can repeat the procedure (applying Lemma 12 on M'') until we get a PM with exactly k red edges. We need at most k repetitions, each running in time $f(L)poly(n) = L^{O(L)}poly(n)$ for $L = O(c^3)$, i.e., we get an FPT algorithm parameterized by c. ◀

Theorem 4 illustrates the use the Proposition 3 to develop FPT algorithms for Exact Matching. However, the circumference of the graph can in general be quite large. We believe that Proposition 3 can be applied to get other more interesting FPT algorithms for EM and related problems. In Section 4.4 we show one such application.

4 Top-k Perfect Matching

In this section we study TkPM which, as we show later, is polynomial time equivalent to EM, making it another problem that can be used to test the hypothesis $\mathbf{P} = \mathbf{RP}$, but with the advantage of being an optimization problem.

4.1 Minimum Weight Variant

First, we start by introducing a variant of TkPM in which we are looking for a PM minimizing (instead of maximizing) the top-k weight. This objective function has been studied in the context of other problems such as k-clustering and load balancing [8] but to our knowledge, no prior work considered it in the context of matching problems.

Minimum Weight Top-k Perfect Matching (minTkPM)
Input: A weighted graph G and integer k.
Task: Find a perfect matching in G minimizing the top-k weight function.

We show however, that by simply applying a threshold to the weights of the edges, we are able to reduce this problem to minimum weight perfect matching (minWPM), i.e., it is in $\mathbf{P}$. The proof crucially relies on the idea of thresholding the weights which will also be useful for the approximation algorithms in the next sections.

▶ **Definition 13.** *Given a weighted graph G with weights w, the thresholded weights w_t for a threshold t are defined as follows: for an edge e, $w_t(e) = \max(w(e) - t, 0)$.*

▶ **Theorem 14.** *$minTkPM \equiv_p minWPM$.*

Proof. $minWPM \leq_p minTkPM$ is trivial by setting $k = n/2$ so we need to prove $minTkPM \leq_p minWPM$. Given an instance of minTkPM, let M^* be an optimal PM. Let e_k be the kth edge from M^* in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of smallest top-k value among all solutions) and setting $t := w(e_k)$. We have $w_t(M^*) = w_t^k(M^*) = w^k(M^*) - kt$ since the k values above t are reduced by t, and the rest is set to 0. Now let M be a minimum weight perfect matching in the thresholded graph. Then we have $w_t^k(M) \leq w_t(M) \leq w_t(M^*) \leq w(M^*) - kt$. After removing the threshold, each of the top-k values can only increase by at most t, so we get $w^k(M) \leq w(M^*)$, i.e., we get an optimal solution. ◀

This creates an interesting division between the minimization and maximization of the top-k values, in the context of a perfect matching problem. On the one hand we have a problem that is polynomially equivalent to the general weighted matching problem (known to be in **P**), and on the other hand we get a problem that is polynomially equivalent to EM (as we show next) whose complexity remains unknown.

4.2 Reducing Top-k Perfect Matching to Exact Matching

To help reduce TkPM to EM, we introduce an intermediary problem called maximum weight EM in which we are given an instance of EM as well as edge weights (of polynomial size) and the goal is to find a PM with exactly k red edges having maximum weight among all such PMs.

Maximum Weight Exact Matching (MWEM)
Input: An edge-weighted and edge-colored (with red/blue colors) graph G and integer k.
Task: Find a perfect matching M in G with exactly k red edges and having maximum weight among all such matchings.

We show that this new variant can be reduced to EWPM (with polynomial weights) which in turn can be reduced to EM. This shows that MWEM is in **RP**.

▶ **Lemma 15.** (⋆) *$MWEM \leq_p EWPM \leq_p EM$ for polynomially bounded weights. The reductions also work for bipartite input graphs and for minor closed graph classes.*

Note that even though MWEM is an optimization problem, any approximation for it requires solving EM. So our focus will instead be on TkPM which we reduce to EM when the input weights are polynomially bounded in the input size.

Proof of Theorem 5. We have $MWEM \leq_p EM$ from Lemma 15, so we need to show that $TkPM \leq_p MWEM$. Given an instance of TkPM, let M^* be an optimal solution. Let e_k be the kth edge from M^* in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of highest top-k value among all solutions) and setting the weights of all edges after e_k in the ordering to 0 and coloring them blue, while the rest of the edges are colored red. Note that only red edges can have non-zero weights and that M^* has exactly k red edges. Let M be the output of an algorithm for MWEM on the resulting graph. By optimality of M, we have that $w(M) \geq w(M^*)$, and since they both contain at most k non-zero weight edges we get $w^k(M) \geq w^k(M^*)$ so M is an optimal solution for TkPM. Since we only modify the weights of the edges, the reduction preserves the graph class. ◀

The above lemma, in combination with the result of [14], implies the following theorem.

▶ **Theorem 16.** *TkPM $\equiv_p$ EM for polynomially bounded weights. The equivalence also holds for bipartite input graphs and for minor closed graph classes.*

Note that it is still open whether MWEM and TkPM with exponential weights are reducible to EM or if they are **NP**-hard.

4.3 Approximation Algorithms for Top-k Perfect Matching

Note that the reduction to EM in Lemma 15 does not preserve any approximation factor since it changes the weights of the edges. So we cannot use it in combination with Theorem 1 to get an approximation algorithm for TkPM. We will, however, use Proposition 11 to get a better approximation for TkPM as we will see later. First we show that by simply applying a specific threshold to the weights of the graph, any maximum weight perfect matching (maxWPM) algorithm can output a 0.5-approximation for TkPM.

▶ **Lemma 17.** *Given an instance of TkPM, let M^* be an optimal solution. There exists a threshold t such that for any maximum weight perfect matching M in the thresholded graph, we have $w^k(M) \geq 0.5 \cdot w^k(M^*)$ (in the original graph).*

Proof. Let $t = \frac{w^k(M^*)}{2k}$. Then we have $w_t(M^*) \geq w^k(M^*) - k \cdot \frac{w^k(M^*)}{2k} = 0.5w^k(M^*)$. And since M is a maximum weight perfect matching, we have $w_t(M) \geq w_t(M^*) \geq 0.5w^k(M^*)$. Let k' be the number of edges $e \in M$ with $w_t(e) > 0$. Now we have two cases. First, if $k' \leq k$ then we have $w^k(M) \geq w_t^k(M) \geq 0.5w^k(M^*)$. Otherwise the output matching contains at least k edge of weight more than $\frac{w^k(M^*)}{2k}$, so the total weight is $w^k(M) \geq k \cdot \frac{w^k(M^*)}{2k} = 0.5w^k(M^*)$. ◀

The above lemma guarantees the existence of a threshold that will lead to a 0.5-approximation using any maximum weight PM algorithm. We may not know the exact threshold, but if the weights are polynomial we can simply try all possibilities. Otherwise we can find a good threshold using binary search (see Algorithm 1).

In order to get a better approximation factor, we rely on Proposition 11 which allows us to limit the change in the number of edges with weight above threshold.

Proof Sketch of Theorem 8. We start with a high level intuition on how the algorithm works and why it gives a better approximation.

The core idea of the algorithm is the following: instead of recomputing the maximum weight perfect matching every time we change the threshold (as is done in the previous algorithm), we keep track of one perfect matching M which we incrementally improve using alternating cycles that increase its weight. We also make sure that the cycles do not add too many edges of weight above the threshold. This way the top-k weight of M stays closer to its total weight. To find such cycles, we rely on the algorithm of Proposition 11, which allows us to find positive alternating cycles that do not add too many positive weight edges (at most k). But first we set the edge weights of edges in M to negative (i.e., multiply them by -1). This way the weight of an alternating cycle indicates the total weight change we get when taking its symmetric difference with M to get a new perfect matching. This means that, whenever possible, we can increase the total weight of M while keeping its top-k weight close to its total weight (considering the thresholded weights) since the number of positive weight edges above the threshold is limited.

Algorithm 1 TkPM 0.5-approximation Algorithm.

Input: An instance of TkPM.
Output: PM M with $w^k(M) \geq 0.5w(M^*)$ where M^* is an optimal solution.
$t_1 \leftarrow 0$, $t_2 \leftarrow w_{max}$; `/* where` w_{max} `is the maximum weight in the graph */`
$M \leftarrow$ MaximumWeightPerfectMatching(G, w_{t_1});
if *M contains at most k edges e with $w(e) > 0$* **then**
  return M;
else
  $M_1 \leftarrow$ MaximumWeightPerfectMatching(G, w_{t_1});
  $M_2 \leftarrow$ MaximumWeightPerfectMatching(G, w_{t_2});
  while $t_2 - t_1 \geq 1/(k^2)$ **do**
    $t \leftarrow (t_1 + t_2)/2$;
    $M \leftarrow$ MaximumWeightPerfectMatching(G, w_t);
    if *M contains more than k edges e with $w_t(e) > 0$* **then**
      $t_1 \leftarrow t$; $M_1 \leftarrow M$;
    else
      $t_2 \leftarrow t$; $M_2 \leftarrow M$;
  $M \leftarrow$ BestOf(M_1, M_2);
  return M;

Procedure *BestOf(M_1, M_2)*:
  if $w^k(M_1) \geq w^k(M_2)$ **then**
    return M_1;
  else
    return M_2;

To see why this is helpful, consider the two cases in the proof of Lemma 17: $k' \leq k$ and $k' \geq k$ (remember that k' is the number of edges in M with weight strictly above the threshold).

In the case $k' \leq k$ (let $k_1 := k'$), we know that the top-k weight is the same as the total weight (for the thresholded weights), which means that we do not lose anything when considering only the top-k weight (i.e., $w_t^k(M) = w_t(M) \geq w_t(M^*) = w_t^k(M^*)$). However, we might lose some value because of the threshold. This is because when we go back to the original weights, M^* regains up to $k \cdot t$ in value (k times the threshold, since all its top-k edges might have value above the threshold) whereas M might only regain $k_1 \cdot t$ (since all other edges could have original weight close to zero). So in the case $k_1 << k$, we lose almost all the value from the threshold.

On the other hand, in case $k' \geq k$ (let $k_2 := k'$), M will also regain $k \cdot t$ in top-k weight when we add back the threshold. However, the top-k weight of M can be far from its total weight since many edges can be contributing to the total weight. This is mainly a problem when $k_2 >> k$. If k' is close to k, however, this loss is not so big (at most a fraction $(k_2 - k)/k$ of the total since the k highest weights still count).

To get a worst case approximation factor of 0.5, it must be the case that both $k_1 << k$ and $k_2 >> k$. Note, however, that the procedure detailed above (relying on Proposition 11) allows us to bound the difference between k_1 and k_2 by k (i.e,. $k_2 - k_1 \leq k$). This way, the algorithm manages to guarantee a better approximation factor.

We are now ready to describe the full algorithm. We will first show how to transform exponential weights into polynomially bounded ones while loosing at most a factor of $1 - 1/poly(n)$ in the approximation. We then provide a $0.8 - 1/poly(n)$-approximation algorithm for TkPM with polynomially bounded weights, which proves the theorem.

Dealing with Exponential Weights. For exponential size weights, we first scale and round them to make them bounded by a polynomial function of the input size. We start by deleting all edges that cannot be part of any perfect matching (this can simply be done by checking for every edge whether we could remove it along with its endpoints from the graph and still be able to get a perfect matching on the rest of the graph). Let W be the highest edge weight in the remaining graph. Observe that for $k \geq 1$ an optimal solution M^* to the top-k perfect matching problem must have $w^k(M^*) \geq W$ (since the perfect matching containing the edge of weight W is a valid solution). Now all weight encodings have at most $\log_2(W)$ non-zero bits. Let $f(n) = n \cdot poly(n)$ for any desired polynomial. If $W \leq f(n)$ then all weights are polynomial. Otherwise we re-encode the weights of all edges by only considering their $(\log_2(W) - \log_2(f(n)))$-th to $(\log_2(W))$-th bits (counting from the least significant bit) and dropping all others. We call these weights w'. So all weights are now encoded with at most $\log_2(f(n)) + 1$ bits, i.e., are bounded by a polynomial function of the input size. Now let M' be a $0.8 - 1/poly(n)$-approximation for TkPM on the graph with weights w'. Observe that for any edge e, $|w(e) - w'(e) \cdot \frac{W}{f(n)}| \leq \frac{W}{f(n)}$ (the rounding error). Since a perfect matching contains $n/2$ edges we get

$$w^k(M') \geq w'^k(M') \cdot \frac{W}{f(n)} - \frac{nW}{2f(n)} \geq w'^k(M') \cdot \frac{W}{f(n)}(1 - \frac{n}{f(n)}).$$

The last inequality resulting from the fact that the optimal solution has weight at least $W \geq f(n)$ so $w'^k(M') \geq f(n)/2$. Now since M' is a $0.8 - 1/poly(n)$-approximation we get

$$w^k(M') \geq (0.8 - 1/poly(n)) \cdot w'^k(M^*) \cdot \frac{W}{f(n)}(1 - \frac{n}{f(n)}) \geq (0.8 - 2/poly(n)) \cdot w'^k(M^*) \cdot \frac{W}{f(n)}.$$

Going back to the original weights, we get

$$w^k(M') \geq (0.8 - 2/poly(n)) \cdot (w^k(M^*) - \frac{nW}{2f(n)}).$$

Finally, using $w^k(M^*) \geq W \geq f(n)$ we get

$$w^k(M') \geq (0.8 - 3/poly(n))w^k(M^*).$$

Approximation algorithm for polynomial weights. We start with a preprocessing of the edge weights. Given an instance of TkPM, let M^* be an optimal solution. Let e_k be the kth edge from M^* in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of highest top-k value among all solutions) and setting the weights of all edges after e_k in the ordering to 0. This means that some solution M^* will contain at most k edges of strictly positive weight. Note that this does not modify the top-k weight of M^* and can only decrease the top-k weight of any matching, i.e., if we find a $0.8 - 1/poly(n)$-approximation on the new weights, it is also a $0.8 - 1/poly(n)$-approximation on the original weights. Also note that now we have $w^k(M^*) = w(M^*)$. For ease of notation, let $\epsilon = 1/poly(n)$ for the desired polynomial function in the approximation factor. The algorithm will have two phases. In the first phase, we use a slight modification of Algorithm 1

(we call it ModifiedTkpmApprox in Algorithm 2) where instead of returning M, it returns M_1, M_2 and t_2. We label them M'_0, M_0 and t_0 respectively. This way we are guaranteed that M_0 has at most k edges with $w_{t_0}(e) > 0$ and M'_0 has at least k edges with $w_{t_0-\epsilon/k}(e) > 0$, i.e., we have one matching with less then k edges of weight strictly more than t_0 and one with more than k edges of weight strictly more than $t_0 - \epsilon/k$. Algorithm 1 also guarantees that both matchings have maximum total weight with respect to their thresholds.

The second phase only works for bipartite graphs since it will rely on the algorithm of Proposition 11 (see ImproveUsingBoundedCycles in Algorithm 2) to increase the weight of the matching instead of recomputing the maximum weight perfect matching from scratch whenever we decrease the threshold (as is done in Algorithm 1). This way we again search for a threshold t_1 such that ImproveUsingBoundedCycles fails to get a PM containing more than k edges e with $w_{t_1}(e) > 0$ and instead stops at a perfect matching M_{t_1} containing at most k edges e with $w_{t_1}(e) > 0$, but for threshold $t_2 = t_1 - \epsilon/k$ ImproveUsingBoundedCycles outputs two perfect matchings M_1 and M_2 such that M_2 has at most k edges e with $w_{t_2}(e) > 0$ and M_1 has more than k edges e with $w_{t_2}(e) > 0$.

The full algorithm is given in Algorithm 2. Note that for a graph $G := (A \cup B, E, w)$ and a perfect matching M, we define the directed graph $G_M := (A \cup B, E, w')$ where every edges in M is oriented from A to B and has weight $w'(e) = -w(e)$ and every edge not in M is oriented from B to A and has weight $w'(e) = w(e)$. This means that for $M' := M\Delta C$ where C is a directed cycle (which implies that it is M-alternating), M' is a PM with $w(M') = w(M) + w'(C)$. The proof of correctness and running time of the algorithm can be found in the extended version of this paper [12]. ◀

4.4 Parametrized Complexity of TkPM

In this section we show that TkPM can be solved in Fixed Parameter Tractable (FPT) time parameterized by k and α, the independence number of the graph (i.e., the size of the largest independent set). The algorithm mainly uses Proposition 3. To prove its correctness we will rely on what [13] defines as skip, which allows us to shorten alternating cycles using the bound on the independence number. This way we manage to bound the total number of edges, in a symmetric difference with some optimal solution, by a function of k and α. We start by adapting the following definition and lemma from [13].

▶ **Definition 18** (adapted from [13]). *Let C be a an M-alternating cycle. A skip S is a set of 2 non-matching edges $e_1 := (v_1, v_2)$ and $e_2 := (v'_1, v'_2)$ with $e_1, e_2 \notin C$ and $v_1, v'_1, v_2, v'_2 \in C$ s.t. $C' = e_1 \cup e_2 \cup C \setminus (C[v_1, v'_1] \cup C[v_2, v'_2])$ is an M-alternating cycle and $|C| - |C'| > 0$.*

▶ **Lemma 19** (adapted from [13]). *Let P be an M-alternating path containing a set $\mathcal{P}$ of disjoint paths, each of length at least 3 starting and ending at non-matching edges, and $|\mathcal{P}| \geq 4^\alpha$. Then P contains a skip.*

This allows us to prove the following.

▶ **Proposition 20.** ($\star$) *Let M be a PM in a weighted graph G of independence number α, with edge colors red and blue. Let $\mathcal{C}$ be a set of disjoint M-alternating cycles in G. Let $R \subseteq E(\mathcal{C})$ be the set of red edges in $\mathcal{C}$ and $k = |R|$. Then G must contain a set of disjoint M-alternating cycles $\mathcal{C}'$ s.t. $R \subseteq \mathcal{C}'$, $|E(\mathcal{C})| \leq kf(\alpha)$ for $f(\alpha) = 4^{\alpha+1}$ and all edges in $E(\mathcal{C}')\backslash E(\mathcal{C})$ are non-matching edges.*

▶ **Lemma 21.** ($\star$) *Given an instance of TkPM and a PM M, there exists an optimal solution M' s.t. $|E(M\Delta M')| \leq kf(\alpha)$ for $f(\alpha) = 4^{\alpha+1}$, where α is the independence number of the input graph.*

Algorithm 2 TkPM 0.8-approximation Algorithm.

Input: An instance of TkPM and $\epsilon := 1/poly(n)$ for some polynomial function.
Output: PM M with $w^k(M) \geq (0.8 - \epsilon)w(M^*)$ where M^* is an optimal solution.
for $e_k \in E(G)$ **do**
  for $e \in E(G)$ **do**
    if $e > e_k$ **then**
      $w(e) \leftarrow 0$;
  $M'_0, M_0, t_0 \leftarrow$ ModifiedTkpmApprox; $M_1, M_2 \leftarrow M_0$; $Success \leftarrow False$;
  while $Success == False$ *and* $t - \epsilon/k > 0$ **do**
    $t \leftarrow t - \epsilon/k$;
    $M_1, M_2, Success \leftarrow$ ImproveUsingBoundedCycles(M_1, t);
    if $Success$ **then**
      $t_2 \leftarrow t$; $t_1 \leftarrow t + \epsilon/k$;
    else
      $M_0 \leftarrow M_1$;
  $M \leftarrow$ BestOf(M, M_0, M'_0, M_1, M_2);
return M;

Procedure *ImproveUsingBoundedCycles*(M, t)**:**
  if M_1 *contains more than* k *edges* e *with* $w_t > 0$ **then**
    return $M_1, M_1, True$;
  else
    $M_1, M_2 \leftarrow M$;
    while M_1 *contains at most* k *edges* e *with* $w_t(e) > 0$ *and there exists a positive cycle in* G_{M_1}*, for threshold* t*, containing at most* k *edges* e *with* $w_t(e) > 0$ *and* $e \notin M_1$ **do**
      Use Proposition 11 to find a cycle C with the above properties;
      $M_2 \leftarrow M_1$; $M_1 \leftarrow M_2 \Delta C$;
    if M_1 *contains less than* k *edges* e *with* $w_t(e) > 0$ **then**
      return $M_1, M_1, False$;
    else
      return $M_1, M_2, True$;

Proof of Theorem 9. Given an instance of TkPM, we start by computing any perfect matching M. By Lemma 21, we know that there exists an optimal solution M' s.t. $|E(M \Delta M')| \leq kf(\alpha)$ for $f(\alpha) = 4^{\alpha+1}$. Let e_k be the kth edge from M' in the edge ordering. The algorithm starts by guessing e_k (i.e., running for all possibilities of e_k and outputting the matching of highest top-k value among all solutions) and setting the weights of all edges after e_k in the ordering to 0 and coloring them blue, while the rest of the edges are colored red. Note that this does not modify the top-k weight of M' and can only decrease the top-k weight of any other matching. Now we can use the algorithm of Proposition 3 to find a PM M'' with $w(M'') \geq w(M')$ and $|R(M'')| = |R(M')| = k$. Since only red edges have non-zero weight we get $w^k(M'') \geq w^k(M')$ so M'' is an optimal solution (since M' is an optimal solution). The running time is dominated by the algorithm of Proposition 3 which runs in time $L^{O(L)}$ where $L = |E(M \Delta M')| \leq k4^{\alpha+1}$. ◀

5 Conclusion and Open Problems

In the paper we study the Top-k perfect matching problem which is shown to be polynomially equivalent to the Exact Matching problem. In the course of developing approximation algorithms for this problem we also initiate a new direction of study for the EM problem where the goal is to minimize the constraint violation while requiring the output to be a perfect matching. We also continue the study of the parameterized complexity of EM that was initiated by [13]. To show the utility of these developments, we provide FPT algorithms for TkPM which rely on them.

Our work leaves open many questions, we list a few. Starting with questions related to EM, can we reduce the constraint violation in Theorem 1 to less than $0.5k$? The aim would be to get $o(k)$, but so far no constant improvement is known. Also, can we design an FPT algorithm for EM parameterized by k and α? This would be an intermediate step towards an FPT algorithm parameterized only by α and resolving the open problem in [13]. For TkPM, the first open question is whether the problem is **NP**-hard. We know it is unlikely for the case of polynomial sized input weights since we can reduce it to EM, but the exponential size weights case is still open. Another interesting problem is to get an FPT (or even XP) algorithm for TkPM parameterized only by α. Note that this is known for EM, but the reduction of TkPM to EM does not preserve the independence number of the graph. An FPT algorithm parameterized only by k, for either EM or TkPM would also be highly desirable. Finally an important step forward would be to improve the approximation algorithms for TkPM, with the goal of getting a polynomial time approximation scheme.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995.

2 Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Jacobo Torán. Solving linear equations parameterized by hamming weight. *Algorithmica*, 75(2):322–338, 2016.

3 Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

4 André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. *Mathematical Programming*, 128(1):355–372, 2011.

5 Jacek Błażewicz, Piotr Formanowicz, Marta Kasprzak, Petra Schuurman, and Gerhard J Woeginger. A polynomial time equivalence between DNA sequencing and the exact perfect matching problem. *Discrete Optimization*, 4(2):154–162, 2007.

6 Daniel P Bovet and Pierluigi Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall International (UK) Ltd., GBR, 1994.

7 Paolo M. Camerini, Giulia Galbiati, and Francesco Maffioli. Random pseudo-polynomial algorithms for exact matroid problems. *Journal of Algorithms*, 13(2):258–273, 1992.

8 Deeparnab Chakrabarty and Chaitanya Swamy. Approximation algorithms for minimum norm and ordered optimization problems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 126–137, 2019.

9 Radu Curticapean and Mingji Xia. Parameterizing the permanent: Hardness for K_8-minor-free graphs. *arXiv preprint*, 2021. `arXiv:2108.12879`.

10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

11 Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

12 Nicolas El Maalouly. Exact matching: Algorithms and related problems. *arXiv preprint*, 2022. `arXiv:2203.13899`.

13 Nicolas El Maalouly and Raphael Steiner. Exact Matching in Graphs of Bounded Independence Number. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 46:1–46:14, 2022.

14 Nicolas El Maalouly and Lasse Wulf. Exact matching and the top-k perfect matching problem. *arXiv preprint*, 2022. `arXiv:2209.09661`.

15 Dennis Fischer, Tim A Hartmann, Stefan Lendl, and Gerhard J Woeginger. An investigation of the recoverable robust assignment problem. *arXiv preprint*, 2020. `arXiv:2010.11456`.

16 Anna Galluccio and Martin Loebl. On the theory of Pfaffian orientations. I. Perfect matchings and permanents. *Electronic Journal of Combinatorics*, 6:R6, 1999.

17 Hans-Florian Geerdes and Jácint Szabó. A unified proof for karzanov's exact matching theorem. Technical Report QP-2011-02, Egerváry Research Group, Budapest, 2011. `egres.elte.hu`.

18 Christopher David Godsil. *Algebraic combinatorics*. Routledge, 2017.

19 Fabrizio Grandoni and Rico Zenklusen. Optimization with more than one budget. *arXiv preprint*, 2010. `arXiv:1002.2147`.

20 Rohit Gurjar, Arpita Korwar, Jochen Messner, Simon Straub, and Thomas Thierauf. Planarizing gadgets for perfect matching do not exist. In *International Symposium on Mathematical Foundations of Computer Science*, pages 478–490. Springer, 2012.

21 Rohit Gurjar, Arpita Korwar, Jochen Messner, and Thomas Thierauf. Exact perfect matching in complete graphs. *ACM Transactions on Computation Theory (TOCT)*, 9(2):1–20, 2017.

22 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *computational complexity*, 13(1-2):1–46, 2004.

23 AV Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics*, 23(1):8–13, 1987.

24 Monaldo Mastrolilli and Georgios Stamoulis. Constrained matching problems in bipartite graphs. In *International Symposium on Combinatorial Optimization*, pages 344–355. Springer, 2012.

25 Monaldo Mastrolilli and Georgios Stamoulis. Bi-criteria and approximation algorithms for restricted matchings. *Theoretical Computer Science*, 540:115–132, 2014.

26 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

27 Christos H Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *Journal of the ACM (JACM)*, 29(2):285–309, 1982.

28 Irena Rusu. Maximum weight edge-constrained matchings. *Discrete applied mathematics*, 156(5):662–672, 2008.

29 Georgios Stamoulis. Approximation algorithms for bounded color matchings via convex decompositions. In *International Symposium on Mathematical Foundations of Computer Science*, pages 625–636. Springer, 2014.

30 Ola Svensson and Jakub Tarnawski. The matching problem in general graphs is in quasi-nc. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 696–707. Ieee, 2017.

31 Tongnyoul Yi, Katta G Murty, and Cosimo Spera. Matchings in colored bipartite networks. *Discrete Applied Mathematics*, 121(1-3):261–277, 2002.

32 Raphael Yuster. Almost exact matchings. *Algorithmica*, 63(1):39–50, 2012.

Counting Temporal Paths

Jessica Enright
School of Computing Science, University of Glasgow, UK

Kitty Meeks
School of Computing Science, University of Glasgow, UK

Hendrik Molter
Department of Computer Science and Department of Industrial Engineering and Management, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Abstract

The betweenness centrality of a vertex v is an important centrality measure that quantifies how many optimal paths between pairs of other vertices visit v. Computing betweenness centrality in a temporal graph, in which the edge set may change over discrete timesteps, requires us to count temporal paths that are optimal with respect to some criterion. For several natural notions of optimality, including *foremost* or *fastest* temporal paths, this counting problem reduces to #Temporal Path, the problem of counting *all* temporal paths between a fixed pair of vertices; like the problems of counting foremost and fastest temporal paths, #Temporal Path is #P-hard in general. Motivated by the many applications of this intractable problem, we initiate a systematic study of the parameterised and approximation complexity of #Temporal Path. We show that the problem presumably does not admit an FPT-algorithm for the feedback vertex number of the static underlying graph, and that it is hard to approximate in general. On the positive side, we prove several exact and approximate FPT-algorithms for special cases.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Approximation algorithms analysis; Mathematics of computing → Discrete mathematics

Keywords and phrases Temporal Paths, Temporal Graphs, Parameterised Counting, Approximate Counting, #P-hard Counting Problems, Temporal Betweenness Centrality

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.30

Related Version *Full Version*: `https://arxiv.org/abs/2202.12055`

Funding *Jessica Enright*: Supported by EPSRC grant EP/T004878/1.
Kitty Meeks: Supported by EPSRC grants EP/T004878/1 and EP/V032305/1.
Hendrik Molter: Supported by the ISF, grants No. 1456/18 and No. 1070/20, and European Research Council, grant number 949707.

Acknowledgements This work was initiated at the Dagstuhl Seminar "Temporal Graphs: Structure, Algorithms, Applications" (Dagstuhl Seminar Nr. 21171).

1 Introduction

Computing a (shortest) path between two vertices in a graph is one of the most important tasks in algorithmic graph theory and serves as a subroutine in a wide variety of algorithms for connectivity-related graph problems. The *betweenness centrality* measure for vertices in a graph was introduced by Freeman [23] and motivates the task of *counting* shortest paths in a graph. Intuitively, betweenness centrality measures the importance of a vertex for information flow under the assumption that information travels along optimal (i.e. shortest) paths. More formally, the betweenness of a vertex v is based on the ratio of the number of shortest paths between vertex pairs that visit v as an intermediate vertex and the total number of shortest paths, thus its computation is closely related to shortest path counting.


© Jessica Enright, Kitty Meeks, and Hendrik Molter;
licensed under Creative Commons License CC-BY 4.0


The betweenness centrality is a commonly used tool in network analysis and it can be computed in polynomial time; e.g. Brandes' algorithm [9] serves as a blueprint for all modern betweenness computation algorithms and implicitly also counts shortest paths.

In contrast to the tractability of counting shortest paths, the problem of counting *all* paths between two vertices in a graph is one of the classic problems discussed in the seminal paper by Valiant [46] that is complete for the complexity class #P (the counting analogue of NP) and hence is presumably not doable in polynomial time.

Temporal graphs are a natural generalisation of graphs that capture dynamic changes over time in the edge set. They have a fixed vertex set and a set of *time-edges* which have integer time labels indicating at which time(s) they are active. In recent years, the research field of studying algorithmic problems on temporal graphs has steadily grown [28, 29, 34, 35]. In particular, an additional layer of complexity is added to connectivity related problems in the temporal setting. Paths in temporal graphs have to respect time, that is, a *temporal path* has to traverse time-edges with non-decreasing time labels [32][1]. This implies that temporal connectivity is generally not symmetric and not transitive, a major difference from the non-temporal case. Furthermore, there are several natural optimality concepts for temporal paths, the most important being *shortest*, *foremost*, and *fastest* temporal paths [10]. Intuitively speaking, shortest temporal paths use a minimum number of time-edges, foremost temporal paths arrive as early as possible, and fastest temporal paths have a minimum difference between start and arrival times. We remark that an optimal path with respect to any of these three criteria can be found in polynomial time [10, 47]. The existence of multiple natural optimality concepts for temporal paths implies several natural definitions of temporal betweenness, one for each path optimality concept [40, 12, 34].

Similar to the non-temporal case, the ability to *count* optimal temporal paths is a key ingredient for the corresponding temporal betweenness computation. However, the picture is more complex in the temporal setting. Shortest temporal paths can be counted in polynomial time and the corresponding temporal betweenness can be computed in polynomial time [12, 33, 27, 40]. In contrast, counting foremost or fastest temporal paths is #P-hard [39, 12, 36], which implies that computing the corresponding temporal betweenness is #P-hard as well [12]. Indeed, Buß et al. [12] show that there is a polynomial time reduction from the problem of counting foremost or fastest temporal paths to the problem of the corresponding temporal betweenness computation. Note that a reduction in the other direction is straightforward.

In this work, we study the (parameterised) computational complexity of (approximately) counting foremost or fastest temporal paths. In fact, we study the simpler and arguably more natural problem of counting all temporal paths from a start vertex s to a destination vertex z in a temporal graph.

Let $\mathcal{G} = (V, \mathcal{E}, T)$ denote a temporal graph with vertex set V, time-edge set $\mathcal{E}$, and maximum time label (or lifetime) T (formal definitions are given in Section 2). We are then concerned with the following computational problem:

#Temporal Path	
Input:	A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ and two vertices $s, z \in V$.
Task:	Count the temporal (s, z)-paths in $\mathcal{G}$.

[1] Temporal paths that traverse time-edge with non-decreasing time labels are often referred to as "non-strict", in contrast to *strict* temporal paths, which traverse time-edges with increasing time labels. In this work, we focus on non-strict temporal paths.

It is easy to see that #Temporal Path generalises the problem of counting paths in a non-temporal graph (all time-edges have the same time label), hence we deduce that #Temporal Path is #P-hard. Furthermore, observe that using an algorithm for #Temporal Path, it is possible to count foremost or fastest temporal paths with only polynomial overhead in the running time; we discuss this reduction in more detail in Section 2.3. Hence, all exact algorithms we develop for #Temporal Path can be used to compute the temporal betweenness based on foremost or fastest temporal paths with polynomial overhead in the running time. To the best of our knowledge, this is the first attempt to systematically study the parameterised complexity and approximability of #Temporal Path.

1.1 Related Work

As discussed above, the temporal setting adds a new dimension to connectivity-related problems. The problems of computing shortest, foremost, and fastest temporal paths have been studied thoroughly [10, 47, 5]. The temporal setting also offers room for new natural temporal path variants that do not have an analogue in the non-temporal setting. Casteigts et al. [13] study the problem of finding *restless temporal paths* that dwell an upper-bounded number of time steps in each vertex, while Füchsle et al. [24] study the problem of finding *delay-robust routes* in a temporal graph (intuitively, temporal paths that are robust with respect to edge delays); both problems turn out to be NP-hard.

The problem of *counting* (optimal) temporal paths has mostly been studied indirectly in the context of temporal betweenness computation. However, the computation of temporal betweenness has received much attention [12, 39, 40, 33, 44, 27, 43, 2, 45, 38, 41]. Most of the mentioned work considers temporal betweenness variants that are polynomial-time computable. The corresponding optimal temporal paths are mostly shortest temporal paths or variations thereof. There are at least three notable exceptions: Buß et al. [12] also consider *prefix-foremost* temporal paths and the corresponding temporal betweenness and show that the latter is computable in polynomial time. Furthermore they show #P-hardness for several temporal betweenness variants based on *strict* optimal temporal paths. Rad et al. [39] consider temporal betweenness based on foremost temporal paths and show that its computation is #P-hard. They further give an FPT-algorithm to compute temporal betweenness based on foremost temporal paths for the number of vertices as a parameter (note that the size of a temporal graph generally cannot be bounded by a function of the number of its vertices). Rymar et al. [40] give a quite general sufficient condition called *prefix-compatibility* for optimality concepts for temporal paths that makes it possible to compute the corresponding temporal betweenness in polynomial time.

In the static setting the general problem of counting (s, z)-paths in static graphs is known to be #P-complete [46]. In the parameterised setting, the problem of counting length-k paths (with parameter k) was one of the first problems shown to be #W[1]-complete [21], but the problem does admit an efficient parameterised approximation algorithm [3]. It is also generally considered folklore that the problem of counting paths (of any length) admits an FPT-algorithm parameterised by the treewidth of the input graph.

1.2 Our Contribution

Our goal is to initiate the systematic study of the parameterised and approximation complexity of #Temporal Path. We provide an argument that #Temporal Path is essentially equivalent to counting foremost or fastest paths or computing the respective temporal betweenness centrality in Section 2.3. Due to space constraints, proofs of results marked with ($\star$) are (partially) deferred to the full version of this work [19].

Hardness results (Section 3). The main technical contribution of this paper is a reduction showing that #Temporal Path is intractable even when very strong restrictions are placed on the underlying graph; specifically the problem is hard for $\oplus$W[1] when parameterised by the feedback vertex number of the underlying graph, which rules out the existence of FPT algorithms with respect to several common parameters. We also show that it is NP-hard even to approximate the number of temporal (s, z)-paths in general, motivating the study of approximate counting in more restricted settings.

Exact algorithms for special cases (Section 4). We show that the problem is polynomial-time solvable if the underlying graph is a forest, and then use a wide range of algorithmic techniques to generalise this result in different ways. We show that the problem is fixed-parameter tractable with respect to two "distance to forest" parameterisations that are larger than the feedback vertex number of the underlying graph (timed feedback vertex number and underlying feedback edge number). We further show that #Temporal Path is in FPT parameterised by the treewidth of the underlying graph and the lifetime combined, or parameterised by the recently introduced parameter "vertex-interval-membership-width".

Approximation algorithms (Section 5). We show that there is an FPTRAS for #Temporal Path parameterised by the maximum permitted length of a temporal (s, z)-path. We then turn our attention to the problem of approximating betweenness centrality, as the relationship between path counting and computing betweenness is not so straightforward in the approximate setting: we demonstrate that, whenever there exists an FPRAS (respectively FPTRAS) for #Temporal Path, we can efficiently approximate the maximum betweenness centrality of any vertex in the temporal graph. These two results together give an FPTRAS to estimate the maximum betweenness centrality of any vertex in a temporal graph (with respect to either foremost or fastest temporal paths) parameterised by the vertex cover number or treedepth of the underlying input graph.

2 Preliminaries and Basic Observations

In this section we provide all basic notations, definitions, and terminology used in this work. We discuss the relation between temporal path counting and temporal betweenness computation in more detail in Section 2.3. We use standard definitions and terminology from parameterised complexity theory [18, 22, 16, 3, 21]. Additional background on parameterised and approximate counting complexity are given in the full version of this work [19].

Given a static graph $G = (V, E)$, we say that a sequence $P = (\{v_{i-1}, v_i\})_{i=1}^{k}$ of edges in E forms a *path* in G if $v_i \neq v_j$ for all $0 \leq i < j \leq k$.

2.1 Temporal Graphs and Paths

There are several different definitions and notations used in the context of temporal graphs [28, 29, 34, 35] which are mostly equivalent. Here, we use the following definitions and notations:

An (undirected, simple) *temporal graph* with lifetime $T \in \mathbb{N}$ is a tuple $\mathcal{G} = (V, \mathcal{E}, T)$, with time-edge set $\mathcal{E} \subseteq \binom{V}{2} \times [T]$. We assume all temporal graphs in this paper to be undirected and simple. The *underlying graph* of $\mathcal{G}$ is defined as the static graph $G = (V, \{\{u, v\} \mid \exists t \in [T] \text{ s.t. } (\{u, v\}, t) \in \mathcal{E}\})$. We denote by E_t the set of edges of G that are active at time t, that is, $E_t = \{\{u, v\} \mid (\{u, v\}, t) \in \mathcal{E}\}$.

For every $v \in V$ and every time step $t \in [T]$, we denote the *appearance of vertex v at time t* by the pair (v,t). For a time-edge $(\{v,w\},t)$ we call the vertex appearances (v,t) and (w,t) its *endpoints* and we call $\{v,w\}$ its *underlying edge*.

We assume that every number in $[T]$ appears at least once as a label for an edge in $\mathcal{E}$. In other words, we ignore labels that are not used for any edges since they are irrelevant for the problems we consider in this work. It follows that we assume $T \leq |\mathcal{E}|$ and hence $T \in \mathcal{O}(|\mathcal{G}|) = \mathcal{O}(|V| + |\mathcal{E}|)$.

A *temporal (s,z)-path* (or *temporal path*) of length k from vertex $s = v_0$ to vertex $z = v_k$ in a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$ is a sequence $P = ((\{v_{i-1}, v_i\}, t_i))_{i=1}^{k}$ of time-edges in $\mathcal{E}$ such that the corresponding sequence of underlying edges forms a path in the underlying graph of $\mathcal{G}$ and, for all $i \in [k-1]$, we have that $t_i \leq t_{i+1}$. Given a temporal path $P = ((\{v_{i-1}, v_i\}, t_i))_{i=1}^{k}$, we denote the set of vertices of P by $V(P) = \{v_0, v_1, \ldots, v_k\}$ and we say that P *visits* the vertex v_i if $v_i \in V(P)$. Moreover, we call vertex appearances (v_{i-1}, t_i) *outgoing* for P and we call the vertex appearances (v_i, t_i) *incoming* for P. Note that, if $t_i = t_{i+1}$, then (v_i, t_i) is both incoming and outgoing for P. We define $(v_0, 1)$ to be incoming for P and (v_k, T) to be outgoing for P. We say that a vertex appearance is *visited* by P if it is outgoing or incoming for P (so a vertex is visited by P if and only if at least one of its appearances is visited by P). We say that P *starts* at v_0 at time t_1 and *arrives* at v_k at time t_k. We say that P' is a *temporal subpath* of P if P' is a subsequence of P. Furthermore, we define the following optimality concepts for temporal (s,z)-paths P.

- P is a *shortest* temporal (s,z)-path if there is no temporal path P' from s to z such that the length of P' is strictly less than the length of P.
- P is a *foremost* temporal (s,z)-path if there is no temporal path P' from s to z such that P' arrives at z at a strictly smaller time than P.
- P is a *fastest* temporal (s,z)-path if there is no temporal path P' from s to z such that the difference between the time at which P' starts at s and the time at which P' arrives at z is strictly smaller than the analogous difference of times for P.

2.2 Temporal Betweenness Centrality

We follow the notation and definition for temporal betweenness given by Buß et al. [12]. Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. For any $s, z \in V$, $\sigma_{sz}^{(\star)}$ is the number of $\star$-optimal temporal paths from s to z. We define $\sigma_{vv}^{(\star)} := 1$. For any vertex $v \in V$, we write $\sigma_{sz}^{(\star)}(v)$ for the number of $\star$-optimal paths that pass through v. We set $\sigma_{sz}^{(\star)}(s) := \sigma_{sz}^{(\star)}$ and $\sigma_{sz}^{(\star)}(z) := \sigma_{sz}^{(\star)}$.

We do not assume that there is a temporal path from any vertex to any other vertex in the graph. To determine between which (ordered) pairs of vertices a temporal path exists, we use a *connectivity matrix* A of the temporal graph: let A be a $|V| \times |V|$ matrix, where for every $v, w \in V$ we have that $A_{v,w} = 1$ if there is a temporal path from v to w, and $A_{v,w} = 0$ otherwise. Note that $A_{s,z} = 1$ if and only if $\sigma_{sz}^{(\star)} \neq 0$. Formally, temporal betweenness based on $\star$-optimal temporal paths is defined as follows.

▶ **Definition 1** (Temporal Betweenness). *The* temporal betweenness *of any vertex $v \in V$ is given by:*

$$C_B^{(\star)}(v) := \sum_{s \neq v \neq z \text{ and } A_{s,z}=1} \frac{\sigma_{sz}^{(\star)}(v)}{\sigma_{sz}^{(\star)}}.$$

2.3 Temporal Betweenness vs. Temporal Path Counting

In this subsection we discuss the relationship between the problems of computing temporal betweenness and counting temporal paths. We show that we can compute temporal betweenness based on foremost and fastest temporal paths using an algorithm for #Temporal Path with only polynomial overhead in the running time. Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. We start with the following easy observation.

▶ **Observation 2.** *Given an algorithm to count all $\star$-optimal temporal (s, z)-paths in $\mathcal{G}$ in time $t(\mathcal{G})$, we can compute the temporal betweenness based on $\star$-optimal temporal paths of any vertex of $\mathcal{G}$ in $t(\mathcal{G}) \cdot |\mathcal{G}|^{\mathcal{O}(1)}$ time.*

This follows by observing that we can count the number of temporal (s, z)-paths in $\mathcal{G}$ that visit a vertex v by first counting all temporal (s, z)-paths in $\mathcal{G}$ and then subtracting the number of temporal (s, z)-paths in $\mathcal{G} - \{v\}$.

Next we observe that we can count foremost and fastest temporal (s, z)-paths using an algorithm for #Temporal Path, with only polynomial overhead.

▶ **Observation 3.** *Given an algorithm for #Temporal Path that runs in time $t(\mathcal{G})$, we can compute all foremost temporal (s, z)-paths and all fastest temporal (s, z)-paths in $t(\mathcal{G}) \cdot |\mathcal{G}|^{\mathcal{O}(1)}$ time.*

First, note that we can compute a foremost temporal (s, z)-path and a fastest temporal (s, z)-path in polynomial time [10, 47]. In the case of foremost temporal (s, z)-paths, we can in this way obtain the time at which a foremost temporal (s, z)-path arrives at z and remove all time-edges with later time labels from $\mathcal{G}$. After this modification, every temporal (s, z)-path is foremost hence we can count them using an algorithm for #Temporal Path.

In the case of fastest temporal (s, z)-paths, we can in the same way obtain the time difference t_f between starting at s and arriving at z for any fastest temporal (s, z)-path. We can now iterate over all intervals $[t_0, t_0 + t_f]$ with $1 \leq t_0 \leq T - t_f$ and, for each one, create an instance of #Temporal Path by removing all time-edges from $\mathcal{G}$ that are either earlier than t_0 or later than $t_0 + t_f$. After this modification, every temporal (s, z)-path in the instance corresponding to any interval is fastest, and every fastest temporal path survives in exactly one instance; hence we can count fastest temporal paths by calling an algorithm for #Temporal Path on each instance and summing the results.

Using Observations 2 and 3 we obtain the following lemma, which implies that our polynomial-time and FPT-algorithms for special cases of #Temporal Path yield polynomial-time solvability and fixed-parameter tractability results respectively for temporal betweenness based on foremost temporal paths or fastest temporal paths, under the same restrictions.

▶ **Lemma 4.** *Given an algorithm for #Temporal Path that runs in time $t(\mathcal{G})$, we can can compute the temporal betweenness based on foremost temporal paths or fastest temporal paths of any vertex of $\mathcal{G}$ in $t(\mathcal{G}) \cdot |\mathcal{G}|^{\mathcal{O}(1)}$ time.*

If we can only count temporal paths *approximately*, however, the relationship between temporal path counting and temporal betweenness computation is not so straightforward. In the exact setting, we were able to determine the number of temporal (s, z)-paths visiting v by calculating the difference between the number of temporal (s, z)-paths in $\mathcal{G}$ and $\mathcal{G} - \{v\}$ respectively. However, in the approximate setting, we cannot use the same strategy: if there are N temporal paths in total and N_{-v} is an ε-approximation to the number of temporal paths that do not contain v, it does not follow that $N - N_{-v}$ is an ε-approximation to the number of temporal paths that do contain v, as the relative error will potentially be much

higher if the proportion of temporal paths containing v is very small. A similar issue arises if we aim to estimate the number of temporal paths through v by sampling a collection of temporal paths (from an approximately uniform distribution) and using the proportion that contain v as an estimate for the total proportion of temporal paths containing v: if the proportion that contain v is exponentially small, we would need exponentially many samples to have a non-trivial probability of finding at least one temporal path which does contain v; otherwise we deduce incorrectly that there are no temporal paths through v and output 0, which cannot be an ε-approximation of a non-zero number of temporal paths for any $\varepsilon < 1$.

Lastly, we briefly shift our attention to computational hardness. Buß et al. [12] provide a reduction from #Temporal Path to the computation of temporal betweenness based on foremost temporal paths and to the computation of temporal betweenness based on fastest temporal paths. In both cases, three new vertices are added to the temporal graph and all newly added time-edges are incident with at least one of the newly added vertices. This implies that our parameterised hardness result in the next section (Theorem 5) also holds for temporal betweenness computation based on foremost temporal paths or fastest temporal paths, since the reductions by Buß et al. [12] increase the feedback vertex number of the underlying graph by at most three.

3 Intractability Results for Temporal Path Counting

In this section we prove two hardness results for #Temporal Path. In Section 3.1 we demonstrate parameterised intractability with respect to the feedback vertex number of the underlying graph. We follow this in Section 3.2 with an easy reduction demonstrating that the classical #P-complete #Path problem [46] (definition as below) is unlikely to admit an FPRAS in general, which straightforwardly implies the same result for #Temporal Path.

#Path	
Input:	A graph $G = (V, E)$ and two vertices $s, z \in V$.
Task:	Compute the number of paths from s to z in G.

3.1 Parameterised Hardness

In this section we present our main parameterised hardness result, which provides strong evidence that #Temporal Path does not admit an FPT algorithm when parameterised by the feedback vertex number of the underlying graph. Note that this also rules out FPT algorithms for many other parameterizations, including the treewidth of the underlying graph. However, it is folklore that #Path admits an FPT algorithm parameterised by treewidth as a parameter (this is also implied by our result Theorem 14). The result here, therefore, means that #Temporal Path is strictly harder than #Path in terms of parameterised complexity for the parameterisations that are at most the feedback vertex number of the underlying graph and at least the treewidth of the underlying graph.

▶ **Theorem 5** (⋆). *#Temporal Path is $\oplus$W[1]-hard when parameterised by the feedback vertex number of the underlying graph.*

Proof. We present a parameterised counting Turing reduction from $\oplus$Multicoloured Independent Set on 2-Track Interval Graphs parameterised by the number of colours k. In $\oplus$Multicoloured Independent Set on 2-Track Interval Graphs we are given a set I of interval pairs and a colouring function $c : I \to [k]$ and asked whether there is an odd number of k-sized sets of interval pairs in I such that in each set,

every two interval pairs have different colours and are non-intersecting. Two interval pairs $([x_a, x_b], [x_{a'}, x_{b'}]), ([y_a, y_b], [y_{a'}, y_{b'}])$ are considered non-intersecting if $[x_a, x_b] \cap [y_a, y_b] = \emptyset$ and $[x_{a'}, x_{b'}] \cap [y_{a'}, y_{b'}] = \emptyset$.

Inspecting the W[1]-hardness proof by Jiang [31] for Independent Set on 2-Track Interval Graphs shows that the reduction used from Multicoloured Clique parameterised by the number of colours k is parsimonious[2] and the reduction also shows W[1]-hardness for the multicoloured version of the problem. Since $\oplus$Multicoloured Clique is $\oplus$W[1]-hard when parameterised by the number of colours k [8], we can conclude that $\oplus$Multicoloured Independent Set on 2-Track Interval Graphs is $\oplus$W[1]-hard when parameterised by the number of colours k.

Given an instance (I, c) of $\oplus$Multicoloured Independent Set on 2-Track Interval Graphs, where I is a set of interval pairs and $c : I \to [k]$ is a colouring function, we create $\mathcal{O}(2^k)$ temporal graphs. We assume w.l.o.g. that for all $([x_a, x_b], [x_{a'}, x_{b'}]), ([y_a, y_b], [y_{a'}, y_{b'}]) \in I$ that $|\{x_a, x_b, y_a, y_b\}| = 4$ and $|\{x_{a'}, x_{b'}, y_{a'}, y_{b'}\}| = 4$ or $([x_a, x_b], [x_{a'}, x_{b'}]) = ([y_a, y_b], [y_{a'}, y_{b'}])$, that is, if two interval pairs are different, we assume that all endpoints on each track are pairwise different. Furthermore, we assume w.l.o.g. that all intervals contained in pairs in I are integer subsets of $[2|I|]$. The main intuition of our construction follows:

- We model track one with a path in the underlying graph and track two with time.
- Through the feedback vertices of the underlying graph, a temporal path can "enter" and "leave" the path that models track one.
- The number of feedback vertices corresponds to the number of colours.
- We have to make sure that we can determine the parity of the number of temporal paths visiting all feedback vertices.
- The number of temporal paths that do not correspond to independent sets should not be considered. It seems difficult to get an exact handle on the number of such paths, however we will show that this number is even. Note that, intuitively, this is the main reason we show hardness for $\oplus$W[1] and not #W[1].

We construct a family of *directed* temporal graphs $(\mathcal{G}^{\mathcal{C}} = (V, \mathcal{A}^{\mathcal{C}}, 2|I| + 1))_{\mathcal{C} \subseteq [k]}$ with rational time labels (such that the maximum time label is at most $2|I| + 1$), where $\mathcal{A}^{\mathcal{C}} \subseteq V \times V \times \mathbb{Q}$ for all $\mathcal{C} \subseteq [k]$. Towards the end of the proof we explain how to remove the need for directed edges which will also have the consequence that the temporal graphs only contain strict temporal paths. Note that we can scale up the lifetime to remove the need for rational time labels, however using rational time labels will be convenient in the construction and the correctness proof.

- We set $V := V_I \cup \{s, z', z\} \cup \{w_1, \ldots, w_k\} \cup \{u_x \mid x \in I\}$, where $V_I := \{v_1, \ldots, v_{2|I|}\}$.
- We set $\mathcal{A}^{\mathcal{C}} := \bigcup_{x \in I \wedge c(x) \in \mathcal{C}} \mathcal{A}_x \cup \{(s, w_i, 1) \mid i \in [k]\} \cup \{(z', z, 2|I| + 1)\}$, where

$$\begin{aligned}\mathcal{A}_x :=& \{(w_{c(x)}, u_x, a'), (u_x, v_a, b'), (u_x, v_a, b' + 1 - a\varepsilon), (v_b, z', b')\} \\ &\cup \{(v_b, w_i, b') \mid i \in [k] \wedge i \neq c(x)\} \\ &\cup \{(v_j, v_{j+1}, b'), (v_j, v_{j+1}, b' + 1 - (j+1)\varepsilon) \mid j \in \{a, \ldots, b-1\}\}\end{aligned}$$

for $x = ([a, b], [a', b']) \in I$ and $\varepsilon = \frac{1}{2|I|}$.

[2] Informally speaking, parsimonious reductions do not change the number of solutions.

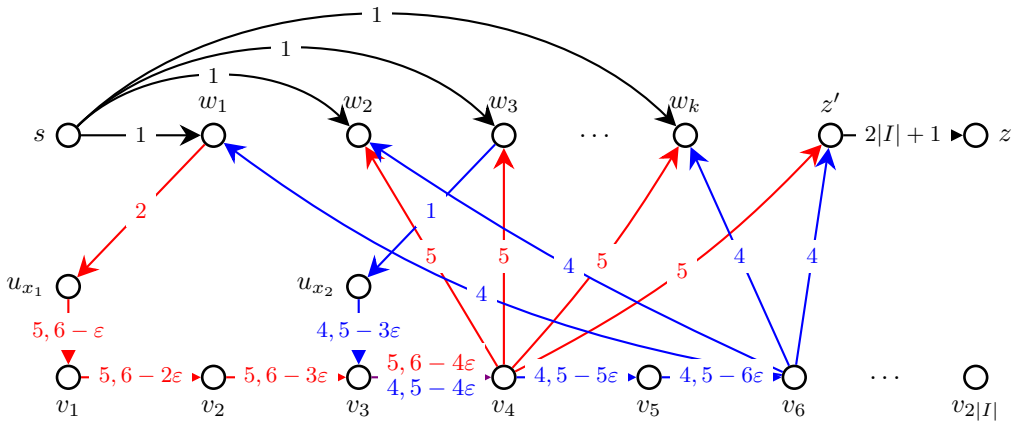


Figure 1 Illustration of $\mathcal{G}^{\mathcal{C}}$ with $\mathcal{C} = \{1, 3\}$ and two interval pairs $x_1, x_2 \in I$ where $x_1 = ([1, 4], [2, 5])$ and $x_2 = ([3, 6], [1, 4])$, and the corresponding colours are $c(x_1) = 1$ and $c(x_2) = 3$. The arcs added for x_1 are depicted in red and the arcs added for x_2 are depicted in blue.

For all $\mathcal{G}^{\mathcal{C}}$ we use s as the starting vertex and z as the end vertex of the temporal paths we want to count. The temporal graphs $\mathcal{G}^{\mathcal{C}}$ can each clearly be constructed in polynomial time and it is easy to see that the vertex set $\{s, z', z, w_1, \ldots, w_k\}$ constitutes a feedback vertex set of size $\mathcal{O}(k)$ for each of them (even if edge directions are removed). The construction is illustrated in Figure 1. The correctness proof of the reduction is deferred to the full version of this work [19]. ◀

3.2 Approximation Hardness

In this section, we prove the following result.

▶ **Theorem 6.** *There is no fully polynomial randomised approximation scheme (FPRAS) for #Temporal Path unless randomised polynomial time (RP) equals NP.*

It is straightforward to reduce from #Path, the problem of counting (s, z)-paths in a static graph, to #Temporal Path: we set every edge to be active at time one only. Hardness of #Path is proved easily by imitating the reduction used by Jerrum et al. [30] to demonstrate that there is no FPRAS to count directed cycles in a directed graph.[3] We note that the reduction also rules out the existence of any polynomial-time (randomised) approximation algorithm achieving any polynomial additive error.

▶ **Theorem 7** (⋆). *There is no FPRAS for #Path unless RP=NP.*

4 Exact Algorithms for Temporal Path Counting

In this section, we present several exact algorithms for #Temporal Path. We start in Section 4.1 with a polynomial-time algorithm for temporal graphs that have a forest as underlying graph. In Section 4.2 we show that our polynomial-time algorithm can be generalised in two ways, obtaining FPT-algorithms for the so-called timed feedback vertex

[3] Indeed, the fact that this technique can be adapted to demonstrate the hardness of approximately counting (s, z)-paths is noted without proof by Sinclair [42].

number and the feedback edge number of the underlying graph. In Section 4.3 we show that #Temporal Path is in FPT when parameterised by the treewidth of the underlying graph and the lifetime combined. Lastly, in Section 4.4, we give an FPT algorithm for #Temporal Path parameterised by the so-called vertex-interval-membership-width.

4.1 A Polynomial Time Algorithms for Forests

As a warm-up, we note that #Temporal Path can be solved in polynomial time with a simple dynamic program if the underlying graph is a forest. This is used as a subroutine for algorithms presented in Section 4.2.

▶ **Theorem 8.** *#Temporal Path is solvable in $\mathcal{O}(|V| \cdot T^2)$ time if the underlying graph of the input temporal graph is a forest.*

Proof. Let $\mathcal{G} = (V, \mathcal{E}, T)$ together with two verices $s, z \in V$ be an an instance of #Temporal Path. We argue that this instance can be solved in polynomial time if there is a unique path between s and z in the underlying graph of $\mathcal{G}$. Note that this is the case if the underlying graph of $\mathcal{G}$ is a forest.

First, observe that when counting (s, z)-paths starting at s and arriving at z, if there is a unique static path between s and z in the underlying graph then we need only consider time-edges between vertices of that unique static path in our temporal graph when counting, as our temporal path may not repeat vertices and so corresponds to a path in the underlying graph. Edges not lying on the unique static path between s and z can therefore be deleted without changing the result, so we may w.l.o.g. consider an instance in which the underlying graph consists only of a static path $P = (v_0, v_1, ..., v_{|P|})$ with $s = v_0$ and $z = v_{|P|}$ as the leaf vertices.

We will base our counting on a recording at each vertex v_i in P of how many temporal (s, v_i)-paths there are starting at s and arriving at v_i at time t or earlier. Note that there are $\mathcal{O}(|P| \cdot T) = \mathcal{O}(|V| \cdot T)$ such vertex-time pairs.

We argue by induction on i that we can correctly compute this number for every vertex-time pair by dynamic programming. As a base case, note that there is one path from s to s for any arrival time. Then we assume that we have these numbers computed correctly for some v_i with $i \geq 0$ and show how we compute them for v_{i+1}. Formally, our dynamic program is defined as follows.

$$F(v_0 = s, t) = 1$$
$$F(v_i, t) = \sum_{(\{v_{i-1}, v_i\}, t') \in \mathcal{E} \text{ with } t' \leq t} F(v_{i-1}, t') \text{ for } i \geq 1.$$

It is straightforward to check that $F(z, T)$ can be computed in the claimed running time. We now formally prove correctness by induction on i. That is, we prove that $F(v_i, t)$ equals the number of temporal (s, v_i)-paths that start at s and arrive at v_i at time t or earlier; it will follow immediately that $F(z, T)$ is the number of (s, z)-paths, so it suffices to compute $F(v_i, t)$ for all $0 \leq i \leq |P|$.

The base case $i = 0$ is trivial. Assume that $i > 0$. We sum over the last time-edge of the temporal (s, v_i)-paths starting at s and arriving at v_i at time t or earlier. Let $\mathcal{P}$ be the set of all temporal (s, v_i)-paths starting at s and arriving at v_i at time t or earlier that use $(\{v_{i-1}, v_i\}, t') \in \mathcal{E}$ as the last time-edge. All these temporal paths need to arrive at v_{i-1} at time t' or earlier, otherwise they cannot use time-edge $(\{v_{i-1}, v_i\}, t')$. Since all temporal paths in $\mathcal{P}$ do not differ in the last time-edge, the cardinality of $\mathcal{P}$ equals the number of

temporal (s, v_{i-1})-paths starting at s and arriving at v_{i-1} at time t' or earlier. By the induction hypothesis this number equals $F(v_{i-1}, t')$. Clearly, if the last time-edge of two temporal (s, v_i)-paths starting at s and arriving at v_i at time t or earlier is different, then the two temporal paths are different, so we do not double count. Hence, we have shown that $F(v_i, t)$ equals the number of temporal (s, v_i)-paths that start at s and arriving at v_i at time t or earlier. ◀

4.2 Generalisations of the Forest Algorithm

In this subsection, we present two generalisations of Theorem 8. The first one results in an FPT-algorithm for the timed-feedback vertex number as a parameter and the second one in an FPT-algorithm for the feedback edge number of the underlying graph as a parameter. We remark that both parameters are larger than the feedback vertex number of the underlying graph, for which Theorem 5 refutes the existence of FPT-algorithms. Both algorithms are inspired by algorithms presented by Casteigts et al. [13] for the so-called Restless Temporal Path problem.

The timed feedback vertex number was introduced by Casteigts et al. [13] and, intuitively, counts the minimum number of *vertex appearances* that need to be removed from a temporal graph to make its underlying graph cycle-free. Formally, it is defined as follows.

▶ **Definition 9** ([13]). *Let $\mathcal{G} = (V, \mathcal{E}, T)$ be a temporal graph. A* timed feedback vertex set *of $\mathcal{G}$ is a set $X \subseteq V \times [T]$ of vertex appearances such that the underlying graph of $\mathcal{G}' = (V, \mathcal{E}', T)$ is a forest, where $\mathcal{E}' := \mathcal{E} \setminus \{(\{v, w\}, t) \in \mathcal{E} \mid (v, t) \in X \vee (w, t) \in X\}$. The* timed feedback vertex number *of a temporal graph $\mathcal{G}$ is the minimum cardinality of a timed feedback vertex set of $\mathcal{G}$.*

Our FPT-algorithm for the timed feedback vertex number as a parameter follows similar ideas as the one by Casteigts et al. [13] for the Restless Temporal Path problem. Roughly speaking, our algorithm performs the following steps.

1. Compute a minimum cardinality timed feedback vertex set of the input temporal graph.
2. Iterate over all possibilities for how a temporal path can traverse the vertex appearances in the timed feedback vertex set.
3. For each possibility, create an instance of the so-called #Weighted Multicoloured Independent Set on Chordal Graphs problem to compute the number of possibilities for connecting the vertex appearances of the timed feedback vertex set that are supposed to be traversed.
4. Use this to compute the total number of temporal (s, z)-paths in the temporal input graph.

The intuition here is that the possibilities for connecting the vertex appearances of the timed feedback vertex set that are supposed to be traversed correspond to path segments in the underlying graph of the temporal graph without the timed feedback vertex set, which is a forest. It is well-known that chordal graphs are intersection graphs of subtrees in forest [25]. This means that an independent set in a chordal graph corresponds to a selection of non-intersecting subtrees (which here will all be paths). The colours can be used to make sure that, for each pair of vertex appearances of the timed feedback vertex set that are supposed to be traversed directly after one another, exactly one path segment connecting them can be in the independent set. The weights can be used to model how many temporal paths follow the corresponding path segment of the underlying graph.

As mentioned above, we have to solve #Weighted Multicoloured Independent Set on Chordal Graphs as a subroutine instead of the unweighted decision version of the problem. This is the main difference between our algorithm and the one by Casteigts et al. [13]. In the following we give a formal definition.

#Weighted Multicoloured Independent Set on Chordal Graphs	
Input:	A chordal graph $G = (V, E)$, a colouring function $c : V \to [k]$, and a weight function $w : V \to \mathbb{N}$.
Task:	Compute $\sum_{X \subseteq V \mid X \text{ is a multicoloured independent set in } G} \prod_{v \in X} w(v)$.

We can observe that #Weighted Multicoloured Independent Set on Chordal Graphs presumably cannot be solved in polynomial time. This follows directly from the NP-hardness of Multicoloured Independent Set on Chordal Graphs [7, Lemma 2]. Hence, we have the following.

▶ **Observation 10.** *#Weighted Multicoloured Independent Set on Chordal Graphs cannot be solved in polynomial time unless P=NP.*

However, we can obtain an FPT-algorithm for #Weighted Multicoloured Independent Set on Chordal Graphs parameterised by the number of colours. This will be sufficient for our purposes. To show this result, we adapt an algorithm by Bentert et al. [6, Proposition 5.6] to solve Maximum Weight Multicoloured Independent Set on Chordal Graphs, where given a chordal graph $G = (V, E)$, a colouring function $c : V \to [k]$, and a weight function $w : V \to \mathbb{N}$, one is asked to compute a multicoloured independent set of maximum weight in G. Here, the weight of an independent set is the *sum* of the weights of its vertices. Note that in #Weighted Multicoloured Independent Set on Chordal Graphs the weight of an independent set is the *product* of the weights of its vertices.

▶ **Proposition 11** (⋆). *#Weighted Multicoloured Independent Set on Chordal Graphs is fixed-parameter tractable when parameterised by the number k of colours.*

Using Proposition 11, we are ready to give our FPT-algorithm for #Temporal Path parameterised by the timed feedback vertex number.

▶ **Theorem 12** (⋆). *#Temporal Path is fixed-parameter tractable when parameterised by the timed feedback vertex number of the input temporal graph.*

Now we consider the feedback edge number of the input temporal graph as our parameter, and show the following fixed-parameter tractability result. It is very similar to an algorithm by Casteigts et al. [13] for the so-called Restless Temporal Path problem parameterised by the feedback edge number, we only sketch the proof.

▶ **Theorem 13.** *#Temporal Path is fixed-parameter tractable when parameterised by the feedback edge number of the underlying graph of the input temporal graph.*

Proof Sketch. Let $(\mathcal{G}, s, z)$ be an instance of #Temporal Path. We adapt an algorithm by Casteigts et al. [13, Theorem 7] for the so-called Restless Temporal Path problem. The algorithm consist of four steps (only the last step needs adaptation to our problem):

1. Exhaustively remove vertices with degree ≤ 1 from the underlying graph of $\mathcal{G}$ (except s and z). Let G' be the resulting (static) graph.
2. Compute a minimum feedback edge set F of G'. Let $f := |F|$.
3. Let $V^{\geq 3}$ denote all vertices of G' with degree at least three. Partition the forest $G' - F$ into a set of maximal paths $\mathcal{P}$ with endpoints in $\bigcup_{e \in F} e \cup V^{\geq 3} \cup \{s, z\}$, and intermediate vertices all of degree 2. It holds that $|\mathcal{P}| \in \mathcal{O}(f)$ [4, Lemma 2].
4. Any temporal (s, z)-path in $\mathcal{G}$ can be formed with time-edges whose underlying edges are feedback edges from F or form paths in $\mathcal{P}$. Enumerate all $2^{\mathcal{O}(f)}$ sequences of underlying edges that a temporal (s, z)-path in $\mathcal{G}$ can follow and for each one count the temporal (s, z)-paths following these underlying edges using Theorem 8. Add up all path counts.

The correctness follows from the correctness of [13, Theorem 7] and that due to the exhaustive search, all temporal (s, z)-paths in $\mathcal{G}$ are considered and correctly counted. ◀

4.3 Parameterisation by Treewidth and Lifetime

Our goal in this subsection is to demonstrate that #Temporal Path is in FPT when parameterised simultaneously by the treewidth of the underlying graph and the lifetime; to do this we give an MSO-encoding of the problem and make use of the counting version of Courcelle's theorem for model-checking on relational structures [15].

▶ **Theorem 14** (⋆). *#Temporal Path is in FPT when parameterised by the combination of the treewidth of the underlying graph and the lifetime.*

4.4 Parameterisation by Vertex-Interval-Membership-Width

In this subsection, we present an FPT algorithm for #Temporal Path parameterised by the so-called vertex-interval-membership-width of the input temporal graph. The vertex-interval-membership-width is a temporal graph parameter recently introduced by Bumpus and Meeks [11] which, like the timed feedback vertex number, depends not only on the structure of the underlying graph but also on the assignment of times to edges. Intuitively, the vertex-interval-membership-width counts the maximum number of vertices that are "relevant" at any timestep, where a vertex is considered relevant if it has an incident edge both (weakly) before and after the current timestep (so, for example, a vertex v is relevant only at times when a temporal path could have entered but not yet left v).

▶ **Definition 15** ([11]). *The* vertex interval membership sequence *of a temporal graph $(G, \mathcal{E}, T)$ is the sequence $(F_t)_{t\in[T]}$ of vertex-subsets of G where*

$$F_t := \{v \in V(G) \mid \exists i \leq t \leq j \text{ and } u, w \in V(G) \text{ such that } \{u, v\} \in E_i \text{ and } \{w, v\} \in E_j\}.$$

Note that we allow $u = w$. The vertex-interval-membership-width *of $(G, \mathcal{E}, T)$ is the integer* $\mathrm{vimw}(G, \mathcal{E}, T) := \max_{t\in[T]} |F_t|$.

Note that every vertex incident with an edge in E_i must belong to F_i, and so $|E_i| \leq \binom{|F_i|}{2} \leq |F_i|^2$. The vertex interval membership sequence gives us a structure we can use for dynamic programming, which we exploit to obtain the following result.

▶ **Theorem 16** (⋆). *#Temporal Path can be solved in time $\mathcal{O}(w^{2w^2+w} \cdot T)$ where T and w are the lifetime and vertex-interval-membership-width respectively of the input graph.*

In our dynamic programming algorithm, a state of the bag F_t is a pair (v, X), where $v \in F_t$ and $X \subseteq F_t \setminus \{v\}$. For any state (v, X) of F_t, we compute the number $P_t(v, X)$ of temporal paths Q from s to v, arriving by time t, such that $V(Q) \cap (F_t \setminus \{v\}) = X$. Computing all such values $P_t(v, X)$ is clearly sufficient, since the total number of temporal (s, z)-paths is $\sum_{Y\subseteq F_T\setminus\{z\}} P_T(z, Y)$. We compute the values for each bag F_t in turn, assuming for $t \geq 1$ that we have already computed all counts corresponding to F_{t-1}.

5 Approximation Algorithms for Temporal Path Counting

In this section we consider the problems of approximating #Temporal Path and approximating the temporal betweenness centrality. For #Temporal Path, recall from Section 3.2 that there is unlikely to be an FPRAS for #Temporal Path in general; in Section 5.1, we show that there is however an FPTRAS for #Temporal Path when the maximum permitted path length is taken as the parameter. This in turn implies the existence of an FPTRAS for #Temporal Path when restrictions are placed on the structure of the

underlying graph that limit the length of the longest path. We remark that Theorem 5 and Theorem 6 do not rule out exact FPT-algorithms for these parameterisations. We leave open whether stronger hardness results or exact algorithms for this case can be obtained.

In Section 5.2 we apply this approximation result to the problem of approximating temporal betweenness: we demonstrate that, whenever we can efficiently approximate #Temporal Path, we can efficiently estimate the maximum temporal betweenness centrality over all vertices of the input graph.

5.1 Approximately Counting Short Temporal Paths

In this subsection we consider the complexity of approximately counting (s, z)-paths parameterised by the length of the path.

#Short Temporal Path	
Input:	A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, two vertices $s, z \in V$, and an integer k.
Task:	Count the temporal (s, z)-paths in $\mathcal{G}$ that contain exactly k edges.

We prove the following result.

▶ **Theorem 17.** *There is a randomised algorithm which, given as input an instance $(\mathcal{G}, s, z)$ of #Short Temporal Path together with error parameters $\varepsilon > 0$ and $0 < \delta < 1$, outputs an estimate $\hat{N}$ of the number of temporal (s, z)-paths in $\mathcal{G}$ containing exactly k edges; with probability at least $1-\delta$, $\hat{N}$ is an ε-approximation to the number of (s, z)-paths in $\mathcal{G}$ containing exactly k edges. The running time of the algorithm is $\mathcal{O}(k!e^k \log(1/\delta)\varepsilon^{-2}n^2T^2)$.*

The key ingredient in the proof is an efficient algorithm for the *multicoloured* version of this problem, in which the input graph is equipped with a vertex-colouring (not necessarily proper) and we wish to count paths containing exactly one vertex of each colour. We note that #Short Temporal Path meets the conditions for a *uniform witness problem* given by Dell et al. [17], and therefore in order to demonstrate the existence of an FPTRAS it would suffice to demonstrate that the multicoloured version of the problem admits an FPT decision algorithm. However, in this case it is easy to show that in fact *exact* counting is tractable in the multicoloured setting, so we can infer the existence of an FPTRAS immediately by applying a standard colour-coding technique, without invoking the power of the metatheorem by Dell et al. [17].

#Multicoloured Temporal Path	
Input:	A temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, two vertices $s, z \in V$, and a partition of $V \setminus \{s, z\}$ into colour sets $V_1 \uplus \cdots \uplus V_\ell$.
Task:	Count the number of temporal (s, z)-paths that contain exactly one vertex from each colour-set $V_1, \ldots, V_\ell$.

▶ **Proposition 18** ($\star$). *#Multicoloured Temporal Path is solvable in $\mathcal{O}((\ell+1)!n^2T^2)$ time.*

Equipped with this algorithm for #Multicoloured Temporal Path, we use a standard colour-coding technique to obtain an FPTRAS for #Short Temporal Path. This involves repeatedly generating random colourings (not necessarily proper) of the vertices of $V \setminus \{s, z\}$ using $k-1$ colours; note that a single colouring can clearly be generated in time $\mathcal{O}(nk)$. For each colouring, we solve the corresponding instance of #Multicoloured Temporal Path using the algorithm of Proposition 18. Setting N to be the sum of counts over all

colourings, we return $Nk^k/k!$. Following the argument by Alon et al. [1, Section 2.1], we see that the number of colourings we must generate to obtain an ε-approximation to #Short Temporal Path with probability at least $1-\delta$ is $\mathcal{O}(e^k \log(1/\delta)\varepsilon^{-2})$, giving the result.

Since the maximum possible path length is bounded by a function of either the vertex cover number or the treedepth[4] of the underlying input graph, we immediately obtain the following corollary to Theorem 17.

▶ **Corollary 19.** *#Temporal Path admits an FPTRAS parameterised by either vertex cover number or treedepth of the underlying input graph.*

5.2 Approximating Temporal Betweenness

Observe that it is not clear how to use an approximation algorithm for #Temporal Path to approximate the temporal betweenness centrality for every vertex in the input graph (we give a detailed discussion in Section 2.3). In this section, we address the simpler problem of determining the maximal temporal betweenness centrality of any vertex in the graph: we show that we can efficiently approximate this quantity whenever there is an FPRAS (or FPTRAS) for #Temporal Path.

▶ **Theorem 20** ($\star$)**.** *Let $\mathcal{C}$ be a class of temporal graphs on which #Temporal Path admits an FPRAS. Then $\mathcal{C}$ admits an FPRAS to estimate, given an input temporal graph $\mathcal{G} = (V, \mathcal{E}, T) \in \mathcal{C}$, $\max_{v \in V} C_B^{(\star)}(v)$, for $\star \in \{$fastest, foremost$\}$. Similarly, if $\mathcal{C}$ is a class of graphs on which there exists an FPTRAS for #Temporal Path with respect to some parameterisation κ then, with respect to the same parameterisation, $\mathcal{C}$ admits an FPTRAS to estimate, given an input temporal graph $\mathcal{G} = (V, \mathcal{E}, T) \in \mathcal{C}$, $\max_{v \in V} C_B^{(\star)}(v)$, for $\star \in \{$fastest, foremost$\}$.*

The proof relies on the fact that we may assume that at least one vertex has temporal betweenness centrality at least $\frac{1}{n(T+1)}$, where n is the number of vertices; we begin by arguing that we can efficiently identify the inputs for which this lower bound does not hold, and that in these cases the correct answer is in fact 0. Using this assumption, we show that the following procedure is likely to produce a good approximation to $\max_{v \in V} C_B^{(\star)}(v)$: for each vertex pair (s, z), sample a large (polynomial) number of $\star$-optimal temporal (s, z)-paths, and record the number that contain each vertex v as an internal vertex; after considering all pairs (s, z), we assume that the vertex $v_{\max}$ we have seen most frequently has the maximum betweenness centrality, and return as our estimate the proportion of sampled paths that contain $v_{\max}$. We note that, applying a general result of Jerrum et al. [30], we can assume the existence of an efficient algorithm to sample $\star$-optimal temporal (s, z)-paths almost uniformly whenever there is an FPRAS (or FPTRAS).

Combining Theorem 20 with Corollary 19 gives the following immediate corollary.

▶ **Corollary 21.** *There is an FPTRAS which, given as input a temporal graph $\mathcal{G} = (V, \mathcal{E}, T)$, computes an approximation to $\max_{v \in V} C_B^{(\star)}(v)$ (for $\star \in \{$fastest, foremost$\}$), parameterised by either the vertex cover number or treedepth of the underlying input graph.*

[4] We refer to the book of Nesetril and de Mendez [37] for the definition of treedepth, and a proof that the maximum length of a path in a graph is bounded by a function of its treedepth.

6 Conclusion

In this work, we initiate the systematic study of the parameterised and approximation complexity of #Temporal Path. We present parameterised and approximation hardness results and complement them with several parameterised exact and approximation algorithms.

In terms of improving our results, we conjecture that it is possible to prove #W[1]-hardness instead of $\oplus$W[1]-hardness for #Temporal Path parameterised by the feedback vertex number of the underlying graph. Furthermore, we leave open whether our parameterised approximation results for vertex cover number or treedepth of the underlying graph can be improved from a classification standpoint by obtaining exact algorithms, or whether we can also show parameterised hardness for those cases.

We leave open to what extent our results transfer to the problem of counting *strict* temporal (s, z)-paths, where the labels on the time-edges have to be strictly increasing. We conjecture that most of our results hold for the strict case. In fact, we believe that the MSO formulation used to obtain fixed-parameter tractability for the treewidth of the underlying graph combined with the lifetime can be simplified: in the strict case, a first-order formula should suffice, which would lead to fixed-parameter tractability in terms of the lifetime on any class of nowhere-dense graphs [26], or for the combined parameter of cliquewidth and lifetime [14].

We conjecture that our polynomial-time algorithm for the case where the underlying graph is a forest (Section 4.1) can be extended to the case where the underlying graph is series-parallel [20]. Recall that a forest can be seen as a series-parallel graph where only series compositions are used. The idea would be to extend the dynamic program to parallel compositions, exploiting the observation that any temporal path can visit the terminal vertices of a series-parallel graph at most once.

Finally, we believe that our FPT-algorithms presented in Section 4.2 for the timed feedback vertex number and feedback edge number of the underlying graph, respectively, can be adapted to count other types of temporal (s, z)-paths for which counting is in general #P-hard. A key ingredient for both algorithms is a polynomial-time algorithm for instances that have a forest as underlying graph. This leads us to believe that the algorithms can be modified to count restless temporal (s, z)-paths [13] and possibly also to count delay-robust (s, z)-routes [24], since both of these path types can be found in polynomial time when the underlying graph of the input temporal graph is a forest.

References

1 Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and Süleyman Cenk Sahinalp. Biomolecular network motif counting and discovery by color coding. In *Proceedings 16th International Conference on Intelligent Systems for Molecular Biology (ISMB '08)*, pages 241–249, 2008.

2 Ahmad Alsayed and Desmond J Higham. Betweenness in time dependent networks. *Chaos, Solitons & Fractals*, 72:35–48, 2015.

3 Vikraman Arvind and Venkatesh Raman. Approximation algorithms for some parameterized counting problems. In Prosenjit Bose and Pat Morin, editors, *Proceedings of the 13th International Symposium on Algorithms and Computation (ISAAC '02)*, volume 2518 of *Lecture Notes in Computer Science*, pages 453–464. Springer, 2002.

4 Matthias Bentert, Alexander Dittmann, Leon Kellerhals, André Nichterlein, and Rolf Niedermeier. An adaptive version of Brandes' algorithm for betweenness centrality. *Journal of Graph Algorithms and Applications*, 24(3):483–522, 2020.

5 Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(1):73, 2020.

6 Matthias Bentert, René van Bevern, and Rolf Niedermeier. Inductive k-independent graphs and c-colorable subgraphs in scheduling: a review. *Journal of Scheduling*, 22(1):3–20, 2019.

7 René Bevernvan Bevern, Matthias Mnich, Rolf Niedermeier, and Mathias Weller. Interval scheduling and colorful independent sets. *Journal of Scheduling*, 18(5):449–469, 2015.

8 Andreas Björklund, Holger Dell, and Thore Husfeldt. The parity of set systems under random restrictions with applications to exponential time problems. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP '15)*, pages 231–242. Springer, 2015.

9 Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.

10 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(02):267–285, 2003.

11 Benjamin Merlin Bumpus and Kitty Meeks. Edge exploration of temporal graphs. In *Proceedings of the 32nd International Workshop on Combinatorial Algorithms (IWOCA '21)*, volume 12757 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2021.

12 S. Buß, H. Molter, R. Niedermeier, and M. Rymar. Algorithmic aspects of temporal betweenness. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, pages 2084–2092, 2020. `arXiv:2006.08668`.

13 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021.

14 B. Courcelle, J.A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1):23–52, 2001.

15 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2012.

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

17 Holger Dell, John Lapinskas, and Kitty Meeks. Approximately counting and sampling small witnesses using a colourful decision oracle. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '20)*, pages 2201–2180. Society for Industrial and Applied Mathematics, 2020.

18 Rodney G Downey and Michael R Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

19 Jessica A. Enright, Kitty Meeks, and Hendrik Molter. Counting temporal paths. *CoRR*, abs/2202.12055, 2022. `arXiv:2202.12055`.

20 David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992.

21 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

22 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer, 2006.

23 Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

24 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-robust routes in temporal graphs. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS '22)*, volume 219 of *LIPIcs*, pages 30:1–30:15, 2022.

25 Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974.

26 Martin Grohe and Nicole Schweikardt. First-order query evaluation with cardinality conditions. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (SIGMOD/PODS '18)*, pages 253–266. ACM, 2018.

27 Habiba, Chayant Tantipathananandh, and Tanya Y Berger-Wolf. Betweenness centrality measure in dynamic networks. Technical Report 19, Department of Computer Science, University of Illinois at Chicago, Chicago, 2007. DIMACS Technical Report.

28 Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):234:1–234:30, 2015.

29 Petter Holme and Jari Saramäki. *Temporal Network Theory*. Springer, 2019.

30 Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

31 Minghui Jiang. On the parameterized complexity of some optimization problems related to multiple-interval graphs. *Theoretical Computer Science*, 411(49):4253–4262, 2010.

32 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002.

33 Hyoungshick Kim and Ross Anderson. Temporal node centrality in complex networks. *Physical Review E*, 85(2):026107, 2012.

34 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(1):61:1–61:29, 2018.

35 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Mathematics*, 12(4):239–280, 2016.

36 Petra Mutzel and Lutz Oettershagen. On the enumeration of bicriteria temporal paths. In *Proceedings of the 15th Annual Conference on Theory and Applications of Models of Computation (TAMC '19)*, volume 11436 of *Lecture Notes in Computer Science*, pages 518–535. Springer, 2019.

37 Jaroslav Nesetril and Patrice Ossona de Mendez. *Sparsity - Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and Combinatorics*. Springer, 2012.

38 Vincenzo Nicosia, John Tang, Cecilia Mascolo, Mirco Musolesi, Giovanni Russo, and Vito Latora. Graph metrics for temporal networks. In *Temporal Networks*, pages 15–40. Springer, 2013.

39 Amir Afrasiabi Rad, Paola Flocchini, and Joanne Gaudet. Computation and analysis of temporal betweenness in a knowledge mobilization network. *Computational Social Networks*, 4(1):5, 2017.

40 Maciej Rymar, Hendrik Molter, André Nichterlein, and Rolf Niedermeier. Towards classifying the polynomial-time solvability of temporal betweenness centrality. In *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '21)*, volume 12911 of *Lecture Notes in Computer Science*, pages 219–231. Springer, 2021.

41 Frédéric Simard, Clémence Magnien, and Matthieu Latapy. Computing betweenness centrality in link streams. *CoRR*, abs/2102.06543, 2021. `arXiv:2102.06543`.

42 Alistair Sinclair. *Randomised algorithms for counting and generating combinatorial structures*. PhD thesis, University of Edinburgh, 1988.

43 John Tang, Ilias Leontiadis, Salvatore Scellato, Vincenzo Nicosia, Cecilia Mascolo, Mirco Musolesi, and Vito Latora. Applications of temporal graph metrics to real-world networks. In *Temporal Networks*, pages 135–159. Springer, 2013.

44 John Tang, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Vincenzo Nicosia. Analysing information flows and key mediators through temporal centrality metrics. In *Proceedings of the 3rd ACM Workshop on Social Network Systems*, pages 3:1–3:6. ACM, 2010.

45 Ioanna Tsalouchidou, Ricardo Baeza-Yates, Francesco Bonchi, Kewen Liao, and Timos Sellis. Temporal betweenness centrality in dynamic graphs. *International Journal of Data Science and Analytics*, 9(3):257–272, 2020.

46 Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

47 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016.

Barriers for Faster Dimensionality Reduction

Ora Nova Fandina
Aarhus University, Denmark

Mikael Møller Høgsgaard
Aarhus University, Denmark

Kasper Green Larsen
Aarhus University, Denmark

Abstract

The Johnson-Lindenstrauss transform allows one to embed a dataset of n points in $\mathbb{R}^d$ into $\mathbb{R}^m$, while preserving the pairwise distance between any pair of points up to a factor $(1 \pm \varepsilon)$, provided that $m = \Omega(\varepsilon^{-2} \lg n)$. The transform has found an overwhelming number of algorithmic applications, allowing to speed up algorithms and reducing memory consumption at the price of a small loss in accuracy. A central line of research on such transforms, focus on developing fast embedding algorithms, with the classic example being the Fast JL transform by Ailon and Chazelle. All known such algorithms have an embedding time of $\Omega(d \lg d)$, but no lower bounds rule out a clean $O(d)$ embedding time. In this work, we establish the first non-trivial lower bounds (of magnitude $\Omega(m \lg m)$) for a large class of embedding algorithms, including in particular most known upper bounds.

2012 ACM Subject Classification Theory of computation → Random projections and metric embeddings

Keywords and phrases Dimensional reduction, Lower bound, Linear Circuits

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.31

Related Version *Full Version*: https://arxiv.org/abs/2207.03304

Funding *Ora Nova Fandina*: Independent Research Fund Denmark (DFF) Sapere Aude Research Leader grant No 9064-00068B.
Mikael Møller Høgsgaard: Independent Research Fund Denmark (DFF) Sapere Aude Research Leader grant No 9064-00068B.
Kasper Green Larsen: Independent Research Fund Denmark (DFF) Sapere Aude Research Leader grant No 9064-00068B.

1 Introduction

Working with high dimensional data can be both costly in memory and computational power, motivating the study of dimensionality reduction techniques. The goal of dimensionality reduction is to take a high dimensional dataset X and embed it to a dataset Y in a lower dimensional space. If Y approximately preserves similarities between points in X, then one may use Y as input to an algorithm in place of X to save both memory and computation time at the cost of a small inaccuracy in ones output. A greatly celebrated dimensionality reduction result is the Johnson-Lindenstrauss lemma [18], which states: For any fixed $X \subset \mathbb{R}^d$, with the size of X being n, and any distortion $0 < \varepsilon < 1$, there exists a map $f : X \to \mathbb{R}^m$ such that for all $x, y \in X$

$$\|f(x) - f(y)\|_2 \in (1 \pm \varepsilon)\|x - y\|_2,$$

with m being $\Theta(\varepsilon^{-2} \lg n)$ [18, 25]. Thus the mapping is approximately preserving the Euclidean distances between the points in X in the lower dimensional space $\mathbb{R}^m$. The

© Ora Nova Fandina, Mikael Møller Høgsgaard, and Kasper Green Larsen;
licensed under Creative Commons License CC-BY 4.0

property of preserving pairwise distances via the Johnson-Lindenstrauss lemma have found great use in many applications, for instance as a preprocessing step to speed up machine learning algorithms.

A standard approach for obtaining an embedding f satisfying the above, is to pick a random $m \times d$ matrix A with each entry being i.i.d. $N(0,1)$ distributed [15] (or uniform $-1/1$ [5]) and embedding any input $x \in X$ to $f(x) = m^{-1/2}Ax$. Computing such an embedding thus takes $O(md)$ time. In some applications of dimensionality reduction, this becomes the bottleneck in the running time, thus motivating faster embedding algorithms. The work on faster dimensionality reduction in Euclidian space can be divided roughly into two categories: 1) using sparse embedding matrices A, or 2), using matrices A with special structure that allows fast matrix-vector multiplication. In both cases, the fastest embedding algorithms use super-linear time in the input dimensionality in the worst case. For sparse matrices, there is near-tight lower bound by Nelson and Nguyen [28] showing that the embedding time cannot be reduced below roughly $\Omega(d\varepsilon^{-1}\lg n)$. For structured matrices, the fastest embeddings use at least $\Omega(d\lg m)$ time, however in this case there are no lower bounds ruling out faster embeddings that could conceivably embed a vector in $O(d)$ time see e.g. [10, 17]. Working towards such lower bounds is the focus of this work.

Our Contributions

In this work, we establish the first non-trivial lower bounds on the time required for dimensionality reduction in Euclidian space when not restricted to using sparse matrices to perform the embedding. Focusing on the case of $d = cm$, for a constant $c > 1$ and optimal $m = O(\varepsilon^{-2}\lg n)$, we prove that a *large class* of embedding algorithms, including most known upper bounds, must use time $\Omega(m\lg m)$. This coincides with known upper bounds for several tradeoffs between ε and n. In addition to establishing a first lower bound, we believe our careful definition of the class of algorithms that the lower bound applies to, shines light on the barriers faced when developing fast embedding algorithms.

In the following section, we survey previous work and formally present our results.

1.1 Fast Dimensionality Reduction

As mentioned above, the previous work on fast dimensionality reduction can be divided into two categories, either based on sparse matrices or on structured matrices. We elaborate on these approaches in the following.

Sparse JL

The basic idea in sparse JL embeddings, is to use an embedding matrix A with only $s < m$ non-zeros per column. With such a matrix A, the product Ax can be computed trivially in $O(sd)$ time rather than $O(md)$, thus speeding up the embedding. Moreover, if x itself has few non-zeros, then the product may even be computed in $O(s\|x\|_0)$ time, where $\|x\|_0$ is the number of non-zeros in x. Using sparse embedding matrices was initiated by [1] and culminated with the current state-of-the-art embedding by Kane and Nelson [21] who showed that it suffices to pick a matrix A having $s = O(\varepsilon^{-1}\lg n)$ random entries (without replacement) in each column set uniformly and independently to $-1/1$ and embedding a vector x to $s^{-1/2}Ax$. Moreover, this nearly matches a sparsity lower bound by Nelson and Nguyen [28] who showed that any sparse embedding matrix must have $s = \Omega(\varepsilon^{-1}\lg n/\lg(1/\varepsilon))$ non-zeros per column. Another line of research in this direction, studies sparsities s below the lower bound by Nelson and Nguyen. For instance, Feature Hashing [34] considers the

extreme case of $s = 1$. Of course, such embeddings cannot work for all data sets X. However, as shown by Weinberger et al. [34] and later refined by Kamma et al. [11] and generalized to $s > 1$ by Jagadeesan [16], one can use extremely sparse embedding matrices, provided that for all pairwise difference vectors $z = x - y$ for $x, y \in X$, the ratio $\|z\|_\infty / \|z\|_2$ is small. That is, there are no single large coordinates in z.

Fast JL

The second line of research on fast embeddings exploits structured matrices A with fast matrix-vector multiplication algorithms. Ailon and Chazelle [2] initiated this direction by introducing the FastJL transform. FastJL embeds a vector by computing a product $m^{-1/2}PHDx$, where P is a sparse matrix, H is the normalized $d \times d$ Hadamard matrix and D is a diagonal matrix with random signs on the diagonal. The trick is that computing Dx can be done in $O(d)$ time and computing $H(Dx)$ takes only $O(d \lg d)$ time by exploiting the structure of the Hadamard matrix. Finally, the transformation HDx has the effect of "smoothening" out the coordinates of the input vector, making the ratio $\|HDx\|_\infty / \|HDx\|_2$ small. This is precisely the setup allowing very sparse embedding matrices. Concretely, Ailon and Chazelle [2] showed that it suffices to let each entry in P be non-zero with probability $q = O((\lg^2 n)/d)$, resulting in a total embedding time of $O(d \lg d + m \lg^2 n)$. Their analysis was recently refined by Fandina et al. [10], showing that the sparsity parameter q can be reduced further. Numerous other embeddings exploiting structured matrices has since then been introduced [22, 8, 3, 6], including for instance embeddings based on Toeplitz matrices [14, 32, 12] and the Kac random walk [19, 17]. If one insists on optimal $m = O(\varepsilon^{-2} \lg n)$ dimensions in the embedding, then the current state-of-the-art is either the FastJL transform or the Kac random walk depending on the relationship between n and ε. However none of these are faster than $O(d \lg m)$ for any tradeoff between ε and n.

Unlike the sparse matrix case, there are no known lower bounds ruling out e.g. $O(d)$ time embeddings via structured matrices. Naturally, the reason for this, is that it is much harder to prove lower bounds for general embedding algorithms that exploit structured matrices than merely bounding the sparsity of the embedding matrix. In fact, proving super-linear lower bounds for general linear circuits (which capture current embedding algorithms) is a major open question in complexity theory. In light of this obstacle, which we will elaborate on in Section 1.3, we identify common traits in most known upper bounds that we exploit to prove lower bounds for dimensionality reduction. In the following, we formally define the model under which we prove our lower bound.

1.2 Formal Lower Bound

As mentioned earlier, our lower bound holds for a large class of dimensionality reducing maps. This class is captured by a certain scaling parameter. Concretely, we define a ScaledJL-matrix as follows:

▶ **Definition 1.** *Let $0 < \varepsilon, \delta < 1$ and $s \in \mathbb{N}$. A stochastic matrix $A \in \mathbb{R}^{m \times d}$ is said to be a ScaledJL(ε, δ, s)-matrix, if for any $x \in \mathbb{R}^d$ we have that*

$$\mathbb{P}_A \left[\left\| s^{-1/2} Ax \right\|_2^2 \notin (1 \pm \varepsilon) \|x\|_2^2 \right] < \delta.$$

Let us remark a few things about Definition 1. First, we assume that a ScaledJL(ε, δ, s)-matrix A is such that $s^{-1/2}A$ preserves the (squared) norm of any single vector x up to $(1 \pm \varepsilon)$ except with probability δ. This is the standard definition of a distributional Johnson-Lindenstrauss

transform and all known upper bounds give such a guarantee. In greater detail, known upper bounds prove the distributional guarantee and then sets $\delta < 1/n^2$ and apply a union bound over all $z = x - y$ for $x, y \in X$ to conclude that the embedding preserves all pairwise (squared) distances among vectors in X. In this work, we focus on the squared distance as it simplifies calculations and anyways only changes ε by a constant factor. The non-standard thing in Definition 1 is the scaling parameter s. Of course, such a scaling parameter can also be implicitly hidden in A by scaling all entries of A by $s^{-1/2}$. To explain the role of s in our model, we need to first introduce a linear circuit/algorithm as defined e.g. by Morgenstern:

▶ **Definition 2** ([26]). *A linear algorithm takes as an input $1, x_1, \ldots, x_d \in \mathbb{R}$ and proceeds in $t > 0$ steps. In the l'th step the algorithm computes x_{d+l} by $x_{d+l} = \lambda_{d+l} x_j + \mu_{d+l} x_i$ for some pair of indices $i, j < d + l$, where $\lambda_{d+l}, \mu_{d+l} \in \mathbb{R}$.*

We say that a linear algorithm computes a linear transformation $B \in \mathbb{R}^{m \times d}$ if there exist indices $1 \leq k_1, \ldots, k_m \leq d + t$ such that: $(Bx)_1 = x_{k_1}, \ldots, (Bx)_m = x_{k_m}$ for every possible input $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$.

Note that the number of steps t determines the number of operations performed by the algorithm (up to a factor 3). Proving super-linear lower bounds for linear algorithms in the sense of Definition 2, is a major open problem [31]. Thus several previous works [27, 7] have considered restrictions where the coefficients λ and μ are bounded in absolute value by a constant r independent of m and d. This is crucially necessary if one wants to avoid the long-standing complexity theoretic barriers further elaborated on in Section 1.3.

With this in mind, the role of s in our definition of ScaledJL(ε, δ, s)-matrix becomes clearer. Concretely, if we consider an embedding $s^{-1/2}Ax$, then we think of A as being computable by a linear algorithm/circuit where all coefficients λ_i and μ_i are bounded by a constant. This naturally leads to a scaling factor $s^{-1/2}$ for some s. Such a scaling also occurs in most known upper bounds. Let us first state our main lower bound result and then discuss how it relates to known constructions:

▶ **Theorem 3.** *Let $A \in \mathbb{R}^{m \times d}$ be a ScaledJL(ε, δ, s)-matrix for $\varepsilon \leq 1/4$, $\delta \leq C$ (C being some universal constant), $s \in \mathbb{N}$, $m = \Theta(\varepsilon^{-2} \lg(1/\delta))$ and $d \geq m$, then the expected (over the random choice of A) minimum number of operations needed for any linear algorithm that computes the transformation A for all $x \in \mathbb{R}^d$ with $|\lambda_i|, |\mu_i| \leq 1$ for all i is $\Omega(m \lg s)$.*

Let us briefly argue that most known constructions are of the form captured by the lower bound and the definition of a ScaledJL(ε, δ, s)-matrix. Concretely, these upper bounds have $\lg s = \Omega(\lg m)$ and thus our lower bound shows that it must take $\Omega(m \lg m)$ operations to compute these embeddings, even if more clever linear algorithms could be devised. As an example of an upper bound, consider first the classic JL construction using a matrix A with i.i.d. random $-1/1$ entries and a scaling of $s^{-1/2} = m^{-1/2}$. In this case, the matrix A can clearly be computed by a linear algorithm using coefficients bounded by 1 in absolute value (just carry out the trivial algorithm). So it falls under the definition of a ScaledJL(ε, δ, s)-matrix with $s = m$. Next consider embeddings based on Toeplitz matrices [14, 32, 12]. Here we embed as $m^{-1/2}TDx$, where D is a diagonal with random signs and T is a Toeplitz matrix with random signs on its diagonals. The matrix T can be computed via a fast Fourier transform using coefficients bounded by a constant. Hence the construction also falls under the definition of ScaledJL(ε, δ, s)-matrix with $s = m$. We could also consider the sparse JL transform by Kane and Nelson [21]. Their construction uses an embedding matrix where each column has $t = \Theta(\varepsilon^{-1} \lg n)$ non-zero entries, each of magnitude $t^{-1/2}$. Such a sparse embedding is typically computed by moving the scaling $t^{-1/2}$ outside and then

doing the straight-forward sparse matrix-vector multiplication using constant magnitude coefficients. It thus falls under the definition of a ScaledJL(ε, δ, s)-matrix with $s = t = \Theta(\varepsilon m)$. This has $\lg s = \Omega(\lg m)$ when m is optimal $O(\varepsilon^{-2} \lg n)$. Finally, consider for instance the $m^{-1/2}PHD$ construction by Ailon and Chazelle [2]. They use the *normalized* Hadamard matrix, i.e. all entries in H are scaled down by $d^{-1/2}$. If we move that scaling factor outside, as $(md)^{-1/2}P\bar{H}D$, then $\bar{H}$ is computed recursively using coefficients of 1 and -1. The entries of P are $b \cdot N(0, q^{-1})$ distributed, where b is a Bernoulli random variable with success probability q for a $q > \lg(1/\delta)/d$. With high probability, no entry of P is thus larger than about $O(\sqrt{d})$. Moving this scaling factor outside, it cancels out with the $d^{-1/2}$ from the Hadamard matrix and then P can also be computed using coefficients bounded by a constant and the final algorithm is a ScaledJL(ε, δ, s)-matrix with $s = \Theta(m)$. Common to all these approaches, is that they project onto something that resembles a random m-dimensional subspace. Intuitively, such a matrix should have m rows all of norm about $\sqrt{d/m}$. With d columns, this would imply that each entry should be about $m^{-1/2}$ in magnitude. Moving the scaling factor outside to have constant magnitude entries, results in the $m^{-1/2}$ scaling factor observed in all these upper bounds.

Thus many known upper bounds fall under the definition of a ScaledJL(ε, δ, s)-matrix with a scaling s satisfying $\lg s = \Omega(\lg m)$. Theorem 3 therefore sheds light on why they all require $\Omega(m \lg m)$ time (which is $\omega(d)$ when $d = O(m)$). Let us also mention the only upper bound we are aware of, that does not seem to suffer from the lower bound. In the Kac JL transform [19, 17], one embeds a vector by repeatedly picking two random coordinates, among the d input coordinates, and performing a random rotation on the two. After sufficiently many steps ($\Omega(d \lg d + m \lg n)$ in the current analysis), all but the first m coordinates are discarded and those m coordinates are scaled by $\sqrt{d/m}$. While seemingly not being captured by the lower bound, we remark that the analysis of Kac JL cannot be sharpened to $o(d \lg d)$ steps as otherwise, by a coupon collector argument, there is a vector e_i among $e_{m+1}, \ldots, e_d$ whose coordinate i is never involved in a rotation and hence e_i is embedded to 0.

Of course, it would have been more natural, if our lower bound in Theorem 3 only required bounded coefficients in the linear algorithm, not that there is also a scaling parameter $s^{-1/2}$. Unfortunately, as we argue in Section 1.3, it seems unlikely that we can establish such a lower bound using current techniques. We thus believe our results can be seen in two ways: 1) as providing strong evidence that FastJL constructions cannot be made much faster, or 2) as pointing towards a direction for further improvements, by trying to design embeddings where a constant scaling parameter s suffices, or super-constant coefficients are used when computing the embedding, or perhaps using non-linearity.

1.3 Barriers for Linear Algorithm Lower Bounds

Proving super-linear unconditional lower bounds is one of the biggest barriers in many areas of complexity theory, including in particular for linear operators. A natural computational model for computing linear operators is a linear algorithm, a.k.a. linear circuit, as in Definition 2. While being a very natural model of computation for linear operators, capturing in particular all known JL constructions, it suffers from a lack of tools for proving lower bounds (without any assumptions on coefficients). Concretely, there are still no super-linear size lower bounds, even for circuits restricted to logarithmic depth. Moreover, this road block is not for lack of trying. For instance, already in 1977, Valiant [31] introduced the notion of *matrix rigidity*. Loosely stated, the rigidity of a square matrix (corresponding to a linear operator) $A \in \mathbb{R}^{n \times n}$, is the minimum number of entries in A that needs to be changed to reduce its rank below $n/2$. Valiant showed that any explicit matrix A with

rigidity $\Omega(n^2/\lg\lg n)$ cannot have a linear-sized and log-depth linear circuit for computing the corresponding linear operator. Matrix rigidity has since then been the topic of much research, see e.g. [13, 4, 30, 9], however none of these works lead to super-linear lower bounds (also when considering rectangular matrices) for explicit matrices, despite the fact that a random matrix has high rigidity with high probability.

Bounded Coefficients

In light of the above strong barriers for proving lower bounds for linear circuits, a natural restriction to the computational model, is to assume that all coefficients λ_i and μ_i used by the gates are bounded in absolute value by a constant r. Indeed, if we enforce such a restriction, then Morgenstern [27] for instance proved an $\Omega(n \lg n)$ lower bound on the size of any linear circuit computing the $n \times n$ *unnormalized* fast Fourier transform. A lower bound for the size of circuits with bound coefficients by Pudlak [29] yields a similar lower bound for the unnormalized fast Fourier and Hadamard transform. Similarly, Chazelle [7] proved $\Omega(n \lg n)$ lower bounds for linear circuits, with bounded integer coefficients, for computing linear transformation corresponding to incidence matrices for various geometric range searching problems. Common to these techniques, is that they relate the circuit complexity to the eigenvalues of the corresponding matrix A. In particular, the lower bounds one obtains peak at $\Omega(\ell \lg \gamma_\ell)$, where γ_ℓ denotes the ℓ'th largest eigenvalue of $A^T A$.

Now in the context of dimensionality reduction, an embedding matrix $A \in \mathbb{R}^{m \times d}$ can have at most m non-zero eigenvalues. This means that lower bounds obtained via these techniques will be proportional to only $\Omega(m \lg \gamma_\ell)$ for an $\ell \in \Theta(m)$. Since the size of the circuit is already at least d, it makes most sense from a lower bound point of view to consider setups where m and d are within constant factors. However, since embedding matrices A must preserve the norm of standard unit vectors e_i, their columns will have norms of magnitude $(1 \pm \varepsilon)$. This implies that the trace of $A^T A$ is $d(1 \pm \varepsilon) = \Theta(m)$. Since the trace of $A^T A$ equals the sum of its eigenvalues, we get for $\ell \in \Theta(m)$ that γ_ℓ is at best a constant. Thus the lower bounds we may hope to obtain are only $\Omega(m)$, i.e. trivial. Thus considering only the restriction to have coefficients bounded by a constant is insufficient for proving non-trivial lower bounds using known techniques.

Output Scaling

Having observed the above, we examined existing FastJL constructions and found a common trait in most of them: they embed a vector x by computing $s^{-1/2}Ax$ for some scaling factor s and matrix A, where A can be computed efficiently by a linear circuit using coefficients of constant magnitude. Given the obstacles mentioned above, we thus settled on proving lower bounds for embeddings that follow this template, resulting in Theorem 3 above.

2 Lower Bound for Linear Algorithms

The goal of this section is to prove our lower bound from Theorem 3 on the operations needed for any linear algorithm computing a ScaledJL(ε, δ, s)-matrix. We state a stronger version of the theorem here:

▶ **Theorem 4.** *Let $A \in \mathbb{R}^{m \times d}$ be a ScaledJL(ε, δ, s)-matrix for $\varepsilon \leq 1/4$, $\delta \leq C$ (C being some universal constant), $s \in \mathbb{N}$ and $t\varepsilon^{-2}\lg(1/\delta) = m$, $t \geq 1$ and $d \geq m$, then the expected (over the random choice of A) minimum number of operations needed for any linear algorithm computing Ax for any $x \in \mathbb{R}^d$ with $|\lambda_i|, |\mu_i| < r$ for all i and $r > 1/2$, is $\Omega(m \lg(s/t^2)/(t \lg(2r))$.*

We notice that Theorem 3 is a special case of Theorem 4 where r is set equal to 1 and $t = \Theta(1)$.

The main tool for proving Theorem 4 is a lemma by Morgenstern relating the operations needed by a linear algorithm computing a linear transformation B to the determinants of square submatrices of B:

▶ **Lemma 5** ([27]). *Let B be a real matrix and let $\Delta(B)$ denote the maximum over the absolute value of the determinant of any square submatrix of B. A linear algorithm computing the linear transformation B, with $|\lambda_i|, |\mu_i| < r$ for all i and $r > 1/2$, must use at least $\lg(\Delta(B))/\lg(2r)$ operations.*

Using Lemma 5 as our offset, our goal is thus to show that any ScaledJL(ε, δ, s)-matrix A must have a submatrix whose determinant is in the order of $s^{\Omega(m)}$. Since A is allowed to be stochastic and fail to preserve the norm of a vector x with probability δ, we only prove that this holds with constant probability over A:

▶ **Lemma 6.** *Let $A \in \mathbb{R}^{m\times d}$ be a ScaledJL(ε, δ, s)-matrix for $\varepsilon \leq 1/4$, $\delta \leq C$ (C being some universal constant), $s \in \mathbb{N}$ and $t\varepsilon^{-2}\lg(1/\delta) = m$, $t \geq 1$ and $d \geq m$, then there exist a set $S \subseteq supp(A)$ such that $\mathbb{P}_A[S] \geq 1/2$ and for $B \in S$ it holds that there exists a square submatrix F of B such that*

$$|\det(F)| \geq \left(c^2 s/(3(et)^2)\right)^{\lceil cm/t\rceil/2}$$

where c is some universal constant less than 1.

The proof of Theorem 4 follows immediately from the above two lemmas:

Proof of Theorem 4. Let A be a ScaledJL(ε, δ, s)-matrix. Lemma 6 gives the existence of a set $S \subseteq \text{supp}(A)$ with $\mathbb{P}_A[S] \geq 1/2$ and for $B \in S$, B has a square submatrix F such that $|\det(F)| \geq \left(c^2 s/(3(et)^2)\right)^{\lceil cm/t\rceil/2}$ implying that $\Delta(B) \geq \left(c^2 s/(3(et)^2)\right)^{\lceil cm/t\rceil/2}$. It now follows by Lemma 5 that a linear algorithm calculating Bx for all $x \in \mathbb{R}^d$ must use $\lg(\Delta(B))/\lg(2r)$ operations. Since $\lg(\Delta(B)) \geq (\lceil cm/t\rceil)\lg(c^2 s/(3(et)^2))/2 = \Omega(m\lg(s/t^2)/t)$ we get that $\lg(\Delta(B))/\lg(2r) = \Omega(m\lg(s/t^2)/(t\lg(2r)))$. Thus we conclude, since $\mathbb{P}_A[S] \geq 1/2$, that the expected number of operations needed by any linear algorithm computing the transformation A is $\Omega(m\lg(s/t^2)/(t\lg(2r)))$, which concludes the proof of Theorem 4. ◀

The main challenge we face is thus establishing Lemma 6, i.e. proving that for any ScaledJL(ε, δ, s)-matrix A, it is often the case that A has a square submatrix of large determinant. This is the focus of the next section.

2.1 Submatrix with Large Determinant (Proof Lemma 6)

To prove Lemma 6, we have to show that with probability at least $1/2$, a ScaledJL(ε, δ, s)-matrix has a square submatrix with an $\left(c^2 s/(3(et)^2)\right)^{\lceil cm/t\rceil/2}$ large determinant. For this, we will use a technical lemma from [23] which relates the eigenvalues of $B^T B$ to the determinants of square submatrices of B:

▶ **Lemma 7.** *([23] proof of Theorem 10) For $B \in \mathbb{R}^{m\times d}$, with $m \leq d$, let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m \geq 0$ denote the eigenvalues of $B^T B$. For all positive integers $l \leq m$, there exists a square submatrix $F \in \mathbb{R}^{l\times l}$ of B such that*

$$|\det(F)| \geq \sqrt{\frac{\prod_{i=1}^{l}\lambda_i}{\binom{d}{l}\binom{m}{l}}}.$$

By the above lemma, we can reduce the problem of finding a square submatrix of a ScaledJL(ε, δ, s)-matrix A with large determinant, to lower bounding the eigenvalues of a A^TA. Using $\lambda_i(B^TB)$ to denote the i'th largest eigenvalue of B^TB, this is precisely the contents of the following lemma:

▶ **Lemma 8.** *Let $A \in \mathbb{R}^{m\times d}$ be a ScaledJL(ε, δ, s)-matrix for $\varepsilon \leq 1/4$, $\delta \leq C$(C being some universal constant) and $s \in \mathbb{N}$, $t\varepsilon^{-2}\lg(1/\delta) = m$, $t \geq 1$ and $d \geq m$, then there exist a set $S \subseteq supp(A)$ such that $\mathbb{P}_A[S] \geq 1/2$ and for $B \in S$ it holds that*

$$\lambda_{\lceil cm/t\rceil}(B^TB) \geq ds/(3m)$$

where c is some universal constant less than 1.

Before we give the proof of Lemma 8, let us see that it suffices to finish the proof of Lemma 6:

Proof of Lemma 6. Let A be ScaledJL(ε, δ, s)-matrix such that the conditions of Lemma 8 are met. We then have for B in the set S described in Lemma 8 that the $l = \lceil cm/t\rceil$'th largest eigenvalue of B^TB is at least $ds/(3m)$. Now by Lemma 7. we have that there exist a square submatrix $F \in \mathbb{R}^{l\times l}$ of B such that $|\det(F)| \geq (\prod_{i=1}^{l}\lambda_i/\binom{d}{l}\binom{m}{l})^{1/2}$. Now using these two properties combined with $\binom{n}{k} \leq (en/k)^k$ and $l \geq cm/t$ we get that

$$|\det(F)| \geq \left(\prod_{i=1}^{l}\lambda_i/\left(\binom{d}{l}\binom{m}{l}\right)\right)^{1/2} \geq \left(dsl^2/(3e^2dm^2)\right)^{l/2} \geq \left(c^2s/(3(et)^2)\right)^{\lceil cm/t\rceil/2}.$$

Thus Lemma 6 follows by the conditions in Lemma 6 and Lemma 8 on the ScaledJL(ε, δ, s)-matrix being the same. ◀

After having established the above connection between eigenvalues and linear algorithms, we are left with proving Lemma 8, i.e. to show that for a ScaledJL(ε, δ, s)-matrix A, it is often the case that A^TA has many large eigenvalues. We first give an overview of the main ideas in the proof, before proceeding to give the formal details.

Proof Overview

The proof of Lemma 8 is at a high level inspired by methods used in [24]. The main result of [24] was a lower bound of $m = \Omega(\varepsilon^{-2}\lg n)$ on the embedding dimension of any linear dimensionality reducing map. Their lower bound was proved for a "hard" set of vectors consisting of the standard basis vectors and several independent Gaussian vectors. The standard basis vectors were used to lower bound the trace $Tr(A^TA)$ where A is the full embedding matrix (including any scaling factors), whereas the Gaussian vectors were used to upper bound the squared Frobenius norm $\|A^TA\|_F^2$. Since $Tr(A^TA)$ is the sum of the eigenvalues of A^TA and $\|A^TA\|_F^2$ is the sum of squared eigenvalues, one cannot have a large $Tr(A^TA)$ and a small $\|A^TA\|_F^2$ without having many non-zero eigenvalues. Their lower bound on m follows by observing that the number of non-zero eigenvalues equals the rank of A, and the rank cannot exceed m. We remark that the idea of using Gaussian vectors as a hard instance was also seen in [20].

Compared to the proof above, we need to show something stronger. More precisely, the previous work merely showed that there are $\Omega(\varepsilon^{-2}\lg n)$ non-zero eigenvalues. We need to show that there are $\Omega(\varepsilon^{-2}\lg n)$ eigenvalues that are all at least $ds/(3m)$ large. This requires a more refined analysis and the introduction of the scaling parameter $s^{-1/2}$ in the embedding $s^{-1/2}Ax$ as in the definition of a ScaledJL(ε, δ, s)-matrix.

The hard instance in our lower bound is also the standard basis vectors $e_1, \ldots, e_d$ in $\mathbb{R}^d$ together with a Gaussian distributed vector $g \in \mathbb{R}^d$. By Markov's inequality, we get that the following two events hold simultaneous with constant probability over the random choice of A: The number of basis vectors whose norm is preserved, i.e. $|\{i : \|Ae_i/\sqrt{s}\|^2 \in (1 \pm \varepsilon)\}|$, is $\Omega(d)$, and secondly, the probability that the random Gaussian vector has its norm preserved satisfies $\mathbb{P}_g[\|Ag/\sqrt{s}\|^2 / \in (1 \pm \varepsilon) \|g\|^2] \geq 1 - \Theta(\delta)$. Thus if we now consider an outcome B of A which satisfies these two relations, we get by $|\{i : \|Be_i/\sqrt{s}\|^2 \in (1 \pm \varepsilon)\}| = \Omega(d)$ that the trace of $B^T B$, which is equal to the sum of the eigenvalues $B^T B$, is $\Omega(ds)$. Now by $\|Bg/\sqrt{s}\|^2$ being in $(1 \pm \varepsilon) \|g\|^2$ and $\|g\|^2$ being in $(1 \pm \varepsilon)d$, both with probability least $1 - \delta^{\Theta(1)}$ over g, we also get with probability at least $1 - \delta^{\Theta(1)}$ over g that $\|Bg\|^2 \in (1 \pm \Theta(\varepsilon))ds$.

Now using the lower bound $\sum \lambda(B^T B)_i = \Omega(ds)$ and the fact that $B^T B$ has at most m non-zero eigenvalues, we get that the sum of the eigenvalues larger than $ds/(3m)$ is at least $\Omega(ds) - m(ds/(3m)) = \Omega(ds)$ (provided that we can prove a large enough constant in the $\Omega(ds)$ notation). However, we also need to prove that there are not just a few such eigenvalues that are huge and account for most of the sum. For this, let l denote the number of eigenvalues that are greater than or equal to $ds/(3m)$.

To prove a lower bound on l, we first use anti-concentration inequalities to relate the distribution of $\|Bg\|^2$ to $Tr(B^T B)$, obtaining an upper bound on $\|B^T B\|_F^2 = \sum \lambda(B^T B)_i^2 \leq O((ds)^2/m)$ (like in previous work). Using the upper bound on $\sum \lambda(B^T B)_i^2$ and Cauchy-Schwartz, we then conclude that the sum of the eigenvalues larger than $ds/(3m)$ is at most $\Theta(ds\sqrt{l/m})$ - hence combining the lower and upper bound on the sum of the eigenvalues larger than $ds/(3m)$, we get that $\Theta(ds\sqrt{l/m}) = \Omega(ds)$, so we conclude that $l = \Omega(m)$ as wanted. We remark that while this last part of our proof carries some resemblance to that in [24], we believe that the whole reduction above, reducing the problem to arguing that the embedding matrix must have many large eigenvalues, is highly novel in its own right.

Preliminaries

To prove Lemma 8, we need the following two concentration bounds for normal distributed random variables.

▶ **Lemma 9** ([35]). *Let $g_1, \ldots, g_d$ be independent $N(0,1)$ random variables and $u_1, \ldots, u_d$ be non-negative numbers, then for constants $c_1 \leq 1$ and $C_1 \geq 1$ we have that*

$$c_1 \exp\left(-C_1 x^2/\|u\|_2^2\right) \leq \mathbb{P}\left[\sum_{i=1}^d u_i(g_i^2 - 1) \geq x\right], \quad \forall 0 \leq x$$

$$c_1 \exp\left(-C_1 x^2/\|u\|_2^2\right) \leq \mathbb{P}\left[\sum_{i=1}^d u_i(g_i^2 - 1) \leq -x\right], \quad \forall 0 \leq x \leq c_1 \|u\|_2^2/\|u\|_\infty.$$

▶ **Lemma 10** (Example 2.11 [33]). *Let $g_1, \ldots, g_d$ be independent $N(0,1)$ random variables then*

$$\mathbb{P}\left[\left|\sum_{k=1}^d g_i^2 - d\right| \geq \alpha d\right] \leq 2e^{-d\alpha^2/8}, \quad \text{for all } \alpha \in (0,1).$$

Proof of Lemma 8

We are now ready to give the proof of Lemma 8.

Proof. Let $A \in \mathbb{R}^{m\times d}$ be a ScaledJL(ε,δ,s)-matrix for $\varepsilon \leq 1/4$ and $\delta \leq C$ (where C is a constant to be fixed later), $t\varepsilon^{-2}\lg(1/\delta) = m$ and $d \geq m$.

Let $e_1,\ldots,e_d$ be the standard basis vectors in $\mathbb{R}^d$. Let further $\mathbb{P}_g$ denote the measure of a standard Gaussian random vector $g \in \mathbb{R}^d$ independent of A. We now claim the existence of a set of matrices S such that $A \in S$ holds with probability at least $1/2$ and for $B \in S$, we have that

$$|\{i : |\left\|Be_i/\sqrt{s}\right\|^2 - \|e_i\|^2| > \varepsilon\|e_i\|^2\}| < 4\delta d \tag{1}$$

and

$$\mathbb{P}_g\left[|\left\|Bg/\sqrt{s}\right\|^2 - \|g\|^2| > \varepsilon\|g\|^2\right] < 4\delta. \tag{2}$$

To show this, define for each $i \in [d]$ the event $E_i = \{|\left\|Ae_i/\sqrt{s}\right\|^2 - \|e_i\|^2| > \varepsilon\|e_i\|^2\}$ and set X_i equal to $\mathbf{1}_{E_i}$, such that $\sum_{i=1}^d X_i = |\{i : |\left\|Be_i/\sqrt{s}\right\|^2 - \|e_i\|^2| > \varepsilon\|e_i\|^2\}|$. By the ScaledJL$(\varepsilon,\delta,s)$-matrix assumption of A, we have that

$$\mathbb{E}_A\left[\sum_{i=1}^d X_i\right] \leq \delta d$$

so by Markov's inequality we get that

$$\mathbb{P}_A\left[\sum_{i=1}^d X_i \geq 4\delta d\right] \leq 1/4$$

similarly by the ScaledJL(ε,δ,s)-matrix assumption we have that

$$\mathbb{E}_A\left[\mathbb{P}_g\left[|\left\|Ag/\sqrt{s}\right\|^2 - \|g\|^2| > \varepsilon\|g\|^2\right]\right] < \delta$$

so by applying Markov's inequality again, we get that

$$\mathbb{P}_A\left[\mathbb{P}_g\left[|\left\|Ag/\sqrt{s}\right\|^2 - \|g\|^2| > \varepsilon\|g\|^2\right] \geq 4\delta\right] \leq 1/4.$$

Now using a union bound gives that Equation (1) and Equation (2) hold simultaenously with probability at least $1/2$ as claimed.

If we can show that for $B \in S$, it holds that $\lambda(B^TB)_{\lceil cm/t\rceil} > ds/(3m)$, then we are done since the probability of A being in S is at least $1/2$. So let $B \in S$. We now notice that by Equation (1) there exist $(1-4\delta)d$ indices in $i \in [d]$ such that $(B^TB)_{i,i} \in (1\pm\varepsilon)s$. If we now let $\lambda_i(B^TB)$ denote the i'th largest eigenvalue of B^TB, we get the following lower bound on the sum of eigenvalues of B^TB (assuming $\varepsilon \leq 1/4$ and $\delta \leq C \leq 1/36$):

$$\sum_{i=1}^m \lambda_i(B^TB) = Tr(B^TB) \geq (1-\varepsilon)(1-4\delta)ds \geq 2ds/3. \tag{3}$$

Now by Cauchy-Schwartz, we also have that

$$\begin{aligned}\sum_{i=1}^m \lambda_i(B^TB) &\leq \sqrt{m\sum_{i=1}^m \lambda_i(B^TB)^2} \\ &\leq \sqrt{m\sum_{i=1}^m \lambda_i(B^TB)^2}\sqrt{\sum_{i=1}^m \lambda_i(B^TB)^2/\lambda_1} = \sqrt{m}\sum_{i=1}^m \lambda_i(B^TB)^2/\lambda_1\end{aligned} \tag{4}$$

Combining Equation (3) and Equation (4), we get that

$$(\sum_{i=1}^{m} \lambda_i(B^TB)^2)/\lambda_1(B^TB) \geq 2ds/3\sqrt{m} \geq ds/4\sqrt{m}. \tag{5}$$

Now since B was in S, we have by Equation (2) that $\|Bg\|^2 \in (1 \pm \varepsilon)\|g\|^2$ with probability at least $1 - 4\delta$ over g. At the same time, we have by Lemma 10 that for $0 < \alpha < 1$, it holds that $\|g\|^2 \in (1 \pm \alpha)d$ with probability at least $1 - 2\exp(-d\alpha^2/8)$. Now choosing $\alpha = \varepsilon$, we get that $2\exp(-d\varepsilon^2/8) \leq 2\delta^{1/8}$. By the assumption that $d \geq m \geq \varepsilon^{-2}\lg(1/\delta)$, we get that $\|g\|^2 \in (1 \pm \varepsilon)d$ with probability at least $1 - 2\delta^{1/8}$ over g. Now combining this with $\|Bg\|^2 \in (1 \pm \varepsilon)\|g\|^2$ with probability at least $1 - 4\delta$ over g, we get by a union bound that

$$\|Bg\|^2 \in (1 \pm \varepsilon)(1 \pm \varepsilon)ds = (1 - 2\varepsilon + \varepsilon^2, 1 + 2\varepsilon + \varepsilon^2)ds \tag{6}$$

with probability at least $1 - 6\delta^{1/8}$ over g.

Now using the eigenvalue decomposition of B^TB into U^TDU, where U is an orthogonal matrix and D an diagonal matrix with the eigenvalues of B^TB on its diagonal in decreasing order, and that a standard normal Gaussian vector is invariant in distribution under rotations, we obtain the following relation

$$\|Bg\|^2 - Tr(B^TB) = g^TB^TBg - Tr(B^TB) = g^TU^TDUg - Tr(B^TB) \tag{7}$$

$$\stackrel{d}{=} \tilde{g}^TD\tilde{g} - \sum_{i=1}^{d} \lambda_i(B^TB) = \sum_{i=1}^{d} \lambda_i(B^TB)(\tilde{g}_i^2 - 1). \tag{8}$$

Our next step is to relate $\sum_i \lambda_i^2(B^TB)$ to δ. Here we take two different approaches depending on $Tr(B^TB)$. c_1 and C_1 in the following are the constants of Lemma 10.

Case 1. If $Tr(B^TB) \leq (1 - 2\varepsilon + c_1/(4\sqrt{m}))ds$ then by Equation (6) (and the comment above the equation) we have with probability at least $1 - 6\delta^{1/8}$ over g that

$$\|Bg\|^2 - Tr(B^TB) \geq ((1 - 2\varepsilon + \varepsilon^2) - (1 - 2\varepsilon + c_1/(4\sqrt{m})))ds > -c_1ds/4\sqrt{m}.$$

implying that $6\delta^{1/8} \geq \mathbb{P}_g\left[\|Bg\|^2 - Tr(B^TB) \leq -c_1ds/4\sqrt{m}\right]$.

Now noticing that $c_1ds/4\sqrt{m} \leq c_1(\sum_{i=1}^{m} \lambda_i(B^TB)^2)/\lambda_1(B^TB)$ by Equation (5), we may invoke the second relation in Lemma 9 on Equation (8) to get:

$$\mathbb{P}_g\left[\|Bg\|^2 - Tr(B^TB) \leq -c_1ds/4\sqrt{m}\right] = \mathbb{P}_{\tilde{g}}\left[\sum_{i=1}^{d} \lambda_i(B^TB)(\tilde{g}_i^2 - 1) \leq -c_1ds/4\sqrt{m}\right]$$

$$\geq c_1 \exp\left(-C_1(c_1ds)^2/(16m\sum_{i=1}^{d} \lambda_i^2(B^TB))\right).$$

Yielding that $6\delta^{1/8} \geq c_1 \exp\left(-C_1(c_1ds)^2/(16m\sum_{i=1}^{d} \lambda_i^2(B^TB))\right)$.

Case 2. If $Tr(B^TB) \in [(1 - 2\varepsilon + c_1/(4\sqrt{m}))ds, \infty)$ then by Equation (6) (and the comment below the equation) we have with probability at least $1 - 6\delta^{1/8}$ over g that

$$\|Bg\|^2 - Tr(B^TB) \leq ((1 + 2\varepsilon + \varepsilon^2) - (1 - 2\varepsilon + c_1/(4\sqrt{m})))ds < 5\varepsilon ds.$$

implying that $6\delta^{1/8} \geq \mathbb{P}_g\left[\|Bg\|^2 - Tr(B^TB) \geq 5\varepsilon ds\right]$.

Now using the first relation in Lemma 9 combined with Equation (8), it follows that

$$\mathbb{P}_g\left[\|Bg\|^2 - Tr(B^TB) > 5\varepsilon ds\right]\mathbb{P}_{\tilde{g}}\left[\sum_{i=1}^{d}\lambda_i(B^TB)(\tilde{g}_i^2-1) > 5\varepsilon ds\right]$$
$$\geq c_1\exp\left(-C_1(5\varepsilon ds)^2/(\sum_{i=1}^{d}\lambda_i^2(B^TB))\right).$$

Yielding that $6\delta^{1/8} \geq c_1\exp\left(-C_1(5\varepsilon ds)^2/(\sum_{i=1}^{d}\lambda_i^2(B^TB))\right)$.

Conclusion. Now using that $m \geq \varepsilon^{-2}\lg(1/\delta)$ and $c_1 \leq 1$ it follows that $c_1^2/16m \leq 5^2\varepsilon^2$ which then implies that $C_1(c_1ds)^2/(16m\sum_{i=1}^{d}\lambda_i^2(B^TB)) \leq C_1(5\varepsilon ds)^2/(\sum_{i=1}^{d}\lambda_i^2(B^TB)$. Combining this with the conclusion of the above two cases, we get $6\delta^{1/8} \geq c_1\exp\left(-C_1(5\varepsilon ds)^2/(\sum_{i=1}^{d}\lambda_i^2(B^TB))\right)$. With this relation, choosing the universal constant $C = (c_1/6)^{16}$ (less than $1/36$ as used in Equation (3)), which implies that $c_1/(6\delta^{1/16}) \geq 1$, and using that $m = t\varepsilon^{-2}\lg(1/\delta)$, we now get that

$$\lg(6\delta^{1/8}) \geq \lg(c_1) - C_1(5\varepsilon ds)^2/(\sum_{i=1}^{d}\lambda_i^2(B^TB))$$

which implies

$$\sum_{i=1}^{d}\lambda_i^2(B^TB) \leq C_1(5\varepsilon ds)^2/(\lg(c_1/(6\delta^{1/8}))) \leq C_1 16(5\varepsilon ds)^2/\lg(1/\delta) \leq 20^2C_1t(ds)^2/m. \quad (9)$$

We now define the vector $w \in \mathbb{R}^d$ as

$$[w]_i = \begin{cases} 1 & \text{if } \lambda_i(B^TB) \geq ds/(3m) \\ 0 & \text{else} \end{cases}$$

and let l be equal to the number of non-zero entries of w. Let further λ denote the vector in $\mathbb{R}^d$ with the eigenvalues of B^TB in decreasing order. It then follows by Cauchy-Schwartz and Equation (9) that we have the following upper bound on the sum of the eigenvalues of B^TB larger than $ds/(3m)$:

$$\sum_{i:\lambda_i(B^TB)\geq ds/(3m)}\lambda_i(B^TB) = \langle\lambda, w\rangle \leq \|\lambda\|\,\|w\| = \sqrt{\sum_{i=1}^{d}\lambda_i^2(B^TB)l} \leq \sqrt{20^2C_1t(ds)^2l/m}.$$

At the same time, we get the following lower bound on the sum of the eigenvalues of B^TB larger than $ds/(3m)$ by Equation (3) and the fact that (B^TB) has rank at most m and hence at most m non-zero eigenvalues

$$\sum_{i:\lambda_i(B^TB)\geq ds/(3m)}\lambda_i(B^TB) = \sum_{i=1}^{d}\lambda_i(B^TB) - \sum_{i:\lambda_i(B^TB)<ds/(3m)}\lambda_i(B^TB)$$
$$\geq 2ds/3 - ds/3 = ds/3.$$

Hence combining the upper and lower bound we obtain that $ds/3 \leq \sqrt{20^2C_1t(ds)^2l/m}$, implying that $m/(60^2C_1t) \leq l$, which by setting c in Lemma 8 equal to $1/60^2C_1 \leq 1$ ($C_1 \geq 1$ by Lemma 9) concludes the proof of Lemma 8. ◀

References

1 Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003. doi:10.1016/S0022-0000(03)00025-4.

2 Nir Ailon and Bernard Chazelle. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39:302–322, 2009.

3 Nir Ailon and Edo Liberty. Fast dimension reduction using rademacher series on dual BCH codes. In Shang-Hua Teng, editor, *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1–9. SIAM, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347083.

4 Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2009. doi:10.1007/978-3-642-03685-9_26.

5 Rosa I. Arriaga and Santosh S. Vempala. An algorithmic theory of learning: Robust concepts and random projection. *Mach. Learn.*, 63(2):161–182, 2006. doi:10.1007/s10994-006-6265-7.

6 Stefan Bamberger and Felix Krahmer. Optimal fast johnson–lindenstrauss embeddings for large data sets. *Sampling Theory, Signal Processing, and Data Analysis*, 19(1):3, 2021. doi:10.1007/s43670-021-00003-5.

7 Bernard Chazelle. A spectral approach to lower bounds with applications to geometric searching. *SIAM Journal on Computing*, 27(2):545–556, 1998. doi:10.1137/S0097539794275665.

8 Thong T. Do, Lu Gan, Yi Chen, Nam Nguyen, and Trac D. Tran. Fast and efficient dimensionality reduction using structurally random matrices. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1821–1824, 2009. doi:10.1109/ICASSP.2009.4959960.

9 Zeev Dvir, Alexander Golovnev, and Omri Weinstein. Static data structure lower bounds imply rigidity. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 967–978. ACM, 2019. doi:10.1145/3313276.3316348.

10 Ora Nova Fandina, Mikael Møller Høgsgaard, and Kasper Green Larsen. The fast johnson-lindenstrauss transform is even faster. *CoRR*, abs/2204.01800, 2022. doi:10.48550/arXiv.2204.01800.

11 Casper Freksen, Lior Kamma, and Kasper Green Larsen. Fully understanding the hashing trick. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 5394–5404, Red Hook, NY, USA, 2018. Curran Associates Inc.

12 Casper Benjamin Freksen and Kasper Green Larsen. On using toeplitz and circulant matrices for johnson-lindenstrauss transforms. *Algorithmica*, 82(2):338–354, 2020. doi:10.1007/s00453-019-00644-y.

13 Joel Friedman. A note on matrix rigidity. *Comb.*, 13(2):235–239, 1993. doi:10.1007/BF01303207.

14 Aicke Hinrichs and Jan Vybíral. Johnson-lindenstrauss lemma for circulant matrices. *Random Structures & Algorithms*, 39(3):391–398, 2011. doi:10.1002/rsa.20360.

15 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/276698.276876.

16 Meena Jagadeesan. Understanding sparse JL for feature hashing. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15177–15187, 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/502cc2c94be1a7c4ca7ef25b8b50bc04-Abstract.html.

17 Vishesh Jain, Natesh S. Pillai, and Aaron Smith. Kac meets johnson and lindenstrauss: a memory-optimal, fast johnson-lindenstrauss transform. *CoRR*, abs/2003.10069, 2020. To appear in Annals of Applied Probability. doi:10.48550/arXiv.2003.10069.

18 William Johnson and Joram Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, January 1984. doi:10.1090/conm/026/737400.

19 Mark Kac. Foundations of kinetic theory. In *Proceedings of The third Berkeley symposium on mathematical statistics and probability*, pages 171–197. University of California Press Berkeley and Los Angeles, California, 1958.

20 Daniel M. Kane, Raghu Meka, and Jelani Nelson. Almost optimal explicit johnson-lindenstrauss families. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 628–639, 2011. doi:10.1007/978-3-642-22935-0_53.

21 Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1):4:1–4:23, 2014. doi:10.1145/2559902.

22 Felix Krahmer and Rachel Ward. New and improved johnson-lindenstrauss embeddings via the restricted isometry property. *SIAM J. Math. Anal.*, 43(3):1269–1281, 2011. doi:10.1137/100810447.

23 Kasper Green Larsen. Constructive discrepancy minimization with hereditary L2 guarantees. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, pages 48:1–48:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.48.

24 Kasper Green Larsen and Jelani Nelson. The johnson-lindenstrauss lemma is optimal for linear dimensionality reduction. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 82:1–82:11, 2016. doi:10.4230/LIPIcs.ICALP.2016.82.

25 Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 633–638. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.64.

26 Jacques Morgenstern. On linear algorithms. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 59–66. Academic Press, 1971. doi:10.1016/B978-0-12-417750-5.50009-9.

27 Jacques Morgenstern. Note on a lower bound on the linear complexity of the fast fourier transform. *J. ACM*, 20:305–306, 1973.

28 Jelani Nelson and Huy L. Nguyen. Sparsity lower bounds for dimensionality reducing maps. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 101–110. ACM, 2013. doi:10.1145/2488608.2488622.

29 Pavel Pudlák. A note on the use of determinant for proving lower bounds on the size of linear circuits. *Information Processing Letters*, 74(5):197–201, 2000. doi:10.1016/S0020-0190(00)00058-2.

30 Shubhangi Saraf and Sergey Yekhanin. Noisy interpolation of sparse polynomials, and applications. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, USA, June 8-10, 2011*, pages 86–92. IEEE Computer Society, 2011. doi:10.1109/CCC.2011.38.

31 Leslie G. Valiant. Graph-theoretic arguments in low-level complexity. In Jozef Gruska, editor, *Mathematical Foundations of Computer Science 1977, 6th Symposium, Tatranska Lomnica, Czechoslovakia, September 5-9, 1977, Proceedings*, volume 53 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 1977. doi:10.1007/3-540-08353-7_135.

32 Jan Vybiral. A variant of the johnson-lindenstrauss lemma for circulant matrices. *Journal of Functional Analysis*, 260:1096–1105, February 2010. doi:10.1016/j.jfa.2010.11.014.

33 Martin J. Wainwright. *Basic tail and concentration bounds*, pages 21–57. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019. doi:10.1017/9781108627771.002.

34 Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pages 1113–1120, 2009.

35 Anru R. Zhang and Yuchen Zhou. On the non-asymptotic and sharp lower tail bounds of random variables. *Stat*, 9(1):e314, 2020. e314 sta4.314. doi:10.1002/sta4.314.

A Regular and Complete Notion of Delay for Streaming String Transducers

Emmanuel Filiot
Université libre de Bruxelles, Belgium

Ismaël Jecker
University of Warsaw, Poland

Christof Löding
RWTH Aachen University, Germany

Sarah Winter
Université libre de Bruxelles, Belgium

Abstract

The notion of delay between finite transducers is a core element of numerous fundamental results of transducer theory. The goal of this work is to provide a similar notion for more complex abstract machines: we introduce a new notion of delay tailored to measure the similarity between streaming string transducers (SST). We show that our notion is *regular*: we design a finite automaton that can check whether the delay between any two SSTs executions is smaller than some given bound. As a consequence, our notion enjoys good decidability properties: in particular, while equivalence between non-deterministic SSTs is undecidable, we show that equivalence *up to fixed delay* is decidable. Moreover, we show that our notion has good *completeness* properties: we prove that two SSTs are equivalent if and only if they are equivalent up to some (computable) bounded delay. Together with the regularity of our delay notion, it provides an alternative proof that SSTs equivalence is decidable. Finally, the definition of our delay notion is machine-independent, as it only depends on the *origin semantics* of SSTs. As a corollary, the completeness result also holds for equivalent machine models such as deterministic two-way transducers, or MSO transducers.

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

Keywords and phrases Streaming string transducers, Delay, Origin

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.32

Related Version *Full Version*: `https://arxiv.org/abs/2205.04287`

Funding This work was partially supported by the Fonds de la Recherche Scientifique – F.R.S.-FNRS under the MIS project F451019F.
Emmanuel Filiot: Emmanuel Filiot is a senior research associate at F.R.S-FNRS.
Sarah Winter: Sarah Winter is a postdoctoral researcher at F.R.S.-FNRS.

1 Introduction

Transducers are another fundamental extension of finite automata for computing *functions*, and more generally *relations*, between structures. In this paper, we consider string-to-string transducers, which operate on (input) strings and produce (output) strings. The most basic, *finite transducers*, are obtained by adding output words on the transitions of a finite automaton [32, 21]. At the heart of several important results in the theory of finite transducers is a notion, called *delay*, allowing to finely compare, in a regular way (with a finite automaton), transducer executions. The goal of this paper is to provide such a notion for a strictly more powerful class of transducers, streaming string transducers [1], which have

© Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter;
licensed under Creative Commons License CC-BY 4.0

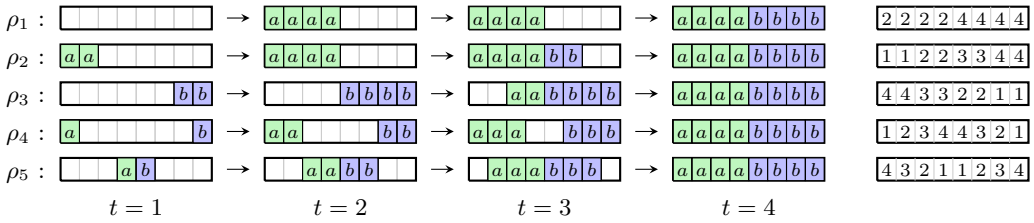


Figure 1 Five ways of producing the output sequence a^4b^4, and the corresponding origin functions.

received a lot of attention in the recent years, due to their robustness and equivalence with many other formalisms to define string-to-string functions. In particular, this paper answers positively the following high-level question:

Is it possible to measure in a regular manner how differently executions of equivalent streaming string transducers produce their outputs?

Finite transducers. We first explain how the above question is answered for finite transducers. Transitions of finite transducers over some alphabet Σ are tuples (p, σ, w, q) where p, q are states, $\sigma \in \Sigma$ is a symbol read on the input tape, and $w \in \Sigma^*$ is a word (possibly empty) written on the output tape. A finite transducer execution, i.e., a sequence of successive transitions $\rho = (p_1, \sigma_1, w_1, p_2) \dots (p_{n-1}, \sigma_n, w_n, p_n)$, operates on the input word $\mathsf{in}(\rho) = \sigma_1 \dots \sigma_n$ and produces the output word $\mathsf{out}(\rho) = w_1 \dots w_n$. The semantics of a finite transducer is the set of pairs $(\mathsf{in}(\rho), \mathsf{out}(\rho))$ for all accepting executions ρ. Two different executions ρ_1 and ρ_2 might define the same input/output pair, but can produce the output in a different way because the output words w_i occurring on the transitions of the two runs may differ, although their whole concatenation is the same.

Origin information. Differences in the way transducers produce their output are best captured by the notion of *origin information*, initially proposed for streaming string transducers [10]. An origin function maps any position of the output word to the input position where it was produced. In an execution such as ρ above, any position of w_i has origin i. Figure 1 illustrates five different ways of producing the same output word a^4b^4 (the input word of length 4 is irrelevant here and not depicted). Consider for example the execution ρ_1. When reading the first input symbol (timestep $t = 1$), nothing is produced. At timestep $t = 2$, a^4 is produced, and its positions have origin 2, as depicted to the right of the figure. The sequence b^4 is produced at timestep $t = 4$ so its corresponding positions have origin 4. Note that ρ_3, ρ_4, ρ_5 do not correspond to finite transducer executions, as a^4b^4 is not produced from left to right.

Delay for finite transducers. A natural way of comparing two finite transducer executions ρ_1, ρ_2 on the same input and producing the same output, is to compare the origin functions o_1, o_2 they induce, and in particular how much one is ahead of the other. In Figure 1, both ρ_1 and ρ_2 produce the same output from left to right, yet at different speed: ρ_1 produces bigger chunks of output at lower frequency. The *delay* between both executions is 2 for $t \in \{1, 3\}$ and 0 for $t \in \{2, 4\}$. The global delay $\mathsf{delay}(\rho_1, \rho_2)$ is defined as the maximal delay along the executions, in this case 2. Two transducers T_1 and T_2 are said to be *k-delay equivalent* if for each run ρ_1 of T_1 or T_2 the other transducer has a run ρ_2 with the same input and output satisfying $\mathsf{delay}(\rho_1, \rho_2) \leq k$. This delay notion enjoys two important properties:

Regularity: While the set $E = \{(\rho_1, \rho_2) \mid \mathsf{in}(\rho_1) = \mathsf{in}(\rho_2) \wedge \mathsf{out}(\rho_1) = \mathsf{out}(\rho_2)\}$ is not regular[1] in general for finite transducer executions, its restrictions to pairs of executions with bounded delay is regular: For every $k \in \mathbb{N}$, the set $E_k = \{(\rho_1, \rho_2) \in E \mid \mathsf{delay}(\rho_1, \rho_2) \leq k\}$ is regular [24]. As finite automata compose well with other abstract machines, this allows to use the delay in all kinds of constructions while preserving good decidability properties. For instance k-delay equivalence is decidable [24].

Completeness: For any two finite transducers T_1, T_2 defining string-to-string *functions*, there exists a computable bound k such that T_1 and T_2 are equivalent iff they are k-delay equivalent. The completeness even holds in broader classes [24], such as *finite-valued* transducers, for which there is a global bound on the number of outputs mapped to a single input.

Applications. Due to its regularity and completeness, the notion of delay, while basic, proves to be very powerful, and is omnipresent in the study of finite transducers. It provides decidable approximations of various undecidable decision problems, which reveals to be exact for broad classes of transducers, such as finite-valued transducers [24]. Various patterns characterizing important subclasses of transducers are based on the delay notion: For instance, transducers which are *sequential* [14, 9] (i.e., definable by an input-deterministic transducer); equivalent to finite union of sequential functions [27, 20]; finite-valued [29, 31]. Moreover, it has been used to show that any finite-valued transducer can be decomposed as a union of 1-valued transducers [33, 30]. Canonical notions for finite transducers are also based on delay: for input-deterministic transducers [15] and 1-valued transducers [28].

Streaming string transducers and regular functions. *Streaming string transducers* (SST) are obtained by equipping deterministic finite automata with a finite set of registers that store output [1]. Those registers cannot be tested, but are updated by concatenating them, or prepending/appending some symbols to them. E.g., consider a single-state SST with a loop which, whatever it reads as input, updates its single register O by the operation $O := aOb$, and finally outputs the whole content of O. The execution ρ_5 in Figure 1 represents an execution of this SST. Consider an equivalent single-state SST with two registers X, Y which, whatever the input, executes $X := Xa$ and $Y := bY$, and finally outputs the concatenation XY (ρ_4 in Figure 1 produces the output in this way). While equivalent, those two transducers are not equivalent if the origin information is included in the semantics.

SSTs define a robust set of string-to-string functions, called *regular functions*, which can be equivalently defined by deterministic two-way transducers [1], monadic second-order transductions [16, 22, 4], regular transducer expressions [5, 8, 18, 19], and a logic with origins [17]. Regular functions also enjoy a Krohn-Rhodes decomposition theorem [11]. SSTs have a decidable equivalence problem [26], have been applied to the verification of *list-processing programs* [2], and have been implemented for evaluating string transformations [3].

There are two natural ways of extending the delay defined for finite transducers to compare executions of SSTs with the same input and output: we can either simply compare the number of produced output letters without caring about the positions where it is produced, or we can count the number of positions whose output has already been produced in one run but not in the other. Both methods match the previously defined delay if the output is produced from left to right, but unfortunately neither preserves conjointly the regularity and

[1] Regularity here is defined as classical regularity of word languages, where a pair (ρ_1, ρ_2), are encoded as a single word over a product alphabet of transitions.

the completeness once other output production mechanisms, such as in SSTs, are considered[2]. As the basic notions of delay for SSTs fail to satisfy good properties, better ways of comparing SSTs were introduced, yet none that conjointly has good decidability *and* completeness properties. For instance, *bounded regular resynchronizers* can alter origin functions of SSTs in a controlled manner while preserving good decidability properties, but they are are not complete [13, 12]. In the restricted setting of *single-register* SSTs, a regular and complete notion of delay is defined based on word equations [25]. This approach was applied to prove a decomposition theorem for single-register SSTs, but unfortunately fails when more registers are allowed.

Contributions. We define a delay notion for comparing SST executions, based on the origin functions they induce, that is both regular (Theorem 5) and complete for fundamental SST decision problems including equivalence (Theorem 3 and Corollaries 10 and 11). This delay notion is described in the next subsection. We first present our results. Since our notion is based on origins, it more generally applies to regular functions with origin information. SSTs are known to be equivalent to deterministic two-way and MSO transducers, via origin-preserving encodings [10], so we obtain as a corollary that our delay notion is also complete for equivalence of deterministic two-way transducers and MSO transducers (Corollary 4).

The origin semantics allows us to recover decidability of several decision problems: While non-deterministic SST equivalence is undecidable, we can decide whether two non-deterministic SSTs define the same relations *with same origins*. However, asking for identical origins is very restrictive: it fails to detect equivalence if the origins are perturbed ever so slightly. This observation was already made in [24] in the context of finite transducers, where it is proposed to relax several decision problems, such as equivalence with same origins, to decision problems with *close* origins, such as equivalence up to a given delay bound k. We prove that the inclusion, equivalence and variable minimization problems up to a given delay bound are decidable for (non-deterministic) SSTs (Theorems 8 and 9), while in general those problems are undecidable. Since our delay notion is complete for equivalence of (deterministic) SSTs, the latter result provides an alternative proof that such SSTs have decidable equivalence problem, and sheds light on the intrinsic reasons why SSTs executions are equivalent.

Delay for SSTs. Our notion of delay is based on the following observation: The order of production of each output segment that is a power of a small word should have no impact on the delay. For every $\ell \in \mathbb{N}$, we introduce the measure delay_ℓ, which first decomposes the output into blocks that are powers of words smaller than or equal to ℓ. Then, at any position j ending a block (instead of *any* position), we measure the maximal difference at any timestep t, between the number of output positions at the left of j produced before timestep t in the first and second executions. We then take the maximal such value for all j. Consider ρ_4 and ρ_5 for example. delay_1 splits the output a^4b^4 of Figure 1 into two blocks, a^4 and b^4. Consider ending block position $j = 4$. At any timestep, the maximal difference of quantity of outputs produced at the left of j is always 0, because ρ_4, ρ_5 produce exactly one symbol per timestep before position $j = 4$. The same reasoning applies for ending block position $j = 8$, where here instead, exactly two symbols are produced by ρ_4, ρ_5 at any timestep, at the left of position $j = 8$. Hence $\mathsf{delay}_1(\rho_4, \rho_5) = 0$. We refer to Section 3 for a smooth introduction

[2] We refer the interested reader to the appendix for more details about those natural extensions and an explanation of why they are not satisfactory.

to our delay notion and the main results and to Section 4.1 for a proof of its completeness, and to Section 4.2 for a proof of its regularity. The completeness proof is perhaps the most involved. It is based on a key pumping lemma (Lemma 6), which intuitively states that for any SST, there exist computable bounds k and ℓ such that, if the delay delay_ℓ between two executions on the same input/output is greater than k, then those two executions can be pumped to construct two executions over the same input but with *different* outputs.

2 Preliminaries

We fix our notation. Let $\mathbb{N}$ denote the set of non-negative integers. The interval between integers a and b including resp. excluding a and b is denoted $[a, b]$ resp. (a, b).

Free monoid. Given a finite alphabet Σ, the *free monoid* over Σ is the monoid $(\Sigma^*, \cdot, \varepsilon)$ defined as follows. The set Σ^* is composed of finite sequences of elements of Σ, called *words*. The operation $\cdot$ is the usual word concatenation. The neutral element ε is the empty word.

A subset $L \subseteq \Sigma^*$ is called a *language*. Given a word $u = a_1 \cdots a_n \in \Sigma^*$, $|u|$ denotes its length, $u[i]$ denotes its ith letter a_i, $u[i, j]$ denotes its infix $a_i \cdots a_j$. Given an additional word $v = b_1 \cdots b_n \in \Sigma^*$, $u \otimes v$ denote the *convolution* $\binom{a_1 \cdots a_n}{b_1 \cdots b_n} \in (\Sigma \times \Sigma)^*$.

Automaton. A *(finite state) automaton* is a tuple $A = (\Sigma, Q, I, \Delta, F)$ composed of a finite alphabet Σ, a finite set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, and a set of transitions $\Delta \subseteq Q \times \Sigma \times Q$. A *run* of A is a sequence $\rho = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in Q(\Sigma Q)^*$ such that $(q_{i-1}, a_i, q_i) \in \Delta$ for every $1 \leq i \leq n$. The *input* of ρ is the word $a_1 a_2 \cdots a_n \in \Sigma^*$. The run ρ is called *accepting* if its first state is initial ($q_0 \in I$) and its last state is final ($q_n \in F$). The automaton A is *deterministic* if I is a singleton and Δ is expressed as a function $\delta : Q \times \Sigma \to Q$. The *language recognized* by A is the set $L(A) \subseteq \Sigma^*$ composed of the inputs of all its accepting runs.

Substitutions monoid. Given a finite alphabet Σ, a finite set $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ of *variables*, and a designated *output variable* $O \in \mathcal{X}$, the *(copyless) substitutions monoid* $(\mathcal{S}_{\mathcal{X},\Sigma}, \circ, \mathsf{Id}_\mathcal{X})$ is defined as follows. The set $\mathcal{S}_{\mathcal{X},\Sigma}$ contains all functions $\sigma : \mathcal{X} \to (\mathcal{X} \cup \Sigma)^*$ such that no variable appears twice in the image of a variable and no variable appears in the images of two distinct variables, i.e., the word $\sigma(X_1) \ldots \sigma(X_n)$ does not contain twice the same variable. The composition $\sigma_1 \circ \sigma_2$ of two substitutions $\sigma_1, \sigma_2 \in \mathcal{S}_{\mathcal{X},\Sigma}$ is obtained by first applying σ_2, and then σ_1. Formally, it is the function $\hat{\sigma}_1(\hat{\sigma}_2(\cdot))$, where each substitution σ is morphically extended to $\hat{\sigma}$ over $(\mathcal{X} \cup \Sigma)^*$ by letting $\hat{\sigma}(X) = \sigma(X)$ and $\hat{\sigma}(\alpha) = \alpha$ for $\alpha \in \Sigma$. We write σ in place of $\hat{\sigma}$. For instance, the substitution composition $(\sigma_1 \colon X \mapsto \varepsilon) \circ (\sigma_2 \colon X \mapsto aX) \circ (\sigma_3 \colon X \mapsto bXc)$ maps X to bac, since $\sigma_3(X) = bXc$, $\sigma_2(bXc) = baXc$ and $\sigma_1(baXc) = bac$. The neutral element $\mathsf{Id}_\mathcal{X}$ is the identity function mapping each variable $X \in \mathcal{X}$ to itself. Finally, we denote by σ_ε the substitution mapping each variable $X \in \mathcal{X}$ to ε. We recall that O is the designated output variable; the *output* of a substitution σ is the word $\mathsf{out}(\sigma) = (\sigma_\varepsilon \circ \sigma)(O)$.

Streaming string transducer. A *streaming string transducer* (SST) is a tuple $T = (\Sigma, Q, I, \delta, F, \mathcal{X}, O, \kappa, \kappa_F)$ where $A_T = (\Sigma, Q, I, \delta, F)$ is a deterministic automaton, called *underlying automaton* of T, $\mathcal{X}$ is a finite set of variables (also called registers), $O \in \mathcal{X}$ is a final variable, $\kappa \colon \delta \to \mathcal{S}_{\mathcal{X},\Sigma}$ is an output function, and $\kappa_F \colon Q \to \mathcal{S}_{\mathcal{X},\Sigma}$ is a final output function. A *run* of T is a run $\rho = q_0 a_1 q_1 a_2 q_2 \cdots a_n q_n \in Q(\Sigma Q)^*$ of A_T, we define $\kappa(\rho)$ as the substitution

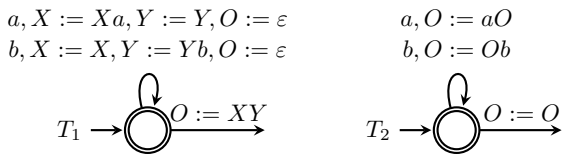


Figure 2 Two deterministic SSTs that realize the same sorting function, cf. Example 1.

obtained by composing sequentially all substitutions occurring on the transitions and the final substitutions, i.e. $\kappa(\rho) = \kappa((q_0, a_1, q_1)) \circ \kappa((q_1, a_2, q_2)) \circ \cdots \circ \kappa((q_{n-1}, a_n, q_n)) \circ \kappa_F(q_n) \in \mathcal{S}_{\mathcal{X},\Sigma}$. The *output* of the run ρ is the word $(\sigma_\varepsilon \circ \kappa(\rho))(O)$. The *transduction* $R(T)$ *recognized* (also called *realized*) by T is the set of pairs $(u, v) \in \Sigma^* \times \Sigma^*$ such that T has an accepting run with input u and output v. Non-deterministic SST are defined the same way except that A_T is non-deterministic. We emphasize that the transduction realized by a (deterministic) SST is a partial function.

▶ **Example 1.** Depicted in Figure 2 are deterministic SSTs T_1 and T_2 that realize the same sorting function $f\colon \{a, b\}^* \to \{a, b\}^*$ that maps $u \in \{a, b\}^*$ to $a^m b^n$, where m (resp. n) is the number of occurrences of a (resp. b) in u.

Origin information. A *word with origins* is a word $\tilde{u} := u \otimes o \in (\Sigma \times \mathbb{N})^*$ where each letter of u is annotated with a natural number signifying its origin in time. A *transduction with origins* is a relation $R \subseteq \Sigma^* \times (\Sigma \times \mathbb{N})^*$.

Naturally, we associate with a sequence $\lambda = \sigma_1 \sigma_2 \cdots \sigma_n \in \mathcal{S}^*_{\mathcal{X},\Sigma}$ of substitutions the output word with origins $\tilde{\mathsf{out}}(\lambda) = \mathsf{out}(\lambda) \otimes i_1 \cdots i_{|\mathsf{out}(\lambda)|}$ with $i_j = \mathsf{out}(\sigma'_1 \sigma'_2 \cdots \sigma'_n)[j]$ for $1 \leq j \leq |\mathsf{out}(\lambda)|$ where for all $1 \leq t \leq n$, the substitution $\sigma'_t \in \mathcal{S}_{\mathcal{X},\mathbb{N}}$ is obtained by replacing each output letter in σ_t with the number t. Furthermore, we associate with an SST T the *transduction with origins* $R_{\mathsf{ori}}(T)$ that is the set of pairs $(u, \tilde{\mathsf{out}}(\lambda)) \in \Sigma^* \times (\Sigma \times \mathbb{N})^*$ such that T has an accepting run ρ with input u and sequence of substitutions $\kappa(\rho) = \lambda$. For example, regarding the SSTs T_1 and T_2 from Figure 2, we obtain that $(abaa, \binom{aaab}{1342}) \in R_{\mathsf{ori}}(T_1)$ and $(abaa, \binom{aaab}{4312}) \in R_{\mathsf{ori}}(T_2)$.

3 Delay measure

As mentioned in the introduction, the concept of delay has proven to be a useful tool in the understanding of finite transducers. Our goal is to introduce a robust delay measure suitable to gain a better understanding of streaming string transducers. Towards that, we informally recall the notion of delay for finite transducers: The delay between two finite transducer computations that produce the same output in the end, is a measure for how much ahead one output is compared to the other during the computation. In other words, the difference between the length of the so-far produced output is measured.

A key difference between finite transducers and SSTs is that the former build their outputs from left to right while SSTs do not have this restriction. To design a notion of delay taking this into account, we use origin information to define a notion of *weight difference*. These notions are illustrated in Example 2.

Weight difference. Given a word with origins $\tilde{u} := u \otimes o \in (\Sigma \times \mathbb{N})^*$, a time $t \in \mathbb{N}$, and $j \in \mathbb{N}$, we define the *positional weight* $\mathsf{weight}_{j,t}(\tilde{u}) \in \mathbb{N}$ as the number of positions of u up to j whose origin is no later than t, i.e.,

$$\mathsf{weight}_{j,t}(\tilde{u}) = \big|\{i \in \{1, 2, \ldots, \min(j, |u|)\} \mid o[i] \leq t\}\big|.$$

Note that $\mathsf{weight}_{j,t}(\tilde{u}) = \mathsf{weight}_{|u|,t}(\tilde{u})$ for all $j \geq |u|$. Given a second word with origins $\tilde{v} \in (\Sigma \times \mathbb{N})^*$, and $j_1, j_2 \in \mathbb{N}$, we define the *weight difference* as

$$\mathsf{diff}_{j_1,j_2,t}(\tilde{u},\tilde{v}) = \left|\mathsf{weight}_{j_1,t}(\tilde{u}) - \mathsf{weight}_{j_2,t}(\tilde{v})\right| \text{ and } \mathsf{diff}_{j,t}(\tilde{u},\tilde{v}) = \mathsf{diff}_{j,j,t}(\tilde{u},\tilde{v}).$$

Furthermore, we define the *maximal weight difference* as

$$\mathsf{max\text{-}diff}_{j_1,j_2}(\tilde{u},\tilde{v}) = \max_{t\in\mathbb{N}} \left(\mathsf{diff}_{j_1,j_2,t}(\tilde{u},\tilde{v})\right) \text{ and } \mathsf{max\text{-}diff}_{j}(\tilde{u},\tilde{v}) = \max_{t\in\mathbb{N}} \left(\mathsf{diff}_{j,t}(\tilde{u},\tilde{v})\right).$$

We remark that the value of $\mathsf{max\text{-}diff}$ is bounded even though t takes infinitely many values. Also, we note that $\mathsf{max\text{-}diff}_0(\tilde{u},\tilde{v}) = 0$. We illustrate these notions.

▶ **Example 2.** We base our example on the SSTs T_1, T_2 from Figure 2. On input *abaaa* both SSTs produce output *aaaab*. The associated origins differ, we have $\tilde{u} = \binom{aaaab}{13452}$ and $\tilde{v} = \binom{aaaab}{54312}$. We obtain $\mathsf{diff}_{2,0}(\tilde{u},\tilde{v}) = 0$, $\mathsf{diff}_{2,1}(\tilde{u},\tilde{v}) = 1$, $\mathsf{diff}_{2,2}(\tilde{u},\tilde{v}) = 1$, $\mathsf{diff}_{2,3}(\tilde{u},\tilde{v}) = 2$, $\mathsf{diff}_{2,4}(\tilde{u},\tilde{v}) = 1$, $\mathsf{diff}_{2,5}(\tilde{u},\tilde{v}) = 0$, and $\mathsf{max\text{-}diff}_2(\tilde{u},\tilde{v}) = 2$. More generally, on input aba^i the output is $a^{i+1}b$ and with origins we have $\tilde{u}_i = \left(\begin{smallmatrix} a & a & a & \cdots & a & a & b \\ 1 & 3 & 4 & \cdots & i{+}1 & i{+}2 & 2 \end{smallmatrix}\right)$ and $\tilde{v}_i = \left(\begin{smallmatrix} a & a & \cdots & a & a & a & b \\ i{+}2 & i{+}1 & \cdots & 4 & 3 & 1 & 2 \end{smallmatrix}\right)$ for all $i > 0$. Moreover, $\mathsf{max\text{-}diff}_{(i+1)/2}(\tilde{u}_i,\tilde{v}_i) = (i+1)/2$ for all odd i.

The first idea of a delay notion for SSTs similar to finite transducers is to consider the maximal weight difference that can occur. In Example 2, since T_1 builds the a-output block from left to right and T_2 from right to left, their maximal weight difference is unbounded even though $T_1 \equiv T_2$. Hence, this first idea of a delay notion violates the completeness requirement. To avoid this problem, we would like our notion of delay to reflect that, for a periodic block, it is not important if a repetition of the period is appended or prepended: the result is the same. Therefore, we only measure the difference at the end of periodic blocks and not inside of them. To this end, we introduce a new notion.

Factors, cuts. The *primitive root* of a word $u \in \Sigma^*$, denoted $\mathsf{root}(u)$, is the shortest word w such that $u = w^k$ for some positive integer k. We call a word $u \in \Sigma^*$ *primitive* if $\mathsf{root}(u) = u$.

Let $u \in \Sigma^*$ and $\ell > 0$. We cut u into factors such that each factor has a primitive root of length at most ℓ. The factors are chosen inductively from left to right in a way to maximize the size of each factor as follows. The first factor u_1 of u is the longest prefix of u such that $|\mathsf{root}(u_1)| \leq \ell$, the ith factor u_i of u is the longest prefix of u' such that $|\mathsf{root}(u_i)| \leq \ell$ where $u = u_1 \cdots u_{i-1}u'$. We refer to this unique *ℓ-factorization* as $\mathsf{factors}_\ell(u)$. Moreover, we denote by $\mathsf{cut}_\ell(u)$ the set that contains the end positions of the factors referred to as *ℓ-cuts*. For example, consider $u = aaababcbabaaaaa$ and $\ell = 2$, then the unique ℓ-factorization is $aaa|ba|bc|baba|aaaa|$ and its ℓ-cuts are $\{3, 5, 7, 11, 15\}$.

Delay. We define delay for a word and two origin annotations (of said word) by considering the maximal weight difference *at the cut positions* which are obtained via the factorization into periodic words as introduced above. Formally, given a word $w \in \Sigma^*$ and two annotated versions $\tilde{u}, \tilde{v}$ with origin, i.e., $\tilde{u} := w \otimes o_1, \tilde{v} := w \otimes o_2 \in (\Sigma \times \mathbb{N})^*$, we define the *$\ell$-delay* $\mathsf{delay}_\ell(\tilde{u},\tilde{v})$ (delay for short) as

$$\mathsf{delay}_\ell(\tilde{u},\tilde{v}) = \max_{j\in\mathsf{cut}_\ell(w)} \mathsf{max\text{-}diff}_j(\tilde{u},\tilde{v}).$$

Going back to Example 2, we have $\mathsf{delay}_1(\tilde{u}_i,\tilde{v}_i) = 0$, because the 1-factorization of $a^{i+1}b$ is $a^{i+1}|b|$ and we have $\mathsf{max\text{-}diff}_{i+1}(\tilde{u}_i,\tilde{v}_i) = \mathsf{max\text{-}diff}_{i+2}(\tilde{u}_i,\tilde{v}_i) = 0$.

We apply the delay notion to transductions with origin. Intuitively, two such transductions are "close" if for every pair $(u, w \otimes o_1)$ (from one transduction) there is some pair $(u, w \otimes o_2)$ (from the other transduction) such that the delay between these outputs with origins is small.

Let R_1, R_2 denote transductions with origin. We say that R_1 is (k,ℓ)-*included* in R_2 (written $R_1 \subseteq_{k,\ell} R_2$) if the following holds: for all $(u, w \otimes o_1) \in R_1$, there exists $(u, w \otimes o_2) \in R_2$ such that $\mathsf{delay}_\ell(w \otimes o_1, w \otimes o_2) \leq k$. We say that R_1 is (k,ℓ)-*equivalent* to R_2 (written $R_1 \equiv_{k,\ell} R_2$) if $R_1 \subseteq_{k,\ell} R_2$ and conversely $R_2 \subseteq_{k,\ell} R_1$. We are ready to state our first main result which illustrates the generality of our delay notion.

▶ **Theorem 3** (Completeness). *Given two SSTs T_1 and T_2, there exist computable integers k, ℓ such that $T_1 \equiv T_2$ iff $R_{ori}(T_1) \equiv_{k,\ell} R_{ori}(T_2)$.*

Section 4.1 is devoted to the proof of Theorem 3 and Section 5 illustrates some consequences of Theorem 3. As the delay between streaming string transducers only depends on their induced transductions with origin, we are able to state a more general result. Corollary 4 uses the fact that deterministic streaming string transducers, deterministic two-way transducers and MSO transducers are equally expressive – they characterize the so-called *regular functions* – and every regular function given in one formalism can be translated into every other one without changing its induced transduction with origins [16, 22, 4].

▶ **Corollary 4.** *Given deterministic two-way transducers resp. MSO transducers T_1 and T_2. Let R_1 and R_2 denote their induced transductions with origin. There exist computable integers k, ℓ such that $T_1 \equiv T_2$ iff $R_1 \equiv_{k,\ell} R_2$.*

We focus on the second main aspect of our delay measure, namely, regularity. Meaning that for every $k, \ell \in \mathbb{N}$, we would like to construct a finite automaton that accepts (suitable representations of) pairs $(w \otimes o_1, w \otimes o_2)$ with $\mathsf{delay}_\ell(w \otimes o_1, w \otimes o_2) \leq k$. As finite automata enjoy good closure properties, this yields a useful tool to solve for instance decision problems up to fixed delay, cf. Section 5. As mentioned in the paragraph before Corollary 4, our delay measure is applicable to transductions with origins and is complete for several transducer models. However, we need to pick some way to represent such transductions to show regularity. Hence, we prove our second main result for streaming string transducers.

▶ **Theorem 5** (Regularity). *Let $\mathcal{S} \subseteq \mathcal{S}_{\mathcal{X},\Sigma}$ be a* finite *subset of $\mathcal{S}_{\mathcal{X},\Sigma}$, let $k \geq 0$ and $\ell > 0$. The following set is a regular language:*

$$\mathbb{D}_{k,\ell,\mathcal{S}} = \{\lambda \otimes \mu \in (\mathcal{S} \times \mathcal{S})^* \mid \mathit{delay}_\ell(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k \text{ and } |\lambda| = |\mu|\}.$$

Note that $\lambda \otimes \mu \in \mathbb{D}_{k,\ell,\mathcal{S}}$ implies $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$ because delay_ℓ is defined only for such substitution sequences. We write $\mathbb{D}_{k,\ell}$ instead of $\mathbb{D}_{k,\ell,\mathcal{S}}$ when $\mathcal{S}$ is clear from the context. We prove Theorem 5 in Section 4.2 and show some applications of this result in Section 5.

4 Completeness and regularity

4.1 Completeness of the delay notion

We now prove the completeness result for SSTs, as stated in Theorem 3. To this end, we show that whenever the delay between two sequences of substitutions is sufficiently large we can pump well-chosen factors to obtain two sequences of substitutions producing outputs that are distinct. Formally, given a sequence of substitutions $\lambda \in \mathcal{S}^*$ and $1 \leq s < t < |\lambda|$, we denote by $\mathsf{pump}_{(s,t]}(\lambda)$ the sequence of substitutions $\lambda[1,t]\lambda(s,t]\lambda(t,|\lambda|)$ obtained by pumping the interval $(s,t]$, and we prove the following:

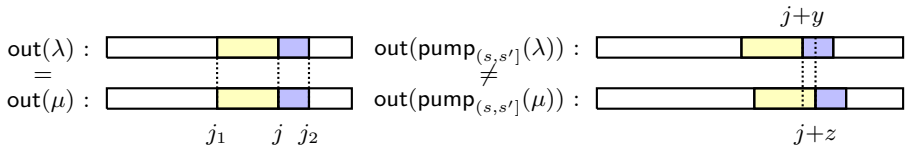


Figure 3 Illustration of the main idea used in the proof of Lemma 6.

▶ **Lemma 6.** *Let $\mathcal{S}$ be a finite set of substitutions. There exist computable integers $k, \ell \in \mathbb{N}$ such that for every integer $C \in \mathbb{N}$ and every pair $\lambda, \mu \in \mathcal{S}^*$ that satisfy $|\lambda| = |\mu|$, $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$, and $\mathsf{delay}_{C\ell}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > C^2k$, there exist $0 \leq t_1 < t_2 < \ldots < t_C < |\lambda|$ satisfying*

$$\mathsf{out}(\mathsf{pump}_{(t_i,t_j]}(\lambda)) \neq \mathsf{out}(\mathsf{pump}_{(t_i,t_j]}(\mu)) \text{ for every } 1 \leq i < j \leq C.$$

Moreover we can choose k and ℓ exponential with respect to the number of variables of $\mathcal{S}$. Before delving into the proof of Lemma 6, we argue that Theorem 3 follows as a corollary.

Proof of Theorem 3. Let T_1 and T_2 be two SSTs with set of states Q_1 and Q_2. By symmetry, it is sufficient to show that $R(T_1) \subseteq R(T_2)$ iff $R_{\mathsf{ori}}(T_1) \subseteq_{C^2k, C\ell} R_{\mathsf{ori}}(T_2)$, where $C = |Q_1| \cdot |Q_2| + 1$, and k, ℓ are as in the statement of Lemma 6 with respect to the union $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ of the substitutions used by T_1 and T_2. The right to left direction of the 'iff' is immediate, as inclusion up to bounded delay is stronger than inclusion.

We now apply Lemma 6 to prove the converse direction. Suppose that $R_{\mathsf{ori}}(T_1) \not\subseteq_{C^2k, C\ell} R_{\mathsf{ori}}(T_2)$. Then there exists a pair with origins $(u, w \otimes o_1) \in R_{\mathsf{ori}}(T_1)$ such that all the pairs with origin $(u, w \otimes o_2) \in R_{\mathsf{ori}}(T_2)$ with matching input and output satisfy $\mathsf{delay}_{C\ell}(w \otimes o_1, w \otimes o_2) > C^2k$. Two possible cases arise: either there is no pair of the form $(u, w \otimes o_2)$ in $R_{\mathsf{ori}}(T_2)$, or there exists such a pair $(u, w \otimes o_2) \in R_{\mathsf{ori}}(T_2)$, and it satisfies $\mathsf{delay}_{C\ell}(w \otimes o_1, w \otimes o_2) > C^2k$. In the former case, we immediately get that $R(T_1) \not\subseteq R(T_2)$, since (u, w) is in $R(T_1)$ but not in $R(T_2)$. In the latter case, we get that there exists a run ρ_1 of T_1 and a run ρ_2 of T_2 over the same input u that both produce the same output w, but with very different origins functions: $\mathsf{delay}_{C\ell}(\tilde{\mathsf{out}}(\kappa_1(\rho_1)), \tilde{\mathsf{out}}(\kappa_2(\rho_2))) > C^2k$. Then Lemma 6 implies that we can find C intermediate points in $\kappa_1(\rho_1)$ and $\kappa_2(\rho_2)$ such that iterating the segment between any two points yields sequences of substitutions producing distinct outputs. As $C = |Q_1| \cdot |Q_2| + 1$, two of these points mark a loop in both ρ_1 and ρ_2, and pumping these loops creates runs of T_1 and T_2 with the same input but different outputs. Since T_2 is deterministic, and therefore cannot map an input word to two distinct output words, this implies that $R(T_1)$ is not included in $R(T_2)$. ◀

Proof overview of Lemma 6. We consider two substitution sequences $\lambda, \mu \in \mathcal{S}^*$ that have the same length, produce the same output, and satisfy $\mathsf{delay}_{C\ell}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > C^2k$ for some $C, k, \ell \in \mathbb{N}$. This implies the existence of a cut position $j \in \mathsf{cut}_{C\ell}(\mathsf{out}(\lambda))$ and a point $t \in [0, |\lambda|)$ for which the weight difference $\mathsf{diff}_{j,t}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is equal to C^2k.

Our proof is based on the following fact, illustrated by Figure 3: If we choose k such that $\mathsf{diff}_{j,t}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is sufficiently large, there are intervals $(s, s'] \subset [1, |\lambda|)$ that, once pumped, add distinct amount of output before position j, thus creating a misalignment between two copies of the letter $\mathsf{out}(\lambda)[j]$ in the output words generated by $\mathsf{pump}_{(s,s']}(\lambda)$ and $\mathsf{pump}_{(s,s']}(\mu)$. Moreover, we show that carefully identifying patterns occurring along the two substitution sequences also allows us to ensure that some neighborhood $\mathsf{out}(\lambda)[j_1, j_2] = \mathsf{out}(\mu)[j_1, j_2]$ of the position j is preserved in both $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda))$ and $\mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$,

in a way that these two copies overlap, but not perfectly (this is the most complex part of the proof). At this point, we use the fact that j is a cut position of $\mathsf{out}(\lambda)$: since j marks the position where the period changes, and this position is not aligned properly in $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda))$ and $\mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$, we can derive the existence of a mismatch, proving that $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda)) \neq \mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$.

Then, all that remains to do is to combine a few counting arguments to show how to choose k and ℓ sufficiently large with respect to the parameters of $\mathcal{S}$ (and, crucially, independently of C) so that the fact that $\mathsf{diff}_{j,t}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > C^2k$ implies the existence of C consecutive intervals $(s, s'] \subset [1, |\lambda|)$, which, as described earlier, satisfy $\mathsf{out}(\mathsf{pump}_{(s,s']}(\lambda)) \neq \mathsf{out}(\mathsf{pump}_{(s,s']}(\mu))$, which concludes the proof of Lemma 6. ◀

4.2 Regularity of the delay notion

Our goal is to prove that the delay notion is regular, as stated in Theorem 5. All over this section, k and ℓ are non-negative integers, and $\mathcal{S}$ is a finite set of substitutions over a finite set of variables $\mathcal{X}$ and an alphabet Σ. Lemma 7 characterizes the pairs of substitution sequences whose outputs end with a unique endmarker symbol $\dashv$ and that are *not* in $\mathbb{D}_{k,\ell,\mathcal{S}}$, using properties which are independently shown to be regular. We first start with the characterization, then give an overview on how to show its regularity.

A characterization. Note that a pair (λ, μ) of two substitution sequences of the same length is not in $\mathbb{D}_{k,\ell,\mathcal{S}}$ if either $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$ or there is a cut witnessing a delay greater than k. Unfortunately, the first condition $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$ is not regular. So in order to characterize the complement of $\mathbb{D}_{k,\ell,\mathcal{S}}$ by regular properties, we somehow have to mix conditions on differences in the output and on positions witnessing a big delay. For the corresponding formal statement in Lemma 7 we need the following definition.

Given a sequence of substitutions λ and $i \geq 0$, let $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ be the smallest output position j such that $j > i$ and $j \in \mathsf{cut}_\ell(\mathsf{out}(\lambda))$, if it exists. Note that, as the last output position is a cut, such a position j always exists unless i is the last output position. Formally, $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ denotes the set $\min(\mathsf{cut}_\ell(\mathsf{out}(\lambda)) \cap \{j \mid j > i\})$. The result is either a singleton or the empty set. In the former case, we write $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda)) = j$ instead of $\mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda)) = \{j\}$.

▶ **Lemma 7.** *Let $\dashv \in \Sigma$ and λ, $\mu \in \mathcal{S}^*$ be sequences with $\mathsf{out}(\lambda), \mathsf{out}(\mu) \in (\Sigma \setminus \{\dashv\})^*\dashv$ and $|\lambda| = |\mu|$. Then $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$ iff there exists $i \in (\mathsf{cut}_\ell(\mathsf{out}(\lambda)) \cap \mathsf{cut}_\ell(\mathsf{out}(\mu))) \cup \{0\}$ such that $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$, and one of the following holds:*

1. *Both $j_1 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ and $j_2 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\mu))$ exist, and either $j_1 \neq j_2$ or $\mathsf{max\text{-}diff}_{j_1,j_2}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$;*
2. *$\mathsf{out}(\lambda)[i + b] \neq \mathsf{out}(\mu)[i + b]$ for some $b \in [0, \ell^2]$.*

Proof sketch. Assume that λ, μ satisfy Item 1 or 2 from the above statement. If $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$, then clearly $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$. So assume that $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$. Then $\mathsf{cut}_\ell(\mathsf{out}(\lambda)) = \mathsf{cut}_\ell(\mathsf{out}(\mu))$. Hence Item 1 is satisfied with $j_1 = j_2 =: j$ and $\mathsf{max\text{-}diff}_j(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$. Since j is a cut, we obtain that $\mathsf{delay}_\ell(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$ and thus $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$.

Conversely, let $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$. First assume $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$. Since $\lambda \otimes \mu \notin \mathbb{D}_{k,\ell,\mathcal{S}}$, there exists some $j \in \mathsf{cut}_\ell(\mathsf{out}(\lambda)) \cap \mathsf{cut}_\ell(\mathsf{out}(\mu))$ such that $\mathsf{max\text{-}diff}_j(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$, and we can satisfy Item 1. Second, we assume $\mathsf{out}(\lambda) \neq \mathsf{out}(\mu)$. Let m be the position of the earliest mismatch (that is, $\mathsf{out}(\lambda)[m] \neq \mathsf{out}(\mu)[m]$), and i be the nearest common cut to the left of the mismatch. If $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$, we have a common cut with a too large difference before a mismatch occurs. We can treat this situation as if $\mathsf{out}(\lambda) = \mathsf{out}(\mu)$ and satisfy Item 1

as before. If $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$, we show that either the mismatch is close to i (that is, $m \leq i + \ell^2$) and Item 2 is satisfied, or the mismatch causes $j_1 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ and $j_2 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\mu))$ to be different and Item 1 is satisfied for $j_1 \neq j_2$. ◀

Proof overview for the regularity of the delay notion. Let us denote by $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ the set of words of the form $\lambda \otimes \mu$ such that $\lambda, \mu \in \mathcal{S}^*$ satisfy the properties of the characterization given in Lemma 7. The main technical part is to show that $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ is regular. Then regularity of $\mathbb{D}_{k,\ell,\mathcal{S}}$ follows by complementation and end-marker removal, which preserve regularity.

First, note that the definition of $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ is existential in nature: it asks for the existence of positions in $\mathsf{out}(\lambda)$ and $\mathsf{out}(\mu)$ satisfying some properties. A classical way of dealing with positions quantified existentially in automata theory is to mark some positions in the input by using an extended alphabet, construct an automaton over the extended alphabet, and then project this automaton over the original alphabet. Here, the positions needed in $\mathbb{C}^{\dashv}_{k,\ell,\mathcal{S}}$ are positions of $\mathsf{out}(\lambda)$ and $\mathsf{out}(\mu)$, while the automata we want to construct read λ and μ as input. So, instead of marking input positions, we rather mark positions in right-hand sides of updates occurring in the substitutions of λ and μ. Let us make this more precise. First, for $n \geq 0$, words u over the alphabet Σ are extended into *n-marked words*, i.e., words over the alphabet $\Sigma \times 2^{\{1,\dots,n\}}$, such that the additional information in $\{1, \dots, n\}$ precisely corresponds to an n-tuple of positions $\overline{x}$ of u (position x_i is marked with label i for all $i \in \{1, \dots, n\}$, and we consider sets because the same position can correspond to different components of $\overline{x}$). By extension, we also define an operation $\triangleright$ which marks any substitution sequence $\lambda \in \mathcal{S}^*_{\mathcal{X},\Sigma}$ by a tuple $\overline{x}$ of positions of $\mathsf{out}(\lambda)$, resulting in a substitution sequence $(\lambda \triangleright \overline{x}) \in \mathcal{S}^*_{\mathcal{X},\Sigma \times 2^{\{1,\dots,n\}}}$ such that $\mathsf{out}(\lambda \triangleright \overline{x}) = \mathsf{out}(\lambda) \triangleright \overline{x}$.

We show that the set of substitution sequences $\lambda \triangleright i$ satisfying $i \in \mathsf{cut}_\ell(\mathsf{out}(\lambda))$ is regular, and similarly for $\mathsf{next\text{-}cut}_\ell$. To do so, we prove that the set $\{u \triangleright i \mid i \in \mathsf{cut}_\ell(u)\}$ is regular (and similarly for $\mathsf{next\text{-}cut}_\ell$) and then transfer this result to marked substitution sequences, as regular languages are preserved under inverse of SSTs.

Then, we show regularity results for predicates of the form $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$ and $\mathsf{max\text{-}diff}_{j_1,j_2}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) > k$. In the end, all parts of the property of the characterization of Lemma 7 are shown to be regular, so that the whole property can be checked by a synchronized product of automata. Perhaps the most interesting part is how to show that the predicate $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$ is regular. More precisely, it is shown that the set of $(\lambda \triangleright i_1) \otimes (\mu \triangleright i_2)$ such that $i_1 = i_2 = i$ and $\mathsf{max\text{-}diff}_i(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is smaller than k (which is a given constant), is regular. Let us intuitively explain why. In general, checking whether two marked positions i_1 (in $\mathsf{out}(\lambda)$) and i_2 (in $\mathsf{out}(\mu)$) are equal cannot be done in a regular way (recall that the automaton reads $\lambda \otimes \mu$ and not their outputs). However, if additionally, one has to check that $\mathsf{max\text{-}diff}_{i_1,i_2}(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu))$ is smaller than k, it is actually regular. To do so, a finite automaton needs to monitor the difference in the outputs produced in λ and μ before positions i_1 and i_2 resp., and check that it is bounded by k (otherwise it rejects). The difference must eventually reach 0 when the whole inputs λ and μ have been read, to ensure $i_1 = i_2$.

We also prove that once a position i in $\mathsf{out}(\lambda)$ is marked, then the next cut $j_1 = \mathsf{next\text{-}cut}_\ell(i, \mathsf{out}(\lambda))$ can be identified in a regular way by an automaton. Similarly, given a constant d, the output position $i + d$ can also be identified in a regular way. This allows us to check the properties of Item 1 and Item 2 respectively of Lemma 7. This concludes the overview of the proof of Theorem 5. A complexity analysis yields that the set $\mathbb{D}_{k,\ell,\mathcal{S}}$ is recognizable by a DFA with a number of states doubly exponential in ℓ^3 and in $|\mathcal{X}|$, and singly exponential in k.

5 Applications of delay completeness and regularity

Decision problems up to fixed delay. Instead of comparing the transductions defined by non-deterministic SSTs for inclusion or equivalence, which are undecidable problems already for finite (variable-free) transducers, we show here that the strengthening of those problems up to fixed delay, $\subseteq_{k,l}$ and $\equiv_{k,l}$, are decidable. The key to decide inclusion and equivalence up to fixed delay is the regularity of the delay notion as stated in Theorem 5. Hence, those problems can be reduced to classical inclusion and equivalence problems of regular languages.

▶ **Theorem 8.** *Given integers k, ℓ, the (k,ℓ)-inclusion problem for non-deterministic SSTs is decidable. It is* PSPACE*-complete if k, ℓ are constants and the number of variables $|\mathcal{X}|$ is a constant. The same results hold for the (k,ℓ)-equivalence problem.*

Proof sketch. We sketch the result for the inclusion problem. For a non-deterministic SST T over input alphabet Σ, we denote by $L(T)$ its language, defined as the set of words of the form $u \otimes \tau(\rho)$, where $u \in \Sigma^*$, ρ is an accepting run of T over u, and $\tau(\rho)$ is the sequence of substitutions occurring on ρ. Let T_1 and T_2 be two non-deterministic SSTs over two finite sets of variables $\mathcal{X}_1$ and $\mathcal{X}_2$ respectively, both with output variable O. Let $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) be the finite set of substitutions occurring in T_1 (resp. T_2). Let $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ and $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$. Let $\ell, k \in \mathbb{N}$.

We let $\mathbb{D}^{in}_{k,\ell,\mathcal{S}} = \{u \otimes \lambda \otimes \mu \mid u \in \Sigma^* \wedge \lambda \otimes \mu \in \mathbb{D}_{k,\ell,\mathcal{S}}\}$. The automaton recognizing $\mathbb{D}_{k,\ell,\mathcal{S}}$ from the proof of Theorem 5 can be easily extended into an automaton which recognizes $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}$. Now, observe that T_1 is (k,ℓ)-included in T_2 iff $L(T_1) \subseteq \mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L(T_2))$, where $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L(T_2))$ denotes the set $\{u \otimes \lambda \mid \exists u \otimes \mu \in L(T_2) \colon u \otimes \lambda \otimes \mu \in \mathbb{D}^{in}_{k,\ell,\mathcal{S}}\}$. ◀

We turn to the variable minimization problem which is open for (deterministic) SSTs and undecidable for non-deterministic SSTs. We prove decidability for variable minimization up to fixed delay in both cases.

▶ **Theorem 9.** *Given integers k, ℓ, m and a non-deterministic SST T, it is decidable whether $T \equiv_{k,\ell} T'$ for some (non-deterministic resp. deterministic) SST T' that uses at most m variables.*

Proof sketch. First, we sketch the result for non-deterministic SST. Since we are looking for some non-deterministic SST T' with m variables such that $T \equiv_{k,\ell} T'$, we need to consider SSTs that produce at most $r := 2k + p$ letters (where p is the maximal number of letters produced by T in one step) per computation step in order to not violate the delay bound. The reasoning is that the difference between the output of the computations can be at most k letters, then in the next computation step, the computation that was k letters ahead may produce p letters, the other computation must recover the difference by producing at least p letters and at most $2k + p$ letters to keep the difference at most k. Thus, let $\mathcal{S} = \mathcal{S}_T \cup \mathcal{S}_{r,m}$ and $\mathcal{X} = \mathcal{X}_T \cup \mathcal{X}_m$, where $\mathcal{S}_T$ (resp. $\mathcal{X}_T$) are the substitutions (resp. variables) occurring in T, $\mathcal{X}_m = \{X_1, \cdots, X_m\}$, and $\mathcal{S}_{r,m}$ are substitutions over $\mathcal{X}_m$ producing at most r letters.

As as in the proof sketch of Theorem 8, $L(T)$ is the set of words of the form $u \otimes \tau(\rho)$, where $u \in \Sigma^*$, ρ is an accepting run of T over u, and $\tau(\rho)$ is the sequence of substitutions occurring on ρ. Furthermore, as in the proof sketch above, we let $\mathbb{D}^{in}_{k,\ell,\mathcal{S}} = \{u \otimes \lambda \otimes \mu \mid u \in \Sigma^* \wedge \lambda \otimes \mu \in \mathbb{D}_{k,\ell,\mathcal{S}}\}$. Now, observe that there exists some non-deterministic SST T' with m variables such that $T \equiv_{k,\ell} T'$ iff $L(T) \subseteq \mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L)$ where L is the set of all words $u \otimes \mu$ such that $u \in \Sigma^*$ and $\mu \in \mathcal{S}^*_{r,m}$. As in the proof sketch above, $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}(L)$ denotes the set $\{u \otimes \lambda \mid \exists u \otimes \mu \in L \colon u \otimes \lambda \otimes \mu \in \mathbb{D}^{in}_{k,\ell,\mathcal{S}}\}$.

If the goal is to have a deterministic SST, we need to adapt the above procedure. Let M be the projection of $\mathbb{D}^{in}_{k,\ell,\mathcal{S}}$ onto its first and third component. Hence, M contains all $u \otimes \mu$ such that there is some $u \otimes \lambda \in L(T)$ with $\mathsf{delay}_\ell(\tilde{\mathsf{out}}(\lambda), \tilde{\mathsf{out}}(\mu)) \leq k$. We reduce the problem to a safety game played on a DFA for M. In alternation, one player provides an input letter, the other chooses a matching transition in the DFA. If the input player has provided a sequence $u \in \mathrm{dom}(R(T))$ then the play must be in an accepting state of the DFA, otherwise the output player has lost. We show that the output player has a winning strategy iff there exists an equivalent deterministic SST with at most m variables. ◀

Comparison with delay for finite transducers. We compare our notion of delay for streaming string transducers with the previously existing delay notion for finite transducers [24]. A *rational* SST is a non-deterministic SST with only one variable O, and updates all of the form $O := Ou$. In other words, a rational SST T is simply a finite transducer. Applying our delay notion to rational SSTs yields that if T_1 is (k,ℓ)-included in T_2 for two rational SSTs T_1, T_2, then T_1 is k-included in T_2 for the definition according to [24] and vice versa. The k-inclusion problem for finite transducers is PSPACE-complete for fixed k [24]. Hence, the complexity obtained in Theorem 8 matches this bound. on conceptual differences between those notions.

Consequences of completeness. We now turn to some consequences of our completeness result (Theorem 3). Inclusion for (deterministic) SSTs is known to be decidable [2]. It is undecidable for non-deterministic SSTs, but (k,ℓ)-inclusion is decidable (Theorem 8). Although Theorem 3 provides a new decision procedure for the inclusion problem for SSTs its value lies in showing that our notion of delay is a sensible approach to gain a better understanding of streaming string transducers. The following two corollaries are easy consequences of Theorem 3.

▶ **Corollary 10.** *Given an SST T and an integer m, there exist integers k, ℓ such that there exists an SST T' with m variables such that $T \equiv T'$ iff there exists an SST T'' with m variables such that $T \equiv_{k,\ell} T''$.*

Note that the above corollary does not imply that k and ℓ are *computable*. This would entail a solution for the variable minimization problem for SSTs which is open (and decidable for concatenation-free SSTs [6]). The next result is about rational functions, that is, functions recognizable by finite transducers.

▶ **Corollary 11.** *Given an SST T, there exist integers k, ℓ such that there exists a rational SST T' such that $T \equiv T'$ iff there exists a rational SST T'' such that $T \equiv_{k,\ell} T''$.*

It was shown that it decidable whether a deterministic two-way transducer (which is effectively equivalent to an SST) recognizes a rational function [23]. The decision procedure is effective, i.e., a finite transducer is constructed if possible. A procedure with improved complexity was given that yields a finite transducer of doubly exponential size in [7]. While computability is not implied by Corollary 11, note that one could compute k, ℓ satisfying the statement of Corollary 11 using Theorem 3 from an equivalent rational SST (if it exists) that has been obtained using the decision procedure from [7].

Other applications. We mention other potential applications of our delay notion that ought to be investigated. For instance, a *decomposition theorem* for SST relations is still only conjectured: Can every finite-valued SST relation be decomposed into a finite union of SST

functions? In other settings where the corresponding statement holds (finite transducers [33], single-variable SST [25]), the main ingredients of the proof is the regularity and completeness of the appropriate notion of delay. So, having a good delay notion seems necessary to obtain such a result, but solving the decomposition theorem for SSTs does *not* seem to be a low-hanging fruit of our present study.

The notion of delay might also help to solve the *variable minimization* problem: Can we determine the minimal number of variables needed to define a given SST function? Corollary 10 makes some progress towards a positive answer, yet it remains to prove that the integers k and ℓ of the statement are computable, which is likely a complex problem. Another interesting research direction is to study how our notion of delay fares beyond SST. For *copyful* SST (where the copyless restriction of the substitutions is dropped), our notion of delay can be defined in the same manner, but its properties are unclear: our proofs of regularity and completeness both crucially rely on the copyless assumption.

References

1 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.1.

2 Rajeev Alur and Pavol Cerný. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL 2011*, pages 599–610. ACM, 2011.

3 Rajeev Alur, Loris D'Antoni, and Mukund Raghothaman. Drex: A declarative language for efficiently evaluating regular string transformations. In Sriram K. Rajamani and David Walker, editors, *POPL 2015*, pages 125–137. ACM, 2015. doi:10.1145/2676726.2676981.

4 Rajeev Alur and Jyotirmoy V. Deshmukh. Nondeterministic streaming string transducers. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *ICALP 2011*, volume 6756 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011. doi:10.1007/978-3-642-22012-8_1.

5 Rajeev Alur, Adam Freilich, and Mukund Raghothaman. Regular combinators for string transformations. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14–18, 2014*, pages 9:1–9:10. ACM, 2014. doi:10.1145/2603088.2603151.

6 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Minimizing resources of sweeping and streaming string transducers. In *ICALP 2016*, volume 55 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

7 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Untwisting two-way transducers in elementary time. In *LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

8 Nicolas Baudru and Pierre-Alain Reynier. From two-way transducers to regular function expressions. *Int. J. Found. Comput. Sci.*, 31(6):843–873, 2020. doi:10.1142/S0129054120410087.

9 Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theor. Comput. Sci.*, 292(1):45–63, 2003. doi:10.1016/S0304-3975(01)00214-6.

10 Mikolaj Bojanczyk. Transducers with origin information. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer, 2014. doi:10.1007/978-3-662-43951-7_3.

11 Mikolaj Bojanczyk. Polyregular functions. *CoRR*, abs/1810.08760, 2018. arXiv:1810.08760.

12 Sougata Bose. *On decision problems on word transducers with origin semantics. (Sur les problèmes de décision concernant les transducteurs de mots avec la sémantique d'origine)*. PhD thesis, University of Bordeaux, France, 2021. URL: https://tel.archives-ouvertes.fr/tel-03216029.

13 Sougata Bose, Anca Muscholl, Vincent Penelle, and Gabriele Puppis. Origin-equivalence of two-way word transducers is in PSPACE. *CoRR*, abs/1807.08053, 2018. arXiv:1807.08053.

14 Christian Choffrut. Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977. doi:10.1016/0304-3975(77)90049-4.

15 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003. doi:10.1016/S0304-3975(01)00219-5.

16 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, volume 138 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2012. URL: http://www.cambridge.org/fr/knowledge/isbn/item5758776/?site_locale=fr_FR.

17 Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for word transductions with synthesis. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 295–304. ACM, 2018.

18 Luc Dartois, Paul Gastin, R. Govind, and Shankara Narayanan Krishna. Efficient construction of reversible transducers from regular transducer expressions. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2–5, 2022*, pages 50:1–50:13. ACM, 2022.

19 Luc Dartois, Paul Gastin, and Shankara Narayanan Krishna. Sd-regular transducer expressions for aperiodic transformations. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 – July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470738.

20 Laure Daviaud, Ismaël Jecker, Pierre-Alain Reynier, and Didier Villevalois. Degree of sequentiality of weighted automata. In Javier Esparza and Andrzej S. Murawski, editors, *FOSSACS 2017*, volume 10203 of *Lecture Notes in Computer Science*, pages 215–230, 2017. doi:10.1007/978-3-662-54458-7_13.

21 Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965. doi:10.1147/rd.91.0047.

22 Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001. doi:10.1145/371316.371512.

23 Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. From two-way to one-way finite state transducers. In *LICS 2013*, pages 468–477. IEEE Computer Society, 2013.

24 Emmanuel Filiot, Ismaël Jecker, Christof Löding, and Sarah Winter. On equivalence and uniformisation problems for finite transducers. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP 2016*, volume 55 of *LIPIcs*, pages 125:1–125:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

25 Paul Gallot, Anca Muscholl, Gabriele Puppis, and Sylvain Salvati. On the decomposition of finite-valued streaming string transducers. In Heribert Vollmer and Brigitte Vallée, editors, *STACS 2017*, volume 66 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.34.

26 Eitan M. Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM J. Comput.*, 11(3):448–452, 1982. doi:10.1137/0211035.

27 Ismaël Jecker and Emmanuel Filiot. Multi-sequential word relations. In Igor Potapov, editor, *DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2015. doi:10.1007/978-3-319-21500-6_23.

28 Christophe Reutenauer and Marcel Paul Schützenberger. Minimization of rational word functions. *SIAM J. Comput.*, 20(4):669–685, 1991. doi:10.1137/0220042.

29 Jacques Sakarovitch and Rodrigo de Souza. On the decidability of bounded valuedness for transducers. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *MFCS 2008*, volume 5162 of *Lecture Notes in Computer Science*, pages 588–600. Springer, 2008. doi:10.1007/978-3-540-85238-4_48.

30 Jacques Sakarovitch and Rodrigo de Souza. On the decomposition of k-valued rational relations. In Susanne Albers and Pascal Weil, editors, *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 1 of *LIPIcs*, pages 621–632. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008. doi:10.4230/LIPIcs.STACS.2008.1324.

31 Jacques Sakarovitch and Rodrigo de Souza. Lexicographic decomposition of k-valued transducers. *Theory Comput. Syst.*, 47(3):758–785, 2010. doi:10.1007/s00224-009-9206-6.

32 Marcel Paul Schützenberger. A remark on finite transducers. *Inf. Control.*, 4(2-3):185–196, 1961. doi:10.1016/S0019-9958(61)80006-5.

33 Andreas Weber. Decomposing A k-valued transducer into k unambiguous ones. *RAIRO Theor. Informatics Appl.*, 30(5):379–413, 1996. doi:10.1051/ita/1996300503791.

New Clocks, Optimal Line Formation and Self-Replication Population Protocols

Leszek Gąsieniec[1]
University of Liverpool, UK

Paul G. Spirakis
University of Liverpool, UK

Grzegorz Stachowiak
University of Wrocław, Poland

Abstract

In this paper we consider a known variant of the standard population protocol model in which agents are allowed to be connected by edges, referred to as the *network constructor* model. During an interaction between two agents the relevant connecting edge can be formed, maintained or eliminated by the transition function. Since pairs of agents are chosen uniformly at random the status of each edge is updated every $\Theta(n^2)$ interactions in expectation which coincides with $\Theta(n)$ parallel time. This phenomenon provides a natural lower bound on the time complexity for any non-trivial network construction designed for this variant. This is in contrast with the standard population protocol model in which efficient protocols operate in $O(\text{poly}\log n)$ parallel time.

The main focus of this paper is on efficient manipulation of linear structures including formation, self-replication and distribution (including pipelining) of *complex information* in the adopted model.

- We propose and analyze a novel edge based phase clock counting parallel time $\Theta(n \log n)$ in the network constructor model, showing also that its leader based counterpart provides the same time guarantees in the standard population protocol model. Note that all currently known phase clocks can count parallel time not exceeding $O(\text{poly}\log n)$.
- We prove that any spanning line formation protocol requires $\Omega(n \log n)$ parallel time if high probability guaranty is imposed. We also show that the new clock enables an optimal $O(n \log n)$ parallel time spanning line construction, which improves dramatically on the best currently known $O(n^2)$ parallel time protocol, solving the main open problem in the considered model [24].
- We propose a new *probabilistic bubble-sort* algorithm in which random comparisons and transfers are limited to the adjacent positions in the sequence. Utilising a novel potential function reasoning we show that rather surprisingly this probabilistic sorting procedure requires $O(n^2)$ comparisons in expectation and whp, and is on par with its deterministic counterpart.
- We propose the first population protocol allowing self-replication of a *strand* of an arbitrary length k (carrying k-bit message of size independent of the state space) in parallel time $O(n(k + \log n))$. The bit pipelining mechanism and the time complexity analysis of self-replication process mimic those used in the probabilistic bubble-sort argument. The new protocol permits also simultaneous self-replication, where l copies of the strand can be created in parallel in time $O(n(k+\log n)\log l)$. We also discuss application of the strand self-replication protocol to pattern matching.

All protocols are always correct and provide time guarantees with high probability defined as $1-n^{-\eta}$, for a constant $\eta > 0$.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Population protocols, constructors, probabilistic bubble-sort, self-replication

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.33

Funding Supported by the National Science Center, Poland (NCN), grant 2020/39/B/ST6/03288 and the EPSRC grant EP/P02002X/1.

[1] Corresponding author

© Leszek Gąsieniec, Paul G. Spirakis, and Grzegorz Stachowiak;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The model of *population protocols* originates from the seminal work of Angluin et al. [4]. The model is used to study distributed processes based on pairwise interactions between anonymous agents drawn from a large population of size n. The interacting pairs of agents are chosen by the *random scheduler* and their states are amended by the predefined *transition function* governing the considered process. The state space of agents is fixed (constant size) and the size n is not known, i.e., not hard-coded in the transition function. We assume that a population protocol starts in the predefined initial configuration of agents' states representing the input, and it concludes in an output configuration reflecting on the solution to the considered problem. The sequential time complexity of a protocol refers to the number of interactions required to stabilise this protocol in one of the final configurations. In more recent work on population protocols the focus is on *parallel time* defined as the total number of pairwise interactions (sequential time) leading to the solution divided by the population size n. For example, a core dissemination tool in population protocols known as *one-way epidemic* [5] distributes simple (e.g., 0/1) messages to all agents in the population utilising $\Theta(n \log n)$ interactions or equivalently $\Theta(\log n)$ parallel time. The parallel time is meant to reflect on massive parallelism of simultaneous interactions. While this is a simplification [14], it provides a good estimation on locally observed time expressed in the number of interactions each agent was involved in throughout the computation process.

Unless stated otherwise, we assume that any protocol starts in the predefined *initial configuration* with all agents being in the same *initial state*. A population protocol *terminates with success* if the whole population stabilises eventually, i.e., it arrives at and stays indefinitely in one of the *final configurations* of states representing the desired property of the solution.

1.1 Network Constructors Model

While in the standard population protocol model the population of agents remains unstructured, in the network *constructors* model introduced in [24] and adopted in this paper during an interaction between two agents the edge connecting them can be formed, maintained or eliminated by the transition function. In this way the protocol instructs agents how to organize themselves into temporary or more definite network structures.

Note that since pairs of agents are chosen uniformly at random the status of any edge is updated on average every $\Theta(n^2)$ interactions which coincides with $\Theta(n)$ parallel time. With the exception of some relaxed expectations [12], this phenomenon provides a natural lower bound on the time complexity of non-trivial network construction processes, see [24]. On the other hand this model enables generic protocols capable of simulating space bounded Turing Machine allowing more complex computations including construction of a large class of networks [24].

Model specificity. Whenever possible we will use capital letters to denote states of the agents. In order to accommodate edge connections the transition function governs the relation between triplets of the following type:

$$P + Q + S \longrightarrow P' + Q' + S'.$$

The first two terms on both sides of the rule refer to the states P and Q of the initiator and the responder (respectively) before and P' and Q' after the interaction. The third term S before and S' after the interaction is a binary flag indicating the status of the connection between the two agents, where the edge presence is declared by 1 and by 0 the lack of it. The

states of agents are often more complex being a combination of a fixed number of attributes. Such states are represented as tuples. For such compound states we use vector representation with acute brackets $<,>$, where the individual attributes are separated by commas.

Probabilistic guarantees. Let η be a universal positive constant referring to the reliability of our protocols. We say that an event occurs with *negligible* probability if it occurs with probability at most $n^{-\eta}$, and an event occurs with high probability (whp) if it occurs with probability at least $1 - n^{-\eta}$. This estimate is of an asymptotic nature, i.e., we assume n is large enough to validate the results. Similarly, we say that an algorithm succeeds with high probability if it succeeds with probability at least $1 - n^{-\eta}$. When we refer to the probability of failure p different to $n^{-\eta}$, we say directly *with probability at least* $1 - p$. Our protocols make heavy use of Chernoff bounds and the new tail bounds for sums of geometric random variables derived in [20]. We refer to these new bounds as Chernoff-Janson bounds.

We also use notation $f(n) \sim g(n) \Leftrightarrow \frac{f(n)}{g(n)} \underset{n\to\infty}{\longrightarrow} 1$

1.2 Our results and their significance

The model of population protocols gained considerably in popularity in the last 15 years. We study here several central problems in distributed computing by focusing on the adopted variant of population protocols. These include *phase clocks*, a distributed synchronisation tool with good space, time accuracy, and probabilistic guarantees. The first study of leader based $O(1)$ space phase clocks can be found in the seminal paper by Angluin et al. in [5]. Further extensions including junta based nested clocks counting any $\Theta(\text{poly}\log n)$ parallel time were analyzed in [18]. Leaderless clocks based on power of two choices principle were used in fast majority protocols [2], and more recently constant resolution phase clocks propelled the optimal *majority* protocol [16]. In this work we propose and analyze a new matching based phase clock allowing to count $\Theta(n \log n)$ parallel time. This is the first clock confirming the conclusion of the slow leader election protocol based on direct duels between the remaining leader candidates. We also propose an edge-less variant of this clock based on the computed leader. This clock powers the first optimal $O(n \log n)$ parallel time spanning line construction, a key component of universal network construction, improving dramatically on the best currently known $O(n^2)$ parallel time protocol, and solving the main open problem from [24].

We also consider a probabilistic variant of the classical bubble-sort algorithm, in which any two consecutive positions in the sequence are chosen for comparison uniformly at random. We show that rather surprisingly this variant is on par with its deterministic counterpart as it requires $\Theta(n^2)$ random comparisons whp. While this new result is of independent algorithmic interest, together with the edge-less clock they conceptually power the strand (line-segment carrying information) self-replication protocol studied at the end of this paper.

In a wider context, *self-replication* is a property of a dynamical system which allows reproduction. Such systems are of increasing interest in biology, e.g., in studies on how life could have begun on Earth [23], in computational chemistry [25], robotics [21] and others. In this paper, a chunk of information is stored collectively in a *strand*, i.e., a line segment of agents of length k, where each agent stores one bit of information. Such strands may represent strings in pattern matching, a large value, or a code in more complex distributed process. In such cases the replication mechanism facilitates an improved accessibility to this information. We propose the first strand self-replication protocol allowing to reproduce a strand of non-fixed size k in parallel time $O(n(k + \log n))$. This protocol permits concurrent replication, where l copies of a strand can be generated in parallel time $O(n(k + \log n) \log l)$. The parallelism of this protocol is utilised in efficient pattern matching in Section 6.1.

1.3 Related work

One of the main tools used in this paper refers to the central problem of *leader election*, with the final configuration comprising a single agent in the *leader* state and all other agents in the *follower* state. The leader election problem received in recent years greater attention in the context of population protocols. In particular, the results in [10, 15] laid down the foundation for the proof that leader election cannot be solved in a sublinear time with agents utilising a fixed number of states [17]. In further work [3], Alistarh and Gelashvili studied the relevant upper bound, where they proposed a new leader election protocol stabilising in time $O(\log^3 n)$ assuming $O(\log^3 n)$ states per agent.

In a more recent work Alistarh et al. [1] considered more general trade-offs between the number of states used by the agents and the time complexity of stabilisation. In particular, the authors delivered a separation argument distinguishing between *slowly stabilising* population protocols which utilise $o(\log\log n)$ states and *rapidly stabilising* protocols relying on $O(\log n)$ states per agent. This result coincides with another fundamental result by Chatzigiannakis et al. [9] stating that population protocols utilizing $o(\log n)$ states are limited to semi-linear predicates, while the availability of $O(n)$ states (permitting unique identifiers) admits computation of more general symmetric predicates. Further developments include also a protocol which elects the leader in time $O(\log^2 n)$ w.h.p. and in expectation utilizing $O(\log^2 n)$ states [8]. The number of states was later reduced to $O(\log n)$ by Alistarh et al. in [2] and by Berenbrink et al. in [7] through the application of two types of synthetic coins.

In more recent work Gąsieniec and Stachowiak reduce memory utilisation to $O(\log\log n)$ while preserving the time complexity $O(\log^2 n)$ whp [18]. The high probability can be traded for faster leader election in the expected parallel time $O(\log n \log\log n)$, see [19]. This upper bound was recently reduced to the optimal expected time $O(\log n)$ by Berenbrink et al. in [6]. One of the main open problems in the area is to establish whether one can elect a single leader in time $o(\log^2 n)$ whp while preserving the optimal number of states $O(\log\log n)$.

2 Two phase clocks and leader election

In order to compute the unique leader and confirm its computation we execute two protocols simultaneously. Namely, the slow leader election protocol which concludes in parallel time $O(n \log n)$ whp, and the new (introduced below) *matching based phase clock* which counts parallel time $\Theta(n \log n)$ whp. The conclusion of leader election is confirmed via one-way epidemic when the final state (in this clock) is reached by any agent. This leader is utilised in the edge-less clock in nearly optimal computation of the line containing all agents, see Section 4, and in self-replication of strands of information, see Section 6.

The transition rules for governing the slow leader election and the new clocks follow.

2.1 Slow leader election

In the initial configuration all agents are in state L and the leader election protocol is driven by a single rule:

$$L + L \to L + F,$$

where L represents a leader candidate, and F stands for a *follower* (or a *free*) agent. It is well known that this leader election protocol operates in the expected parallel time $\Theta(n)$, and in parallel time $\Theta(n \log n)$ whp.

2.2 Matching based phase clock

The proposed matching based clock assumes the constructors model in which the transition function recognises whether two interacting agents are connected by an edge or not, indicated by 1 or 0, respectively. The agents begin in the predefined state $< start >$. When two agents in state $< start >$ interact they get connected and they enter the *counting stage* with their counters set to $< 0 >$. Eventually these counters reach the maximum (exit) value max. The values of the counters can go either up or down, depending on the rule used during the relevant interaction. Note that when i is the smallest counter value, then the cardinality of the subpopulation holding it can only go down. We prove that as long as there are agents with counter value below a fixed threshold, it is almost impossible for any other agent to reach the counter value max. And this is the case for the first $\Theta(n^2 \log n)$ interactions, i.e., $\Theta(n \log n)$ parallel time. Note also that the number of agents taking part in the counting process is always even as they enter and leave this process in pairs. The counting stage guarantees that the counters of all agents which enter this stage reach level max (denoted by state $< max >$) in time $\Theta(n \log n)$, see Theorem 1. And during the next interaction between the two connected agents in state $< max >$ the connection is dropped and the states are updated to $< end >$ indicating the exit from the counting stage.

The rules of the transition function used in the counting stage are as follows:

Initialisation

$$< start > + < start > \ + 0 \ \longrightarrow < 0 > + < 0 > + 1$$

Timid counting

- For all connected $i \leq j$ and $i < max$

$$< i > + < j > + 1 \ \longrightarrow < i+1 > + < i+1 > + 1$$

- For all disconnected $i < j$

$$< i > + < j > + 0 \ \longrightarrow < i > + < i+1 > + 0$$

Maximum level epidemic

$$< max > + < i > + 0 \ \longrightarrow < max > + < max > + 0$$

Conclude and disconnect

$$< max > + < max > + 1 \ \longrightarrow < end > + < end > + 0$$

$$< start > + < end > + 0 \ \longrightarrow < end > + < end > + 0 \ \#$$ takes care of the odd n case

In the next subsection we discuss the rules of an alternative phase clock in which instead of a matching the agents use virtual edges connecting them with the computed leader.

2.3 Leader based (edge-less) phase clock

We allocate separate constant memory to host the states of the leader based clock. This allows to run the two clocks simultaneously and independently. The followers in the leader based clock start with the counters set to $< 0 >$, and L refers to the leader state. Note that state $< 0 >$ is initiated for the leader based clock as soon as the agent reaches state $< max >$ or $< end >$ in the matching based clock. Below we present the timid counting rules in which matching edges are replaced by virtual edges between the leader L and all other agents.

Timid counting

- Leader interactions, for $i < max$

$$< i > + L \longrightarrow < i+1 > + L$$

- Non-leader interactions, for $i < j$

$$< i > + < j > \longrightarrow < i > + < i+1 >$$

One can show that the two clocks have the same asymptotic time performance, see Section 3 for the relevant details. Note also that the leader based clock can be used independently from any edge dependent process executed in the population simultaneously.

2.4 Periodic leader based (edge-less) clock

One can expand the functionality of the leader based clock to pace a series of consecutive rounds of a more complex process, with each round operating in parallel time $\Theta(n \log n)$. The extension is assumed to work in rounds formed of three consecutive stages 0, 1 and 2, where each stage is associated with a single execution (full turn) of the leader based clock. The conclusion of each stage is announced with the help of one-way epidemic in parallel time $O(\log n)$ whp. And when this happens all agents which received the announcement proceed to the next stage. This means that after at most $O(\log n)$ parallel time delay (caused by the epidemic) all agents will run the clock in the same stage whp. Note also that while the signal to start the next stage remains in the population throughout the whole stage, it will be wiped out whp by the signal announcing the beginning of the stage that follows. And since we have 3 stages during each round the synchronisation of agents is guaranteed whp.

3 The clocks' analysis

In this section we provide the time and the probabilistic guarantees for the two phase clocks introduced in Section 2. We first analyze the matching based clock and later extend the reasoning to the leader based (edge-less) clock. We prove the following theorem towards the end of this section.

▶ **Theorem 1.** *In either of the two clocks state <max> is reached by any agent in parallel time $\Theta(n \log n)$ whp.*

When the matching based clock is initialised, it forms a matching consisting of unmatched pairs of agents. In Lemmas 2,3 we specify how fast this is done. In Lemma 7 we prove that whp no counter in the population has value max for as long as the smallest counter value is at most $max - d - 2$. The constant d depends on η and its value can be derived from the proof of Lemma 5. Also, if T is the time elapsed before the value max is observed in the population for the first time, Lemma 8 guarantees that $T > (max - d - 2)0.4n \ln n$ whp. Using this inequality, the top value max can be derived from d and time $T = \Theta(n \log n)$ which upperbounds whp the parallel time of slow leader election process.

▶ **Lemma 2.** *All edges of the matching are formed in the expected parallel time $\Theta(n)$ and whp $O(n \log n)$.*

Proof. The probability of an interaction forming edge $i+1$ when i edges are already formed is $\frac{(n-2i)(n-2i-1)}{n(n-1)}$. Thus the number of interactions separating formation of edges i and $i+1$ has geometric distribution with the expected value $\frac{n(n-1)}{(n-2i)(n-2i-1)}$. Thus the expected number of interactions to form all edges is $\sum_{i=1}^{n/2-1} \frac{n(n-1)}{(n-2i)(n-2i-1)}$, which is $\Theta(n^2)$.

A sufficient condition to form all the edges is that all possible $\binom{n}{2}$ pairs of agents are generated by the random scheduler. The probability of not choosing a fixed pair in first $cn^2 \log n$ interactions is $\left(1-1/\binom{n}{2}\right)^{cn^2 \log n}$, which is negligible for c big enough. Thus all edges are formed after parallel time $O(n \log n)$ whp. ◀

The following lemma refers to early interactions in the matching based clock.

▶ **Lemma 3.** *After parallel time* 0.51 *at least* $\frac{n}{2}$ *agents belong to already formed edges whp.*

Proof. Assume that so far exactly i edges are formed. The probability that during an interaction edge $i+1$ is formed is $\frac{(n-2i)(n-2i-1)}{n(n-1)}$. So the expected number of interactions T_i of forming edge $i+1$ is $\frac{n(n-1)}{(n-2i)(n-2i-1)}$. And in turn the expected number of interactions T of forming first $n/4$ edges satisfies

$$T = T_0 + \cdots + T_{n/4} = \sum_{i=0}^{n/4} \frac{n(n-1)}{(n-2i)(n-2i-1)} \sim n \int_0^{1/4} \frac{dx}{(1-2x)^2} = \frac{n}{2}.$$

We can estimate the probability that T exceeds $0.51n$ using Chernoff-Janson bound (Thm.2.1 of [20]) proving that it is negligible. In this case we can substitute (for n large enough)

$$p_* = \frac{1}{4},\ M \sim \frac{n}{2},\ \text{and}\ \lambda = \frac{0.51}{0.5} > 1.$$

Thus we get that the expected number of interactions $T \geq 0.51n$ (parallel time ≥ 0.51) with probability less than $e^{-p_* M(\lambda - 1 - \ln \lambda)} = e^{-\frac{1}{8} n(\lambda - 1 - \ln \lambda)}$, i.e., with negligible probability. ◀

As soon as the edges are formed the timid counting begins. In order to analyze this process we define the *edge collector problem* in which one is asked to collect (draw) all edges of a given matching M of cardinality $n' > \frac{n}{4}$, with the solution guaranteed in constant parallel time by Lemma 3. In addition, one can also infer from our proof that in fact a maximum matching of cardinality $\lfloor \frac{n}{2} \rfloor$ is formed whp.

▶ **Lemma 4.** *For any cardinality* $n' \in [n/4, n/2]$*, the parallel time of the edge collector problem is* $O(n \log n)$ *whp. In addition, the parallel time needed to collect the last* $0.05 \cdot n$ *edges (of the matching) is at least* $0.4 \cdot n \ln n$ *whp.*

Proof. The probability of collecting an edge in an interaction, when i edges are still missing is $p_i = \frac{2i}{n(n-1)}$. The number of interactions needed to collect this edge is a random variable X_i which has a geometric distribution with the average $\frac{n(n-1)}{2i}$. When k edges are still to be collected, the expected number of interactions to collect extra $k-l$ edges is

$$\sum_{i=l}^{k} \frac{n(n-1)}{2i} = \frac{n(n-1)}{2}(H_k - H_l) \sim \frac{n(n-1)}{2} \ln \frac{k}{l}.$$

Using the upper bound of lower tail (Theorem 3.1) of Chernoff-Janson bounds we show that this number of interactions is at least $0.4n(n-1)\ln n$ whp, for $k = 0.05n$ and $l = n^{0.1}$.

And indeed, for n large enough one can adopt

$$p_* = p_l = \frac{2n^{0.1}}{n(n-1)},\ M \sim \frac{n(n-1)}{2}\ln(0.05n^{0.9}) > 0.44n(n-1)\ln n,\ \text{and}\ \lambda = \frac{0.4}{0.44} < 1.$$

This way we get that the number of interactions smaller than $0.4n(n-1)\ln n$ with probability smaller than $e^{-p_* M(\lambda-1-\ln\lambda)} \leq e^{-0.88n^{0.1}\ln n(\lambda-1-\ln\lambda)}$, i.e., with negligible probability.

The collection of edges concludes, when for each edge its two endpoints interact with one another. The probability of a missing interaction along some edge in the first $cn^2\log n$ interactions is $\left(1-1/\binom{n}{2}\right)^{cn^2\log n}$, which is negligible for c large enough. Thus edge collection concludes whp in $O(n^2\log n)$ interactions translating to parallel time $O(n\log n)$. ◀

In our clock protocol the value of parameter $d > 0$ depends on the constant η with respect to the high probability guarantees. We prove the existence of this parameter d, for any $\eta' = \eta + 3$.

▶ **Lemma 5.** *In a parallel time period of length n^a, for $0 < a < 1$, there are at most d interactions along any edge in the matching whp.*

Proof. By taking into account all possible subsets of d out of n^{1+a} interactions and using the union bound, the probability that an edge is a subject to at least d interactions in parallel time n^a does not exceed

$$\binom{n^{1+a}}{d}\left(\frac{2}{n(n-1)}\right)^d \leq \left(\frac{2n^a}{n-1}\right)^d.$$

and this value is smaller than $n^{-\eta'}$ is for d large enough. ◀

▶ **Lemma 6.** *In a parallel time period of length n^a, for $0.1 \leq a < 1$, there are at most $2.1n^a$ interactions along edges of the matching whp.*

Proof. The probability that a given interaction is a matching edge interaction is $\frac{2n'}{n(n-1)}$. Thus in a parallel time period of length n^a, there are expected $2n^a\frac{n'}{n-1} \leq 2n^a$ edge interactions. By the Chernoff bound the number of edge interactions is at most $2.1n^a$ whp. ◀

In what follows, depending on the context we will use the notions of counters and *levels* interchangeably.

▶ **Lemma 7.** *Let k be an integer where $k < max - d - 2$. There exists a constant c, s.t., during parallel time period $(0.51, cn\log n)$ presence of any agent on level $i < k$ guarantees whp a linear subpopulation of agents of size at least $0.1n$ on levels $j \leq k$. Also during this period no agent reaches level max whp.*

Proof. We prove validity of the lemma whp, i.e., with probability at least (wp) $1 - n^{-\eta}$. Let $\eta' = \eta + 3$. The proof is done by induction on parallel time t. First we show that in any time t of the initial parallel time period $[0.51, n^{0.2}]$ the thesis of the lemma holds wp $1 - 10tn\cdot n^{-\eta'}$. Later we prove by induction that until any considered time t the thesis of the lemma holds wp $1 - 10tn\cdot n^{-\eta'}$. Note that this guarantees that the thesis holds whp, i.e., wp $1 - n^{-\eta}$, for parallel time $t = O(n\log n)$. Assume that all events in the thesis of the lemma hold before parallel time t. We prove that if the thesis of the lemma holds before parallel time t, then it also holds in time t wp $1 - 10n^{-\eta'}$. By the inductive hypothesis before parallel time t or equivalently until parallel time $t - \frac{1}{n}$ the thesis of the lemma holds wp $1 - 10(t-\frac{1}{n})n\cdot n^{-\eta'}$. In turn, we get that until parallel time t the thesis of the lemma holds wp $1 - 10tn\cdot n^{-\eta'}$.

We first prove the base step of induction. As we proved in Lemma 3, during the initial parallel time 0.51 at least $n/2$ agents enter the clock with state (0) wp $1 - n^{-\eta'}$, when also some of these agents could already move to higher levels. By Lemma 6 applied to the initial time period $n^{0.2}$ there are at most $2.1n^{0.2}$ of the latter wp. $1 - n^{-\eta'}$. Thus in parallel time period $[0.51, n^{0.2}]$ level 0 is the host of at least $0.5n - 2.1n^{0.2} > 0.4n$ agents constantly residing at this level wp $1 - 2n^{-\eta'}$. Also, by Lemma 5 no agent reaches level max wp $1 - n^{-\eta'}$. So in parallel time $t = n^{0.2}$ the lemma holds wp $1 - 3n^{-\eta'}$. Note that $1 - 3n^{-\eta'} \geq 1 - 10tn \cdot n^{-\eta'}$ for any $t \geq 0.51$.

Now we prove the inductive step. We observe first that during parallel time period $[t', t]$, where $t' = t - n^{0.1}$, all agents which entered the clock are at least once on level $l \leq k + 1$ wp $1 - n^{-\eta'}$. And indeed during this period an agent avoids interactions with agents on levels $j \leq k$ wp at most

$$(1 - 0.1/n)^{n^{1.1}} < e^{0.1n^{0.1}}$$

Because of this and Lemma 5, during this period, any agent which entered the clock does not elevate to levels higher than $k + 1 + d$ wp $1 - 2n^{-\eta'}$. Therefore no agent reaches level max during period $[t', t]$ wp $1 - 2n^{-\eta'}$.

In order to prove the first thesis of the lemma we consider two cases.

Case 1: in this case in parallel time t' there are at least $0.11n$ agents on levels not exceeding k. Since by Lemma 6 in parallel time period $[t', t]$ at most $2.1n^{0.1}$ such agents can increase their level wp. $1 - n^{-\eta'}$. And in turn, in parallel time t there are at least $0.1n > 0.11n - 2.1n^{0.1}$ agents on levels $j \leq k$.

Case 2: in this case in parallel time t' the number of agents on levels at most k is between $0.1n$ and $0.11n$ and the number of agents on levels below k is at least $3n^{0.1}$. Let Y be the set of agents belonging to the levels above k in time t'. By Lemma 6 the probability that in parallel time period $[t', t]$ the number of agents below level k drops below $0.9n^{0.1} (= 3n^{0.1} - 2.1n^{0.1})$ is negligible, i.e., at most $n^{-\eta'}$. Consider any set X with $0.9n^{0.1}$ agents residing at levels smaller than k and estimate how many agents from set Y interact with them. For as long as $0.38n$ agents from Y do not interact with X, the probability of interaction between an unused (not in contact with agents in X) agent in Y and some agent in X is at least $0.68n^{-0.9}$. Any such interaction increases the number of agents on levels not exceeding k. Consider a sequence of $n^{1.1}$ zeros and ones in which position ι is one (1) if and only if either

- interaction ι is between an unused agent in Y with an agent in X if there are more than $0.38n$ unused agents in Y,
- if this number is smaller than $0.38n$ value 1 is drawn with a fixed probability $0.68n^{-0.9}$.

By Chernoff bound the probability that this sequence has less than $0.6n^{0.2}$ ones is negligible, i.e., at most $n^{-\eta'}$. Since $0.12n < 0.11n + 0.6n^{0.2}$ this sequence has less than $0.6n^{0.2}$ ones only when the number of agents elevated to levels not exceeding k is smaller than $0.6n^{0.2}$. Also by Lemma 6 during parallel time eriod $[t', t]$ at most $2.1n^{0.1}$ other agents may increase their level beyond k wp $1 - n^{-\eta'}$. So in Case 2 the number of agents on levels not exceeding k increases during period $[t', t]$ by at least $0.6n^{0.2} - 2.1n^{0.1}$.

Case 3: assume that in parallel time t' the number of agents on levels $j \leq k$ is between $0.1n$ and $0.11n$ and also the number of agents on levels below k is smaller than $3n^{0.1}$. The probability of an interaction between one of such agents and an agent in set Y of agents above level k is at most $6n^{-0.9}$. Any such interaction increases the number of agents on levels not exceeding k. By Chernoff bound the probability that this number of interactions exceeds $7n^{0.2}$ in $[t', t]$ is negligible, i.e., at most $n^{-\eta'}$. Thus in Case 3 the probability that the number of agents on levels at most k exceeds $0.12n > 0.11m + 7n^{0.2}$ is negligible, i.e., at most $n^{-\eta'}$.

We now formulate Claim 1 that upperbounds the number of agents leaving levels $j \leq k$ and Claim 2 that bounds from below the number of agents joining these levels in Case 3. Because wp $1 - 6n^{-\eta'}$ the levels $j \leq k$ gain agents as a result of these two processes. This will conclude the proof.

▷ Claim 1. In Case 3 during parallel time period $[t', t]$ there are at most $0.26n^{0.1}$ agents located at levels $j \leq k$ which increment their level wp $1 - 4n^{-\eta'}$.

And indeed, for as long as there are at most $0.12n$ agents on levels not greater than k, the probability that such agent interacts as the initiator with a clock agent is at most $0.12/n$. Such an interaction increments the level of this clock agent with probability at most $0.12/n$. We prove that more than $0.13n^{0.1}$ such increments occur in $[t', t]$ with probability at most $4n^{-\eta'}$. Consider a sequence of $n^{1.1}$ zeros and ones in which position ι is one if and only if either

- interaction ι increments initiator's level and there are at most $0.12n$ agents on levels not greater than k
- if this number is greater than $0.38n$ value 1 is drawn with a fixed probability $0.12/n$.

By Chernoff bound this sequence has less than $0.13n^{0.1}$ ones (1s) wp. $1 - n^{-\eta'}$. On the other hand we have at most $0.12n$ agents on levels at most k wp $1 - n^{-\eta'}$. Thus wp $1 - 2n^{-\eta'}$ at most $0.13n^{0.1}$ agents on levels not exceeding k can increment their levels in $[t', t]$ acting as initiators. Analogously, we can prove that wp $1 - 2n^{-\eta'}$ at most $0.13n^{0.1}$ agents on levels not exceeding k can increment their levels in $[t', t]$ acting as responders. So altogether at most $0.26n^{0.1}$ agents on levels $j \leq k$ increment their levels during s$[t', t]$ wp $1 - 4n^{-\eta'}$.

▷ Claim 2. In Case 3 during parallel time period $[t', t]$ there are at least $0.75n^{0.1}$ interactions between agents on levels $i < k$ and those residing on levels higher than k wp $1 - 2n^{-\eta'}$.

For as long as there are at most $0.12n$ agents on levels at most k, at least $0.38n = n/2 - 0.12n$ agents are on levels higher than k. The probability of interaction between such agents and an agent on level $i < k$ is at least $0.76/n = 2 \cdot 0.38/n$. Any such an interaction increases the number of agents on levels not exceeding k. Consider a sequence of $n^{1.1}$ zeros and ones in which at position ι is one (1) if and only if either

- there are at most $0.12n$ agents on levels not greater than k and interaction ι increases the number of such agents
- the number of agents on levels up to k is greater than $0.12n$ and value 1 is drawn with a fixed probability $0.76/n$.

By Chernoff bound this sequence has more than $0.75n^{0.1}$ ones (1s) wp $1 - n^{-\eta'}$. On the other hand we have at most $0.12n$ agents on levels at most k wp $1 - n^{-\eta'}$. Thus wp $1 - 2n^{-\eta'}$ at least $0.75n^{0.1}$ agents on levels exceeding k can reduce their levels to at most k during $[t', t]$ while acting as initiators.

Because of both Claims 1 and 2 after parallel time period $[t', t]$ there are at least $0.51n^{0.1} (= 0.75n^{0.1} - 0.24n^{0.1})$ more agents on levels $j \leq k$ than in parallel time t'. This proves that in parallel time t there are at least $0.1n$ agents on levels $j \leq k$. ◀

▶ **Lemma 8.** *The parallel time in which the first agent achieves level max is greater than* $(max - d - 2) \cdot 0.4n \ln n$ *whp.*

Proof. Let t_k be the time when for the first time there are no agents available at levels lower than k. By Lemma 7 during period $[0.51, t_k]$, there are at least $0.1n$ agents on level k or lower. Let $n_k \in [n/4, n/2]$ be the number of edges at time t_k. Thus between time t_k and t_{k+1} at least $0.1n$ agents must increment their levels to $k + 1$. This is done by collecting (interacting via) edges adjacent to them. By Lemma 4 this takes parallel time at least $0.4n \ln n$. This process is repeated for $max - d - 2$ levels when no agents reach level max whp. ◀

▶ **Lemma 9.** *The first agent moves to level max in parallel time $O(n \log n)$ whp.*

Proof. The total parallel time to form $\lfloor n/2 \rfloor$ edges is $O(n \log n)$ whp by Lemma 2. If the first agent achieves level *max* earlier the lemma remains true. If this is not the case, the parallel time $O(n \log n)$ is determined by collection of all $\lfloor n/2 \rfloor$ edges which needs to be repeated *max* times (to reach the highest level) resulting also in the total parallel time $O(n \log n)$. ◀

Now we are ready to prove Theorem 1. The thesis for matching based clock follows directly from Lemmas 8 and 9. The thesis for the leader based clock can be proved by a sequence of lemmas almost identical to Lemmas 7, 8 and 9. In the analogue of Lemma 7 we can take $n - 1$ followers instead of n' edges. This is because Lemma 2 assures that the parallel time counted by the matching based clock is long enough to form all edges whp. All agents are initiated at level 0 of the leader based clock in parallel time $O(\log n)$ whp, by the epidemic started by the leader. This is followed by removal of all edges of the matching based clock in time $O(n \log n)$ whp. This gives the initialisation of the leader based clock in parallel time $O(n \log n)$.

4 Optimal spanning line formation

In the *spanning line formation* problem, starting in edge-less configuration the task for n agents is to form a sequence $a_{i_0}, \ldots, a_{i_{n-1}}$, in which pairs of agents $a_{i_j}, a_{i_{j+1}}$, for $j \in \{0, \ldots .n-2\}$, become connected by edges, which are the only edges present in the population. We prove that any spanning line formation protocol requires $\Omega(n \log n)$ parallel time. We also propose an optimal spanning line formation protocol which stabilises in parallel time $\Theta(n \log n)$ whp.

▶ **Theorem 10.** *Spanning line formation stabilising whp requires $\Omega(n \log n)$ parallel time.*

Proof. The final spanning line configuration must be preceded by one of the three critical configurations including A) *two lines*, where one of four edges could be inserted to form a line, B) *buffalo*, where one specific edge needs to be removed, or C) *unicorn*, where one of the two edges needs to be removed. Alternatively, the final line configuration is obtained from cycle configuration (a cycle containing all agents) from which one edge is removed. In such case we consider the only two cycle preceding configurations including D) *line*, where a unique edge need to be inserted, or E) *chord*, where specific chord needs to be removed, see Figure 1. Thus to stabilise in the final spanning line configuration, the protocol has to go through one of the *bottleneck* transitions having a choice of a fixed (at most 4) number of edges. This limited choice forces $\Omega(n \log n)$ parallel time if we insist on high probability.

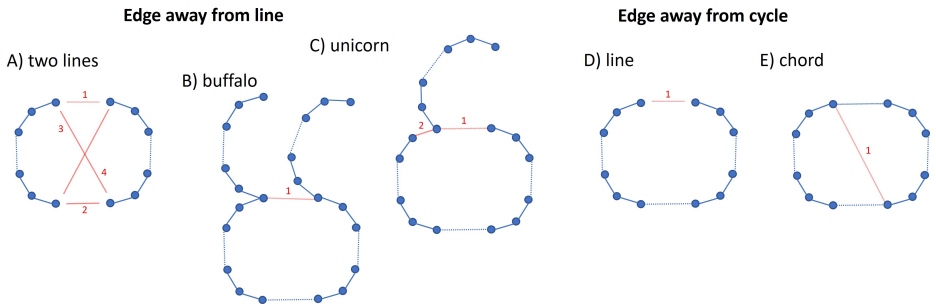


Figure 1 Configurations leading to line and cycle.

Let T be the parallel time required to stabilise in the final spanning line configuration whp. As indicated above, stabilising in such configuration requires passing through a bottleneck transition. At any time, when in a critical configuration, the probability of choosing at random a pair of agents which enables a bottleneck transition is at most $4/\binom{n}{2} = \frac{8}{n(n-1)}$, since there are at most four such pairs for each bottleneck transition. Assume $T < \eta(n-1)\ln n/10$, where η is the parameter of whp requirement. The probability that the algorithm fails in this time is not smaller than the probability of no bottleneck transition which is at least

$$\left(1 - \frac{8}{n(n-1)}\right)^{\eta n(n-1)\ln n/10} \sim n^{-0.8\eta} > n^{-\eta}.$$

Thus also the probability of a spanning line formation protocol not stabilising in parallel time T is larger than $n^{-\eta}$, for n large enough. ◀

Optimal line formation. The protocol is preceded by leader election verified by the matching based clock. And when this happens, the (periodic) leader based clock starts running simultaneously with the following line formation protocol based on two main rules defined below. The constant size state space of the combined protocol is a Cartesian product of the individual states spaces of the leader based clock and the line formation protocols.

Form head and tail

$$L + F + 0 \to H + T + 1$$

This rule creates the initial head in state H and the tail in state T of the newly formed line. Note that since the line formation process uses separate memory the leader in the leader based clock remains in the leadership state, i.e., it is the head state H is used solely in the line formation protocol.

Extend the line

$$H + F + 0 \to R + H + 1$$

This rule extends the current line by addition of an extra agent from the head end of the line. The state R indicates that the agent is in the line between the head and the tail.

▶ **Theorem 11.** *The spanning line formation protocol stabilises whp in parallel time* $O(n \log n)$.

Proof. The probability of an interaction adding agent $i+1$ to the line when i agents are already present is $\frac{2(n-i)}{n(n-1)}$. Such interactions has geometric distribution with the expected value $\frac{n(n-1)}{2(n-i)}$. Thus the expected parallel time of forming the line is

$$\frac{1}{n}\sum_{i=1}^{n}\frac{n(n-1)}{2(n-i)} \sim \frac{n}{2}\sum_{i=1}^{n}\frac{1}{i} \sim \frac{n\ln n}{2}.$$

By Chernoff-Janson bound this parallel time $O(n \log n)$ is guaranteed whp. ◀

In order to make the line formation protocol always correct we need some backup rules for the unlikely case of desynchronisation when two or more leaders survive to the line formation stage. In such case we need to continue leader elimination.

$$L + L + 0 \longrightarrow L + F + 0$$

Also when a leader meets already formed head.

$$L + H + 0 \to F + H + 0$$

Finally we have to dismantle excessive lines if two or more lines are formed. This is done using extra state D which dismantles the line edge by edge starting from the head.

$$H + H + 0 \to H + D + 0$$

$$D + R + 1 \to F + D + 0$$

$$D + T + 1 \to F + F + 0$$

5 Probabilistic bubble-sort

Let array $A[0..n-1]$ contain an arbitrary sequence of n numbers. In the *probabilistic bubble-sort* during each comparison step an index $j \in \{0, \ldots, n-2\}$ is chosen uniformly at random, and if $A[j] > A[j+1]$ these two values are swapped in A. We show that the expected number of comparisons required to sort all numbers in A (in the increasing order) is $\Theta(n^2)$ whp.

In order to prove this result we first remind the reader that any sorting procedure based on fixing local inversions requires $\Omega(n^2)$ comparisons. In order to prove the upper bound we utilise the classical *zero-one principle* stating that if a (probabilistic) sorting network sorts correctly all sequences of zeros and ones, it also sorts an arbitrary sequence of numbers of the same length. More precisely, if we want to prove that a given sequence X of n numbers will be sorted we have to consider only $n+1$ zero-one sequences obtained by replacing k largest elements of X by ones and the remaining elements by zeros, for any $k = 0, \ldots, n$, see [22]. Thus it is enough to prove that the probabilistic bubble-sort utilises $O(n^2)$ comparisons to sort whp any zero-one sequence of length n, and later use the union bound to extend this result to any sequence of numbers, also whp.

▶ **Theorem 12.** *The probabilistic bubble-sort utilises* $4(n-1)(n \ln 2 + \eta \ln n)$ *comparisons whp to sort any zero-one sequence of size* n.

Let k be the number of ones in a zero-one sequence represented by A. We define a *configuration* C as the subset of all positions in A at which ones are situated, where $|C| = k$. The probabilistic bubble-sort starts in the initial configuration (based on the original zero-one sequence) and thanks to the conditional swaps progresses through consecutive configurations including the final one in which all zeros precede k ones. For any configuration C, we define a *potential function* $P(C) = \sum_{i \in C} P[i]$ with a non-negative integer value, where

$$P[i] = 2^{n-k+2l-i} - 2^l \text{ , for } l = |C \cap \{0, \ldots, i-1\}|.$$

Note that the value of this potential is zero for all i if and only if the sequence is sorted. Thus $P(C) = 0$ for a sorted sequence C. Also, when all ones precede all zeros, the potential $P(C)$ is the highest possible. One can notice that always $P(C) < 2^n$.

We prove the following lemma.

▶ **Lemma 13.** *Let* C *be an arbitrary configuration in* A *and* $EP(C')$ *be the expected potential of the next configuration* C' *in the probabilistic bubble-sort. The following inequality holds.*

$$EP(C') \leq \left(1 - \frac{1}{4(n-1)}\right) P(C).$$

Proof. We split configuration C into disjoint blocks of indices $B_1, B_2, \ldots$, each corresponding to a solid run of ones. For any block $B = \{x, \ldots, y\}$ we define a potential $P(B) = \sum_{i=x}^{y} P[i]$. In the subsequent configuration C', let $EP(B')$ be the expected potential of $B' \subset C'$ based on the ones originating from B in the preceding configuration C. We show that

$$EP(B') \leq \left(1 - \frac{1}{4(n-1)}\right) P(B).$$

Let $l = |C \cap \{0, \ldots, y-1\}|$. We have

$$P(B) = \sum_{i=x}^{y} P[i] = \sum_{i=x}^{y} 2^{n-k+2(l+i-y)-i} - \sum_{i=x}^{y} 2^{l+i-y} < 2^{n-k+2l-y+1}$$

Assume first that $y = n-1$. The inequality follows from the fact that $P(B) = P(B') = 0$ as ones located at positions in B cannot be moved any further. Thus we can assume that $y < n-1$. Now, as either $B' = B$ or $B' = \{x, \ldots, y-1\} \cup \{y+1\}$ and the latter happens with probability $\frac{1}{n-1}$, we get

$$P(B') = \sum_{i=x}^{y-1} 2^{n-k+2(l+i-y)-i} - \sum_{i=x}^{y} 2^{l+i-y} + 2^{n-k+2l-y-1} =$$

$$= P(B) - 2^{n-k+2l-y-1} \leq \frac{3}{4} P(B).$$

And in turn

$$EP(B') \leq \left(1 - \frac{1}{n-1}\right) P(B) + \frac{1}{n-1} \cdot \frac{3}{4} P(B) = \left(1 - \frac{1}{4(n-1)}\right) P(B).$$

Note that any configuration C is the union of disjoint blocks B_i and $P(C) = \sum_i P(B_i)$, thus also

$$EP(C') = \sum_i EP(B_i') \leq \sum_i \left(1 - \frac{1}{4(n-1)}\right) P(B_i) = \left(1 - \frac{1}{4(n-1)}\right) P(C) \qquad \blacktriangleleft$$

The initial value of $P(C_0)$ is bounded by 2^n. When after t random comparisons $EP(C_t) \leq n^{-\eta}$ the sequence is sorted whp. This holds because the probability that after t random comparisons the sequence is not sorted is equal to the probability that the potential is greater than zero (i.e., at least 1 as the potential is always integral). This probability is less than or equal to $EP(C_t)$ which is not bigger than $n^{-\eta}$.

Let $c = \left(1 - \frac{1}{4(n-1)}\right)$. Let also $P(C_j)$ and $P(C_{j+1})$ be the potentials of the configurations separated by the jth consecutive comparison. We have shown earlier that $EP(C_{j+1})$, conditioned on the value of $P(C_j)$, is at most $c \cdot P(C_j)$. This implies that the unconditional value of $EP(C_{j+1})$ is at most $c \cdot EP(C_j)$. Thus by an induction argument it follows that after t random comparisons $EP(C_t)$ is at most $c^t \cdot EP(C_0)$. Finally as $EP(C_0) = P(C_0)$, where C_0 is the initial configuration and its potential is not a random variable, in order to estimate t we get inequality

$$EP(C_t) \leq \left(1 - \frac{1}{4(n-1)}\right)^t P(C_0) \leq \exp\left(-\frac{t}{4(n-1)}\right) 2^n \leq n^{-\eta},$$

which holds for $n \ln 2 + \eta \ln n \leq \frac{t}{4(n-1)}$, equivalent with

$$t \geq 4(n-1)(n \ln 2 + \eta \ln n).$$

This concludes the proof of Theorem 12.

6 Strand self-replication

A *strand* is a line segment $a_{i_0}, \ldots, a_{i_{k-1}}$ in which each agent holds a 0/1-bit of information and the only pairs of agents connected by edges are $a_{i_j}, a_{i_{j+1}}$, for all $j \in \{0, \ldots, k-2\}$. The front agent a_{i_0} in a strand is called *the head*, the last one $a_{i_{k-1}}$ is called *the tail*, and the remaining ones are referred to as *regular* or *internal* agents. In the *strand self-replication* problem the agents forming a strand are asked to create an identical copy of this strand from freely available (disconnected) agents. In this section we propose and analyze the first strand self-replication mechanism allowing efficient concurrent reproduction of many, possibly different, *strands*.

Our strand self-replication protocol mimics the pipelining process utilised and analyzed in the probabilistic bubble-sort algorithm in Section 5. There are, however, some small differences between the two processes. In particular, in strand self-replication the transfer (pipelining) of consecutive bits of information between the old and the new strand is done at the same time as the new strand is being constructed. Also any bit transferred to the new strand stops moving as soon as it finds the first unused (newly added) agent in the new strand. Finally, the probability of using an edge in the strand is $1/\binom{n}{2}$ comparing to the probability $1/k$ of choosing any pair of numbers in the probabilistic bubble-sort applied to sequence of size k. In the proof of Theorem 14 we point out that only the last difference separates the self-replication process by a multiplicative factor $\Theta(n^2/k)$ from bubble-sort applied to a sequence with k ones on the left and $2k$ zeros at the right end.

The Algorithm. When a strand is ready for self-replication it first creates a copy of its head, then pushes through this new head (one by one, preserving the order) the bits of information pulled from its own agents. At the same time, in order to accept the incoming bits of information, first the new head and later the copies of the consecutive regular agents of the replicated strand extend the new strand until the tail is formed. When the last (tail) bit of information is delivered to the new strand, the edge bond bridging the two heads is removed and the original (old) strand is ready for the next round of self-replication.

Note that in this version of the self-replication protocol a newly formed strand may simultaneously seek its tail extension and already be involved in self-replication from its head end. In addition, when eventually all free agents are used, a large volume of agents will be likely stuck in partially replicated strands.

▶ **Theorem 14.** *The strand self-replication protocol recreates a k-bit strand in parallel time $O(n(k + \log n))$ whp.*

Proof. See the Appendix including Lemmas 16 and 17. ◀

▶ **Corollary 15.** *The strand self-replication protocol generates l copies of a k-bit strand in parallel time $O(n(k + \log n) \log l)$ whp.*

6.1 Pattern matching with strand self-replication

Pattern matching is a classical problem in Algorithms [13]. In this problem there are two strings, a shorter *pattern* P of length k and usually much longer *text* T of length m. The main task in pattern matching is to find all occurrences of P in T. We demonstrate how to utilise self-replication mechanism in pattern matching in the network constructors model.

Assume we have two strands, one containing P^R (reversed sequence of bits in P) and the other containing T. One can find all occurrences of P in T by forming a single strand containing sequence $T \cdot P^R$, further injection to and pipelining across T the consecutive bits

of P^R while adopting the pattern matching procedure from [11]. Using this approach and Theorem 17 one can prove that utilising a fixed number of states the parallel time of finding all occurrences of P in T is $O(n(m+k+\log n))$ whp.

The pattern matching protocol can be improved by instructing the original strand and all replicas containing P^R to alternate between insertion of its content at a random position in T and self-replication. Each insertion and self-replication takes time $O(n(k+\log n))$. After $\frac{m}{k}$ pattern replications and insertions the distance between any two consecutive insertion points in strand T is $O(k \log m)$ whp, for m large enough. Thus the parallel time of the improved protocol is independent from m and is bounded by $O(n(k+\log n)\log n)$, where $O(n(k+\log n)\log \frac{m}{k})$ comes from all insertions and self-replications, and $O(n(k\log m+\log n))$ refers to pattern matching on each segment of size $O(k \log m)$.

7 Open Problems

Going beyond the proposed strand self-replication protocol one could investigate whether other network structures can self-replicate and at what cost. Also further studies on utilisation of strands (as carriers of complex information) in more complex distributed processes is needed. One should also seek alternative population protocol models which capture the bottleneck of infrequent edge manipulation. This can be done, for example, by imposing a greater bias on interactions between pairs of agents already connected by edges.

References

1 D. Alistarh, J. Aspnes, D. Eisenstat, R. Gelashvili, and R.L. Rivest. Time-space trade-offs in population protocols. In *Proc. SODA 2017*, pages 2560–2579, 2017.

2 D. Alistarh, J. Aspnes, and R. Gelashvili. Space-optimal majority in population protocols. In *Proc. SODA 2018*, pages 2221–2239, 2018.

3 D. Alistarh and R. Gelashvili. Polylogarithmic-time leader election in population protocols. In *Proc. ICALP 2015*, pages 479–491, 2015.

4 D. Angluin, J. Aspnes, Z. Diamadi, M.J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc. PODC 2004*, pages 290–299, 2004.

5 D. Angluin, J. Aspnes, and D. Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008.

6 P. Berenbrink, G. Giakkoupis, and P. Kling. Optimal time and space leader election in population protocols. In *Proc. STOC 2020*, pages 119–129, 2020.

7 P. Berenbrink, D. Kaaser, P. Kling, and L. Otterbach. Simple and efficient leader election. In *Proc. SOSA 2018*, volume 61 of *OASICS*, pages 9:1–9:11, 2018.

8 A. Bilke, C. Cooper, R. Elsässer, and T. Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In *Proc. PODC 2017*, pages 451–453, 2017.

9 I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P.G. Spirakis. Passively mobile communicating machines that use restricted space. *TCS*, 412(46):6469–6483, 2011.

10 H-L Chen, R. Cummings, D. Doty, and D. Soloveichik. Speed faults in computation by chemical reaction networks. In *Proc. DISC 2014*, pages 16–30, 2014.

11 B.S. Chlebus and L. Gąsieniec. Optimal pattern matching on meshes. In *STACS 94, 11th Annual Symposium on Theoretical Aspects of Computer Science*, volume 775 of *Lecture Notes in Computer Science*, pages 213–224. Springer, 1994.

12 M. Connor, O. Michail, and P. Spirakis. On the distributed construction of stable networks in polylogarithmic parallel time. *Information*, 12(6):254–266, 2021.

13 M. Crochemore and T. Lecroq. *String Matching*, pages 2113–2117. Springer New York, 2016.

14 A. Czumaj and A. Lingas. On truly parallel time in population protocols. *CoRR*, abs/2108.11613, 2021.
15 D. Doty. Timing in chemical reaction networks. In *Proc. SODA 2014*, pages 772–784, 2014.
16 D. Doty, M. Eftekhari, L. Gąsieniec, E.E. Severson, G. Stachowiak, and P. Uznański. A time and space optimal stable population protocol solving exact majority. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1044–1055, 2021.
17 D. Doty and D. Soloveichik. Stable leader election in population protocols requires linear time. In *Proc. DISC 2015*, pages 602–616, 2015.
18 L. Gąsieniec and G. Stachowiak. Enhanced phase clocks, population protocols, and fast space optimal leader election. *J. ACM*, 68(1):2:1–2:21, 2021.
19 L. Gąsieniec, G. Stachowiak, and P. Uznański. Almost logarithmic-time space optimal leader election in population protocols. In *Proc. SPAA 2019*, pages 93–102, 2019.
20 S. Janson. Tail bounds for sums of geometric and exponential variables. *Satistics and Probability Letters*, 135(1):1–6, 2018.
21 R.A. Freitas Jr and R.C. Merkle. *Kinematic Self-Replicating Machines*. Landes Bioscience, Georgetown, TX, 2004.
22 D.E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
23 T.A. Lincoln and G.F. Joyce. Self-sustained replication of an rna enzyme. In *Science, Vol 323, Issue 5918*, pages 1229–1232. American Association for the Advancement of Science, 2009.
24 O. Michail and P. Spirakis. Simple and efficient local codes for distributed stable network construction. *Distributed Computing*, 29(3):207–237, 2016.
25 E. Moulin and N. Giuseppone. Dynamic combinatorial self-replicating systems. In *Constitutional Dynamic Chemistry*, pages 87–105. Springer, 2011.

A Appendix

Proof of Theorem 14. We start with presenting further detail of the self-replication protocol. As in all other protocols studied in this paper, the agents utilise a constant number of states, this time organised into the following triplets

$$< Role, B_i, Buffer >,$$

where the three attributes are:

Role refers to the strand's *head* agent H, the *tail* agent T, or to *a regular* contributor R.

B$_i$ refers to the bit of information combined with its position i in the strand. Note that each position i is computed (and stored) modulo 3 counting from the head's position 0. This allows agents to distinguish between the two directions: towards the head and towards the tail on the strand. We use B_T to denote the bit located in the tail agent of the strand with a special index -1. Finally, by $|B_i|$ we denote the sole value of the information bit without its location. Thus the binary representation of the information stored in the replicated strand corresponds to the sequence $|B_0|, |B_1|, \ldots, |B_T|$, and in turn for all $B_i \neq B_T$ we have $B_i =< |B_i|, i \mod 3 >$, and $B_T =< |B_T|, -1 >$.

Buffer is a part of agents' memory handling single information $|B_i|$ bits or control messages. An agent is in the neutral state ϕ when its buffer is empty and no dedicated replication action (apart from waiting for further instructions) is needed from this agent. In the *self-replicated strand* state ϕ^H denotes the empty buffer of an agent supporting bit transfer towards H. Similarly, when the buffer is occupied by a bit $|B_x|$ moving towards H the relevant state is $|B_x|^H$.
In the *newly formed strand* state ψ^T denotes the empty buffer of an agent supporting bit transfer towards the tail end. And similarly, when the buffer is occupied by a bit $|B_x|$ moving towards the tail end the relevant state is $|B_x|^T$. Here the control message ψ^N

indicates that further extension is expected at the current end of the new strand. In this strand we distinguish also state ψ (await further information) in agents just added at the tail end.

Below we explain how the information (the carried sequence of bits) is transferred from the old to the new strand. The full list of strand self-replication protocol rules follows. The relevant diagrams with state transitions in the old and the new strands are shown in Figure 2 and Figure 3 respectively. Please note that this set of rules is designed for strands containing at least three agents, i.e., when all types of agents H, T and R are used. The relevant protocols for shorter strands are trivial as they carry only a constant number of bits.

(R1) Start of the strand self-replication. The replication process begins when the head H in the neutral state ϕ interacts with a free agent in state F.

$$< H, B_0, \phi > + F \ + \ 0 \ \longrightarrow$$

$$< H, B_0, \phi^H > + < H, B_0, \psi > + 1$$

When this rule is applied, in the old strand signal ϕ^H (move all bits towards the head) is created, and in the new strand signal ψ means await further instructions (either to add a new agent or to conclude the replication process).

(R2) Create $|B_i|^H$ or $|B_T|^H$ bit message. When signal ϕ^H arrives at the $(i-1)^{th}$ agent and the i^{th} agent is neutral, message $|B_i|^H$ is placed in the buffer of the latter.

$$< R, B_i, \phi > + < R|H, B_{i-1}, \phi^H > + 1 \ \longrightarrow$$

$$< R, B_i, |B_i|^H > + < R|H, B_{i-1}, \phi^H > + 1$$

A similar action is taken at the tail agent in neutral state $< T, B_T, \phi >$

$$< T, B_T, \phi > + < R|H, B_{i-1}, \phi^H > + 1 \ \longrightarrow$$

$$< T, B_T, |B_T|^H > + < R|H, B_{i-1}, \phi^H > + 1$$

The rules in **R2** enable propagation of the request to pipeline all information bits towards the head H. The rules **R3** and **R4** govern the relevant bit movement.

(R3) Move a non-tail bit message $|B_x|^H$ towards H.

$$< R, B_i, |B_x|^H > + < R|H, B_{i-1}, \phi^H > + 1 \ \longrightarrow$$

$$< R, B_i, \phi^H > + < R|H, B_{i-1}, |B_x|^H > \ +1$$

Note that when the bit message $|B_x|$ is moved state ϕ^H requesting further bit messages remains in the i^{th} agent.

(R4) Move the tail bit message $|B_T|^H$ towards H.

$$< T|R, B_i, |B_T|^H > + < R|H, B_{i-1}, \phi^H > + 1 \ \longrightarrow$$

$$< T|R, B_i, \phi > + < R|H, B_{i-1}, |B_T|^H > + 1$$

Note that when the tail message $|B_T|^H$ is moved the neutrality of the tail agent is restored. Eventually, thanks to the final transfer of the tail message (to the new strand) states of all buffers in the old strand are reset to ϕ.

The following two rules govern transfer of bit messages between the old and the new strand.

(R5) Transfer a non-tail bit message $|B_x|^H$ to the head of the new strand.

$< H, B_0, |B_x|^H > + < H, B_0, \psi^T > + 1 \longrightarrow$

$< H, B_0, \phi^H > + < H, B_0, |B_x|^T > + 1$

After the transfer across the two strands the bit message is now targeting the tail end.

(R6) Transfer the tail bit message $|B_T|^H$ to the head of the new strand.

$< H, B_0, |B_T|^H > + < H, B_0, \psi^T > + 1 \longrightarrow$

$< H, B_0, \phi > + < H, B_0, |B_T|^T > + 0$

As indicated earlier, transfer of the tail message to the new strand and removal of the bridging edge restore the neutrality of the old strand which is now ready to reproduce again.

Finally, we discuss the remaining rules governing the new strand creation. Recall that the control message represented by state ψ at the current end of the new strand indicates that this strand can be still extended.

(R7) Move a non-tail message $|B_x|^T$ towards the tail end.

$< H|R, B_i, |B_x|^T > + < R, B_{i+1}, \psi^T > + 1 \longrightarrow$

$< H|R, B_i, \psi^T > + < R, B_{i+1}, |B_x|^T > + 1$

After this move the i^{th} agent in the new strand awaits further bit messages.

(R8) Move the tail message $|B_T|^T$ towards the tail end.

$< H|R, B_i, |B_T|^T > + < R, B_{i+1}, \psi^T > + 1 \longrightarrow$

$< H|R, B_i, \phi > + < R, B_{i+1}, |B_T|^T > + 1 >$

After this move the neutrality of the i^{th} agent in the new strand is restored, i.e., no further bit messages from the head end are expected.

When there is no room for the bit message coming from the head end another agent has to be added to the tail end of the new strand. This is done in two steps. In the first step the current tail end requests addition of a new agent with control message ψ^N.

(R9) Request strand extension with ψ^N on non-tail bit message $|B_x|^T$ arrival.

$< R, B_i, |B_x|^T > + < R, B_{i+1}, \psi > + 1 \longrightarrow$

$< R, B_i, |B_x|^T > + < R, B_{i+1}, \psi^N > + 1.$

The analogous rule requesting extension beyond the head of the new strand is

$< H, B_0, |B_1|^H > + < H, B_0, \psi > + 1 \longrightarrow$

$< H, B_0, |B_1|^H > + < H, B_0, \psi^N > + 1.$

When ready (signal ψ^N is present) the new agent is added from the pool of free agents.

(R10) Extend the new strand.

$$< H|R, B_i, \psi^N > + F \ + \ 0 \ \longrightarrow$$

$$< H|R, B_i, \psi^T > + < R, *, \psi > + 1$$

Note that after this rule is applied the newly added agent still awaits its bit message which is denoted by $*$. This new bit message arrives with the help of the following two rules.

(R11) Arrival of a non-tail bit message.

$$< H|R, B_i, |B_x|^T > + < R, *, \psi > + 1 \ \longrightarrow$$

$$< H|R, B_i, \psi^T > + < R, B_x, \psi > + 1$$

As a non-tail bit arrived the new strand will be still extended which is denoted by messages ψ^T (expect more bit messages from the head end) in the i^{th} agent and ψ (further extension still possible). The situation is different when the tail bit message arrives.

(R12) Arrival of the tail bit message.

$$< R, B_i, |B_T|^T > + < R, *, \psi > + 1 \ \longrightarrow$$

$$< R, B_i, \phi > + < T, B_T, \phi > + 1$$

After this rule is applied the neutrality at the tail end of the new strand is restored.

Note, however, that since the neutrality of the agents closer to the head of this strand was restored earlier the front of the new strand can be already involved in the next strand replication process. But since we use different messages for the transfers in the old and the new strands, the two simultaneously run processes will not interrupt one another.

We conclude the proof of Theorem 14 with Lemma 16 stating the correctness of the proposed self-replication protocol, and Lemma 17 addressing the parallel time complexity.

▶ **Lemma 16.** *The strand self-replication protocol based on rules* **R1-R12** *is correct.*

Proof. We argue first about correctness of the proposed protocol in the replicated (old) strand. One can observe that the bit messages stored in the agents of the strand move along consecutive edges towards the head H. They do not change their order as they only move when the preceding bit message vacates the relevant buffer. Finally, to conclude the replication process neutrality of each agent need to be restored, and this is done by the eventual transfer of the tail message $|B_T|^H$. In what follows we discuss the actions in all three types of agents in the strand.

- The actions of the tail node are governed by rules **R2** and **R4**. The first rule creates bit message $|B_T|^H$ and the second moves this message towards the head of the strand, restoring the neutrality of the tail agent.
- The actions of a regular node require also rule **R3** which supports movement of multiple non-tail bit messages towards H. And when the tail bit message arrives the neutrality of this regular agent is restored by rule **R4** applied to this agent twice, first on the right then on the left side of this rule.

- The actions of head H are more complex. The self-replication begins with application of rule **R1** which comprises three different actions: forming a bridging edge, adding the head of the new strand, and replication of its bit message in the newly formed head. This is followed by transfer of non-tail bit messages to the new strand by alternating use of rules **R3** and **R5**. When eventually the tail bit message arrives during application of rule **R4**, the neutrality of the head is restored by rule **R6**. This concludes the replication process.

For the full cycles of rules utilised in the replicated strand see Figure 2 in the Appendix. The new strand formation requires different organisation of states and transitions. Note that all agents added to the strand must originate in state F, see Figure 3. Also in this case we argue that the bit messages arrive in the unchanged order and eventually the neutrality of all agents is reached (starting from the head and finishing with the tail agent) with the help of the tail bit message $|B_T|^T$.

- Formation of the tail agent requires application of only two rules: **R10** to add a new agent and **R12** to equip this agents with the tail message $|B_T|$, when neutrality of this agent is reached.
- The situation with the regular nodes is more complex as they have to accept and store their own bit message $|B_i|$ (done by rule **R11**), add additional agent (via alternating application of rules **R9** and **R10**) moving all non-tail bit messages following $|B_i|$ in the old strand (rule **R7**) until the tail bit message arrives (rule **R8**) and finally neutrality of the regular agents is reached via rule **R8** or rule **R12** if the agent precedes the tail agent.
- Rule **R1** creates the head of the new strand, rules **R9** and **R10** add a new agent, rules **R5** and **R7** move non-tail bit messages in the direction of the tail until the tail bit message arrives (rule **R6**) when the neutrality of the head is reached (rule **R8**).

For the full cycle of rules used by agents in the replicated strand see Figure 3 (Appendix). As discussed earlier in the new strand what matters is that neutrality is reached earlier by agents located closer to the head, as this strand is allowed to start self-replication while some bit messages (from the old strand) are still being moved towards the tail end (which may not be fully formed yet). However, it is enough to observe that these two replication processes are independent as they are based on movement of bit messages towards the opposite directions and in turn they share no rules. ◀

▶ **Lemma 17.** *The strand self-replication protocol based on rules* **R1-R12** *stabilises in parallel time* $O(n(k+\log n))$ *whp.*

Proof. The strand self-replication protocol mimics the pipelining mechanism utilised and analyzed in the probabilistic bubble-sort procedure. The main differences is the fact that the bits of information are moved along the original and the new strand at the same time as the new strand is being constructed. In particular, when the leading bit reaches the current end of the new strand, the extension request (for a new agent and edge connection) is successful with probability $\leq 1/\binom{n}{2}$. Also move of any bit along an existing edge is successful with probability $1/\binom{n}{2}$. Thus the expected potential change associated with one interaction (the counterpart of the inequality from Lemma 13) is

$$EP(C') \leq \left(1 - \frac{1}{2n(n-1)}\right) P(C).$$

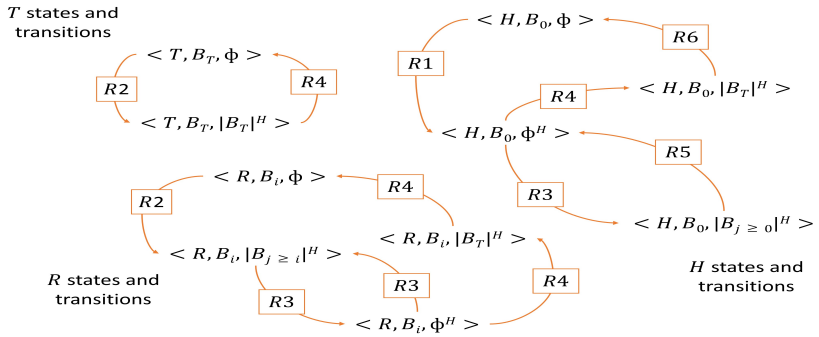


Figure 2 The old strand states and transitions.

As the total number of extension requests is k and the longest distance any bit has to move is $2k$, we get the initial potential $P(C_0) \leq 2^{3k}$ in this case. In order to estimate the number of interactions t, after which $EP(C_t) \leq n^{-\eta}$, we get inequalities

$$EP(C_t) \leq \left(1 - \frac{1}{2n(n-1)}\right)^t \cdot P(C_0) \leq \exp\left(-\frac{t}{2n(n-1)}\right) 2^{3k} \leq n^{-\eta},$$

which holds for $3k \ln 2 + \eta \ln n \leq \frac{t}{2n(n-1)}$ and in turn for $t \geq 2n(n-1)(3k \ln 2 + \eta \ln n)$. ◀

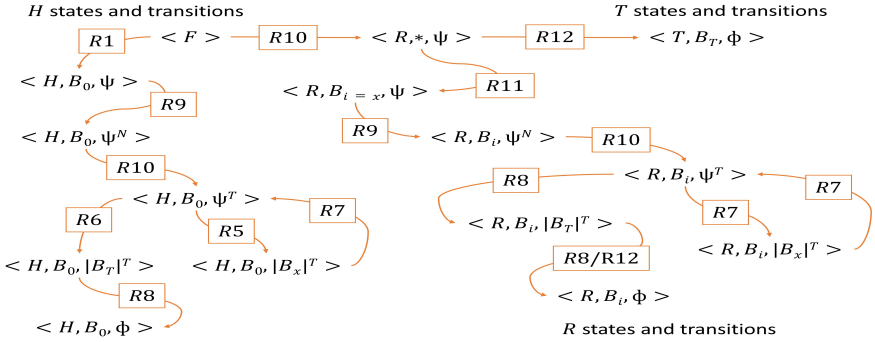


Figure 3 The new strand states and transitions.

Avoidance Games Are PSPACE-Complete

Valentin Gledel
Department of Mathematics and Mathematical Statistics, Umeå University, Sweden

Nacim Oijid
Univ. Lyon, Université Lyon 1, LIRIS UMR CNRS 5205, F-69621, Lyon, France

Abstract

Avoidance games are games in which two players claim vertices of a hypergraph and try to avoid some structures. These games have been studied since the introduction of the game of SIM in 1968, but only few complexity results have been found out about them. In 2001, Slany proved some partial results on Avoider-Avoider games complexity, and in 2017 Bonnet et al. proved that short Avoider-Enforcer games are Co-W[1]-hard. More recently, in 2022, Miltzow and Stojaković proved that these games are NP-hard. As these games correspond to the misère version of the well-known Maker-Breaker games, introduced in 1963 and proven PSPACE-complete in 1978, one could expect these games to be PSPACE-complete too, but the question has remained open since then. Here, we prove here that both Avoider-Avoider and Avoider-Enforcer conventions are PSPACE-complete. Using the PSPACE-hardness of Avoider-Enforcer, we provide in appendix proofs that some particular Avoider-Enforcer games also are.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Games, Avoider-Enforcer, Maker-Breaker, Complexity, Avoider-Avoider, PSPACE-complete

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.34

Funding *Valentin Gledel*: supported by the Kempe Foundation Grant No. JCK-2022.1 (Sweden).
Nacim Oijid: supported by the ANR project P-GASE (ANR-21-CE48-0001-01)

Acknowledgements We want to thank Eric Duchêne, Marianne Fortin, Aline Parreau and the anonymous referees for their help in the writing of this article.

1 Introduction

1.1 Related works

Avoidance games belong to the class of positional games, that were introduced by Hales and Jewett in 1963 [14] and popularized by Erdős and Selfridge in 1973 [11]. In this class of games, the board is a hypergraph and two players alternately claim a vertex of the hypergraph that has not been claimed before. Winning conditions depend on the convention and are related to the hyperedges. Tic-Tac-Toe and Hex are two famous examples of positional games. To learn more about positional games, we refer the reader to the recent survey of Hefetz et al. [18].

Among positional games, a natural dichotomy exists: on the one hand, there are games in which players seek to build a structure, and on the other hand, there are games in which players want to avoid a structure. The former set contains both Maker-Maker and Maker-Breaker conventions, in which the hyperedges are winning sets, and the player either wants to fill up a winning set (Maker role), or to play at least once in each of them (Breaker role). The latter contains Avoider-Avoider and Avoider-Enforcer conventions, that can be seen as the misère version of the former. In these games, the hyperedges are losing sets, and the players either want not to fill up one losing set (Avoider role), or to force their opponent to fill up one of them (Enforcer role).

© Valentin Gledel and Nacim Oijid;
licensed under Creative Commons License CC-BY 4.0

When positional games were introduced, the focus was on Maker-Breaker games, i.e. games in which one player, Maker, aims to fill up a hyperedge, and the second one, Breaker, wants to prevent it by claiming at least one vertex in each hyperedge. This convention is the most popular one, and several games have been studied according to this convention. In particular, the survey of Beck [5] presents several results obtained for Maker-Breaker games. The field of Maker-Breaker games is still well investigated today, and some Maker-Breaker games were introduced recently [10, 25].

The first Avoider-Avoider game was introduced in 1968 with the game of SIM and is presented in [28], but the first study of the complexity of Avoidance games was done by Schaefer in 1978 [27]. Avoider-Enforcer games were introduced later by Lu in 1991 [22, 23] under the name of Antimaker-Antibreaker games and correspond to the misère version of Maker-Breaker games. The standard name for this convention, Avoider-Enforcer, was popularized by Hefetz and different co-authors in 2007 [15, 16, 19, 20]. In this game, Enforcer wins if at some point during the game, Avoider has claimed all the vertices of a hyperedge, otherwise Avoider wins.

Even if most of the studies of positional games are focused on Maker-Breaker games, Avoider-Enforcer games have become more and more relevant: the famous Ramsey game was introduced in Avoider-Enforcer convention by Beck in 2002 [4] as a generalization of SIM. As it was done in the Maker-Breaker convention, some games on graphs were introduced in Avoider-Enforcer or Avoider-Avoider conventions, where the loosing sets correspond to some structure in the graph, see [2, 3, 13, 17].

In terms of complexity, an overview of the field is proposed by Demaine [9]. In positional games, as they are perfect information games, one player always has a winning strategy (or both players can ensure a draw). The natural decision problem related to games is therefore: does the first player have a winning strategy? This problem was quickly proven to be PSPACE-complete for Maker-Breaker games by Schaefer in 1978 [27] even restricted to 11-uniform hypergraphs (i.e. hypergraphs in which all hyperedges have size 11). This bound was recently improved by Rahman and Watson in 2021 [26], proving that the problem is still PSPACE-complete if the hypergraph is 6-uniform. These two proofs are very technical and a simpler proof of the PSPACE-completeness was provided by Byskov in 2004 [8], proving at the same time that Maker-Maker games are also PSPACE-complete. The complexity of Maker-Breaker games is still studied today, as Galliot et al. [12] have proven that the winner of a 3-uniform Maker-Breaker game can be computed in polynomial time, but the gap between the complexity of 6-uniform hypergraphs and 3-uniform hypergraphs remains to be closed.

Despite the fact that Avoidance games were introduced at the same time as Maker-Breaker games, only partial results on complexity are known: determining the winner in Avoider-Avoider games, was proven to be PSPACE-complete by Slany in 2002 [29] for endgames, i.e. games in which some vertices are already attributed to the players, but there are no results yet in the general case. Concerning Avoider-Enforcer, Bonnet et al. in 2017 [6] mentioned that the complexity of this problem is still open, when they proved that short games, i.e. games in which a player only has few moves to make, are co-W[1]-hard, with the number of moves taken as a parameter. The best known result today is due to Miltzow and Stojaković in 2022 [24] that states the NP-hardness of this decision problem and conjectures its PSPACE-completeness.

1.2 Presentation of the results

The Avoider-Enforcer game is played as follows: given a hypergraph H, two players, called *Avoider* and *Enforcer*, alternately claim an unclaimed vertex of H with Avoider starting. The game ends when all the vertices have been claimed. If Avoider has claimed all the vertices of a hyperedge, Enforcer wins. Otherwise, Avoider wins. The related decision problem is the following one.

▶ **Problem 1** (Avoider-Enforcer).
Input: A hypergraph H.

Output: True if and only if Avoider has a winning strategy in the Avoider-Enforcer game on H.

This paper will focus on the proof of the following result:

▶ **Theorem 2.** *The* Avoider-Enforcer *problem is* PSPACE*-complete, even when the entry is restricted to hypergraphs with hyperedges of size at most* 6*.*

Our proof of Theorem 2 follows a similar idea to the proof of Rahman and Watson [26] and the proof of Schaefer [27], by constructing some hyperedges forcing the order of the moves. Contrary to Maker-Breaker games, in Avoider-Enforcer convention, there is no vertex in which the players are urged to play, as in general, players do not want to move in avoidance games. The key idea of this reduction is to create some structures in which playing first is a losing move. In the provided construction, at any moment of the game, only few moves are not losing moves. Thus, we can control the vertices played by the two players.

The proof provided for PSPACE-completeness of Avoider-Enforcer games, enables us to state the following corollary for Avoider-Avoider games that will also be proven later:

▶ **Problem 3** (Avoider-Avoider).
Input: A hypergraph H.

Output: True if and only if the second player has a winning strategy in the Avoider-Avoider game on H.

▶ **Corollary 4.** *The* Avoider-Avoider *problem is* PSPACE*-complete, even when the entry is restricted to* 7*-uniform hypergraphs.*

This paper is organized as follows. In Section 2, we introduce two lemmas that will be used in the proof of Theorem 2. In particular, we show that pairing strategies that are often used in Maker-Breaker conventions can also be applied to Avoider-Enforcer games. Section 3 describes the reduction used to prove the PSPACE-completeness and define an order on the move that we call the *legitimate order*. We also show in this section that the proof holds if both players follow the legitimate order. In Section 4, we show that if a player does not follow the legitimate order then it cannot be a disadvantage to the other player, completing the proof of Theorem 2. Finally, in Section 5, we reduce the Avoider-Enforcer problem to 6-uniform hypergraphs and prove Corollary 4. One use of these results is to prove the PSPACE-completeness of particular Avoider-Enforcer games. In appendix we give two examples of such reductions with the cases of the Avoider-Enforcer domination game and the Avoider-Enforcer vertex H-game.

2 Preliminaries

In Maker-Breaker games, if a vertex is in all the hyperedges, it is always an optimal move for both players to play it. Here, we present a similar result for Avoider-Enforcer games. This result was proved by Miltzow and Stojaković [24], and intuitively states that if a vertex v is in all the hyperedges that contains another vertex u, then claiming v before u cannot benefit any player.

▶ **Lemma 5.** *Let H be a hypergraph, and u, v two vertices of H such that, for every hyperedge e containing u, e also contains v. If a player has a winning strategy, then this player has a winning strategy in which he never claims v while u is unclaimed.*

The second tool we introduce here is *pairing strategies* in Avoider-Enforcer games. In Maker-Breaker convention, these strategies are often described by using the fact that a player can claim at least one vertex in each pair of vertices. Here in Avoider-Enforcer convention, the main idea of pairing strategies is that this is always possible to force the opponent to claim at least one vertex in each pair.

In this section, we will refer to the players as Alice and Bob, as the strategy can be applied both by Avoider and by Enforcer.

▶ **Lemma 6.** *Let $H = (V, E)$ be a hypergraph. Suppose that Alice plays last in H, i.e. if the game is played until all the vertices have been claimed, Alice will claim the last one. Let $(a_1, b_1), \ldots, (a_n, b_n)$ be pairwise disjoint pairs of vertices, and let $v \notin \bigcup_{i=1}^{n} \{a_i, b_i\}$.*

Alice has a strategy which forces Bob to claim at least one vertex in each pair (a_i, b_i). Bob has a strategy which forces Alice to claim v and at least one vertex in each pair (a_i, b_i).

A strategy satisfying the hypothesis of Lemma 6 will be called a *pairing strategy*.

Proof. Consider the following strategy for Alice:

- If Bob claims a vertex in a pair (a_i, b_i), she claims the other vertex of the pair.
- Otherwise, she claims any vertex that is not in a pair.

By construction; when it is Bob's turn, in any pair in which he has played, Alice has also played. Therefore, when it is Alice's turn, there is at most one pair of vertices in which she has to play. As Alice plays the last move, whenever it is her turn to play, the number of remaining vertices is odd. Therefore, if Bob does not play in a pair, at least one vertex in no pair will be available for Alice. Thus, Alice has a strategy to force Bob to claim at least one vertex in each pair (a_i, b_i).

Now, consider the following strategy for Bob:

- If Alice claims a vertex in a pair (a_i, b_i), he claims the other vertex of the pair.
- If Alice claims v, if there exists at least one pair (a, b) in which Alice has not played, he claims a and he considers now that b is the new vertex that Alice will be forced to claim.
- Otherwise, he claims any vertex that is not in a pair nor v.

For the same reason, with this strategy, when it is Alice's turn, in any pair in which she has played, Bob has played too. When it is Bob's turn, note that the number of remaining moves is even, and there always exists exactly one vertex that is not in a pair, and that Bob wants Alice to claim. Thus, the number of vertices on which Bob cannot play before Alice is odd. Therefore, he always has an available move that fulfills this strategy. ◀

3 Proof of the main theorem

In this section, we begin the proof of Theorem 2 by describing the reduction from 3-QBF, introducing an order on the move of Avoider and Enforcer called the *legitimate order* and providing a sketch of the general proof.

▶ **Theorem 2.** *The* Avoider-Enforcer *problem is* PSPACE*-complete, even when the entry is restricted to hypergraphs with hyperedges of size at most* 6*.*

The first step of the proof is to prove that this game is in PSPACE.

▶ **Lemma 7.** *The* Avoider-Enforcer *problem is in* PSPACE*.*

Proof. Let $H = (V, E)$ be a hypergraph. As the players are not allowed to play an already claimed vertex, any game ends after at most $|V|$ moves. Therefore, according to Lemma 2.2 of Schaefer [27], as the game has a polynomial length and a polynomial number of moves, its winner can be computed with polynomial space. ◀

3.1 Construction of the hypergraph

We reduce the problem 3-QBF to an Avoider-Enforcer game. This problem has been proven PSPACE-complete by Stockemeyer and Meyer [30], and we use the gaming version of this problem as it was formulated by Rahman and Watson [26]. The game is played on a quantified formula φ of the form $\forall X_1 \exists X_2 \dots \forall X_{2n-1} \exists X_{2n} \psi$, with ψ a 3-SAT formula. Alternately, two players, namely Falsifier and Satisfier, chose valuation for the variables, Falsifier for the odd variables (quantified with a $\forall$) and Satisfier for the even ones (quantified with a $\exists$). When all the variables have a valuation Satisfier wins if ψ is satisfied, otherwise, Falsifier wins.

▶ **Problem 8** (3-QBF)**.**
Input: A 3-SAT quantified formula φ of the form $\forall X_1 \exists X_2 \dots \forall X_{2n-1} \exists X_{2n} \psi$.
Output: True if and only if Satisfier has a winning strategy in the 3-QBF game on φ

Given a 3-QBF formula of the form defined in Problem 8, we construct a hypergraph with $10n$ vertices $x_1, \overline{x_1}, \dots, x_{2n}, \overline{x_{2n}}, u_1, u_2, \dots, u_{6n}$.

A round in a 3-QBF formula corresponds to a step i during which Falsifier gives a valuation to X_{2i-1} and then Satisfier gives a valuation to X_{2i}. In this reduction, any round corresponds to ten vertices and eight hyperedges. Four of the ten vertices are $\{x_{2i-1}, \overline{x_{2i-1}}, x_{2i}, \overline{x_{2i}}\}$, and the six others are $u_{6i-5}, u_{6i-4}, u_{6i-3}, u_{6i-2}, u_{6i-1}, u_{6i}$. The eight hyperedges are constructed as follows:

$$\begin{aligned} A_{2i} &= (x_{2i}, \overline{x_{2i}}, u_{6i+1}, u_{6i+3}) & B_{2i-1} &= (x_{2i-1}, \overline{x_{2i-1}}, u_{6i-1}) \\ C^+_{6i} &= (u_{6i}, u_{6i+1}, u_{6i+3}, x_{2i}) & C^-_{6i} &= (u_{6i}, u_{6i+1}, u_{6i+3}, \overline{x_{2i}}) \\ C^+_{6i-2} &= (u_{6i-2}, u_{6i-1}, u_{6i+1}, x_{2i}) & C^-_{6i-2} &= (u_{6i-2}, u_{6i-1}, u_{6i+1}, \overline{x_{2i}}) \\ C^+_{6i-4} &= (u_{6i-4}, u_{6i-3}, u_{6i-1}, x_{2i-1}) & C^-_{6i-4} &= (u_{6i-4}, u_{6i-3}, u_{6i-1}, \overline{x_{2i-1}}) \end{aligned}$$

If some of these vertices do not exist, we still add the hyperedges, but with fewer vertices in them. For instance, $A_{2n} = \{x_{2n}, \overline{x_{2n}}\}$. Moreover, for each clause $F_j = l^j_1 \vee l^j_2 \vee l^j_3 \in \psi$ where the vertices l^j_1, l^j_2 and l^j_3 are literals either positive or negative, we add a hyperedge D_j. For $k = 1, 2, 3$, if l^j_k is a positive variable X_p, then x_p is in D_j, if l^j_k is a negative one $\neg X_p$, then $\overline{x_p}$ is in D_j. Moreover, If p is odd, then u_{6p-1} is in D_j, if p is even, then u_{6p+1} is in D_j.

Finally, the CNF game φ is reduced to the hypergraph $H = (V, E)$ with

$$V = \{\{x_i\}_{1\leq i\leq 2n} \cup \{\overline{x_i}\}_{1\leq i\leq 2n} \cup \{u_j\}_{1\leq j\leq 6n}\}$$

$$E = \left\{\{A_{2i}\}_{1\leq i\leq n} \cup \{C^+_{2i}\}_{1\leq i\leq 3n} \cup \{C^-_{2i}\}_{1\leq i\leq 3n} \cup \{B_{2i-1}\}_{1\leq i\leq n} \cup \{D_j\}_{1\leq j\leq m}\right\}$$

With this construction, we say that Avoider and Enforcer follow a *legitimate order* if they claim board elements in the following order for increasing i:

Legitimate order during round i

1. Avoider starts and claims u_{6i-5}.
2. Enforcer claims u_{6i-4}.
3. Avoider claims u_{6i-3}.
4. Enforcer claims one of x_{2i-1} or $\overline{x_{2i-1}}$.
5. Avoider claims the remaining vertex in $(x_{2i-1}, \overline{x_{2i-1}})$
6. Enforcer claims u_{6i-2}.
7. Avoider claims u_{6i-1}.
8. Enforcer claims u_{6i}.
9. Avoider claims one of x_{2i} or $\overline{x_{2i}}$.
10. Enforcer claims the remaining vertex in $(x_{2i}, \overline{x_{2i}})$.

3.2 Sketch of the proof

To prove that, with our construction, Avoider wins the Avoider-Enforcer game, if and only if Satisfier wins the QBF game, we first prove that this statement is true if the order of the moves is legitimate, as the moves will correspond to a valuation obtained in QBF. To force the players to play in the legitimate order, the main idea of the construction is that players want to claim some vertices as late as possible. Therefore, we prove that it is always optimal to respect the legitimate order of the moves. We introduce the following three lemmas that will be proved in the next section.

▶ **Lemma 9.** *When the game is restricted to the legitimate order, Avoider has a winning strategy in the Avoider-Enforcer game on H if and only if Satisfier has a winning strategy for the* 3-QBF *game on φ.*

▶ **Lemma 10.** *If Enforcer has a winning strategy in H when the legitimate order is respected by the two players, then he has a winning strategy in H.*

▶ **Lemma 11.** *If Avoider has a winning strategy in H when the legitimate order is respected by the two players, then she has a winning strategy in H.*

We first admit these lemmas and we prove Theorem 2.

Proof. First, according to Lemma 7, we know that Avoider-Enforcer is in PSPACE. We now prove the PSPACE-hardness of the problem by reduction from 3-QBF.

Let φ be a 3-SAT quantified boolean formula of the form described in Problem 8. Consider the hypergraph H obtained from φ by following the construction of Section 3.1. This construction has polynomial size. According to Lemma 9, when the order is respected, if Satisfier (Falsifier resp.) has a winning strategy in φ, Avoider (Enforcer resp.) has a winning strategy in H. Thus, according to Lemma 11 (Lemma 10 resp.), if Avoider (Enforcer resp.) has a winning strategy on H when the legitimate order is respected, she (he resp.) has one in general in H. Thus, Satisfier wins on φ if and only if, Avoider wins on H. Therefore, the Avoider-Enforcer problem is PSPACE-complete.

As all the construction provides a hypergraph H in which all the hyperedges have of size at most six, the Avoider-Enforcer problem is PSPACE-complete even restricted to hypergraphs in which all the hyperedges have size at most six. ◀

3.3 Game in legitimate order

In this section, we suppose that both players follow a legitimate order of moves.

If the order of moves is legitimate, the only choices available for Avoider and Enforcer are on the vertices x_i and $\overline{x_i}$. For each $1 \leq i \leq 2n$, Avoider claims one of $x_i, \overline{x_i}$ and Enforcer the other. Therefore, if both Avoider and Enforcer play one vertex in $\{x_i, \overline{x_i}\}$, we define the *underlying valuation* given to ψ as the following one:

$$X_i = \begin{cases} True & \text{if Avoider has claimed } \overline{x_i} \text{ and Enforcer has claimed } x_i \\ False & \text{if Avoider has claimed } x_i \text{ and Enforcer has claimed } \overline{x_i} \end{cases}$$

We now prove Lemma 9

Proof. Consider a game played on H for which both Avoider and Enforcer respected the legitimate order through the whole game.

▷ Claim 12. Avoider won the game on H if and only if the formula ψ is satisfied by the underlying valuation of the X_is.

Proof. Since the legitimate order is respected, Enforcer claimed all the vertices u_{2i} and thus played at least once in all the hyperedges C^+_{2i} and C^-_{2i}. Moreover, for each pair of variables $(x_i, \overline{x_i})$, Enforcer claimed one of the vertices of the pair, and so he has claimed at least one vertex in all the hyperedges A_i and B_i. Thus, the only hyperedges that could possibly be fully played by Avoider are the hyperedges D_j.

Since, in the legitimate order, Avoider claimed all the vertices u_{2i+1}, a hyperedge D_j corresponding to a clause F_j is fully played by Avoider if and only if she played on all the vertices $x(l^j_k)$ for $l_k \in F_j$, where $x(l^j_k) = x_p$ if $l^j_k = X_p$ and $x(l^j_k) = \overline{x_p}$ if $l^j_k = \neg X_p$. If this is the case, then this means that the formula ψ is not satisfied by the underlying valuation because the clause F_j has all its literals assigned to False. On the contrary, if the formula ψ is satisfied by the underlying valuation, then, for all clause F_j, at least one of the literals in it is assigned to True and so Enforcer played at least once in each hyperedge D_j.

Therefore, Avoider won the game on H if and only if ψ is satisfied. ◁

Suppose Satisfier has a winning strategy $\mathcal{S}$ on φ. We define a strategy for Avoider as follows: Whenever Avoider has to play a vertex x_{2k} or $\overline{x_{2k}}$, Avoider considers the underlying valuation given to the X_is with $i < 2k$. Then, if Satisfier had put X_{2k} to True, she claims $\overline{x_{2k}}$. Otherwise, she claims x_{2k}. With this strategy, at the end of the game, the underlying valuation of the variables of H will the same as the valuation given by the game that Satisfier played on φ. Since Satisfier has a winning strategy on φ, the underlying valuation satisfies ψ and so Avoider wins the game.

Similarly, if Falsifier has a winning strategy, Enforcer can follow the strategy in such a way that at the end the valuation of variables in the game played by Falsifier correspond to the underlying valuation in H. Since Falsifier wins on φ, Enforcer wins the game on H. ◀

4 Proofs of Lemma 10 and Lemma 11

The first part of our constructions showed that, if the legitimate order is respected, Avoider wins if and only if Satisfier wins the 3-QBF game. We now prove that, if a player has a winning strategy when the order is respected, he has one even if his opponent does not respect the order. We introduce here different sets of variables. These sets will be the main tools of the proofs of Lemma 10 and Lemma 11.

For $i = 1$ to $4n$, we define the set of vertices S_i as $S_{4n} = \{u_{6n}, x_{2n}, \overline{x_{2n}}\}$ and for $i < 4n$:

- if $i = 4k$, $S_i = \{u_{6k}, x_{2k}, \overline{x_{2k}}, u_{6k+1}\} \cup S_{i+1}$
- if $i = 4k-1$, $S_i = \{u_{6k-2}, u_{6k-1}\} \cup S_{i+1}$
- if $i = 4k-2$, $S_i = \{x_{2k-1}, \overline{x_{2k-1}}\} \cup S_{i+1}$
- if $i = 4k-3$, $S_i = \{u_{6k-4}, u_{6k-3}\} \cup S_{i+1}$

4.1 Proof of Lemma 10

We now prove Lemma 10

Proof. Suppose Enforcer has a winning strategy when the legitimate order is respected. Consider a strategy for Enforcer in which he plays according to the legitimate order until Avoider does not. If Avoider respects the order until all the vertices are played, by assumption, Enforcer wins. Otherwise, the proof of the following claim provides a winning strategy for Enforcer.

▷ Claim 13. If, during the game, Avoider plays in a set S_i in which Enforcer has not played yet, then, after this move, Enforcer has a strategy to win the game.

Proof. The proof is by induction on i.

First, notice that each S_i has an odd number of vertices and, as the total number of vertices in H is $10n$, there is also an odd number of vertices outside S_i. Therefore, if Avoider plays first in an S_i, Enforcer answers by playing an arbitrary vertex that is not in S_i and considers an arbitrary pairing outside S_i, which exists as there is an even number of vertices outside S_i after his move. This way, Avoider has to be the next player to play in S_i.

Base cases.

- Case $i = 4n$: If Avoider plays first is S_{4n}, by pairing the two other vertices in S_{4n}, by using Lemma 6, Enforcer can force Avoider to play another vertex in S_{4n}. Hence, as (u_{6n}, x_{2n}), $(u_{6n}, \overline{x_{2n}})$ and $(x_{2n}, \overline{x_{2n}})$ are three hyperedges, Avoider will claim the two vertices of one of them and thus lose.
- Case $i = 4n-1$: As shown previously, Enforcer has a strategy such that Avoider is the next player to play in S_{4n-1}. If Avoider has played at least one of her two first moves in S_{4n}, she has lost by the case $i = 4n$. Otherwise, she has claimed exactly u_{6n-2} and u_{6n-1}. In this case, Enforcer claims u_{6n} and pairs x_{2n} and $\overline{x_{2n}}$ and by Lemma 6 he forces Avoider to claim all the vertices of C^+_{6n-2} or C^-_{6n-2}.

Induction steps. Suppose that the first time Avoider does not respect the order of the move, she plays in a set S_i for $i \leq 4n-2$. If the second move of Avoider in S_i is in S_{i+1}, Enforcer wins by induction hypothesis. Thus, we can suppose that Avoider has claimed two vertices in $S_i \setminus S_{i+1}$. Moreover, as Enforcer has arbitrarily paired the vertices outside S_i, we describe here the strategy in S_i, and Enforcer plays according to the pairing outside S_i. This strategy ensures that the moves in S_i alternate between both players.

- Case $i = 4k$: Avoider has played twice in $\{u_{6k}, x_{2k}, \overline{x_{2k}}, u_{6k+1}\}$. At least one of $\{u_{6k}, x_{2k}, \overline{x_{2k}}\}$ is available. Enforcer claims it. Avoider has to claim the third vertex in this quadruple, otherwise, she plays first in S_{i+1} and loses by induction, and necessarily one of the three vertices she has claimed is u_{6k+1}. Enforcer claims u_{6k+2}. Avoider either plays first in S_{i+2} and loses by induction hypothesis, or claims u_{6k+3}. At this moment, Avoider has played on the vertices u_{6k+1} and u_{6k+3}, and two of the vertices of $\{u_{6k}, x_{2k}, \overline{x_{2k}}\}$. So she has completed one of the hyperedges $C^+_{6k} = (u_{6k}, u_{6k+1}, u_{6k+3}, x_{2k})$, $C^-_{6k} = (u_{6k}, u_{6k+1}, u_{6k+3}, \overline{x_{2k}})$ or $A_{2k} = (x_{2k}, \overline{x_{2k}}, u_{6k+1}, u_{6k+3})$.
- Case $i = 4k - 1$: Avoider has claimed u_{6k-2} and u_{6k-1}. Enforcer claims u_{6k}. Avoider has to play on vertex in $\{x_{2k}, \overline{x_{2k}}, u_{6k+1}\}$, (otherwise she plays first in S_{i+2} and loses by induction). Enforcer claims either x_{2k} or $\overline{x_{2k}}$, as at least one of them is available. If Avoider plays a vertex in S_{i+2} she loses by induction. So she has to play the last vertex available in $S_i \setminus S_{i+1}$. With this strategy, Avoider has necessarily claimed u_{6k+1} and one of x_{2k} and $\overline{x_{2k}}$. Thus, she has played all the vertices of either $C^+_{6k-2} = (u_{6k-2}, u_{6k-1}, u_{6k+1}, x_{2k})$ or $C^-_{6k-2} = (u_{6k-2}, u_{6k-1}, u_{6k+1}, \overline{x_{2k}})$.
- Case $i = 4k - 2$: Avoider has claimed x_{2k-1} and $\overline{x_{2k-1}}$. Enforcer claims u_{6k-2}. Either Avoider plays first in S_{i+2} and loses by induction, or she claims u_{6k-1}, the last available vertex in S_{i+1} and loses by having played all the vertices in $B_{2k-1} = (x_{2k-1}, \overline{x_{2k-1}}, u_{6k-1})$.
- Case $i = 4k - 3$: Avoider has claimed u_{6k-4} and u_{6k-3}. Enforcer claims x_{2k-1}. Avoider has to claim $\overline{x_{2k-1}}$, otherwise she plays first in S_{i+2} and loses by induction. Then Enforcer claims u_{6k-2}. If Avoider plays in S_{i+3} she loses by induction. The last vertex available in $S_i \setminus S_{i+3}$ is u_{6k-1}, and if Avoider claims it, she loses by playing all the vertices $C^-_{6k-4} = (u_{6k-4}, u_{6k-3}, u_{6k-1}, \overline{x_{2k-1}})$.

By applying this induction, at any moment of the game, if Avoider plays first in a set S_i, she looses. ◁

Finally, if Enforcer has played according to the legitimate order, at any moment of the game, Avoider has to play in a set S_i in which Enforcer has already played. Therefore, she has to respect the order of the moves. The only moment when she can change this order is by claiming u_{6k+1} instead of one of the vertices $x_{2k}, \overline{x_{2k}}$. But if she does so, Enforcer can claim one of them, for instance x_{2k}, and Avoider will be forced to claim $\overline{x_{2k}}$. If this happens, everything happens as if Avoider has claimed $\overline{x_{2k}}$ first and u_{6k+1} after. Since these moves could have occurred in the legitimate order, the strategy can then continue as if the order has been respected.

To conclude, if Enforcer has a winning strategy when the legitimate order is respected, Enforcer has a winning strategy in H even without this restriction. ◀

4.2 Proof of Lemma 11

In this section, we prove that if Avoider has a winning strategy when the legitimate order is respected, she also has one if Enforcer does not respect the order. The main idea of the strategy is to respect the order, and if Enforcer does not respect the order, Avoider has a pairing strategy to force Enforcer to claim a vertex in some pair $(x_i, \overline{x_i})$, or the odd u_j that follows them. By construction, any hyperedge containing a vertex x_i or $\overline{x_i}$ also contains the next odd vertex u_j in the legitimate order, and this will prove that whenever Enforcer does not respect the order, it benefits Avoider. We now will prove Lemma 11.

Proof. Suppose Avoider has a winning strategy when the legitimate order is respected. We now describe a winning strategy for Avoider in the general case.

While Enforcer respects the legitimate order, Avoider also respects it. Suppose that at some moment of the game, Enforcer does not respect the legitimate order. Denote by y_A the vertex he would have played according to the legitimate order, and by y_E the vertex he has claimed instead. If y_A is a vertex x_{2i} or $\overline{x_{2i}}$, Avoider pairs it with u_{6i+1} and continues as if Enforcer had to play u_{6i+2}. If this is the case, consider $y_A = u_{6i+2}$. Note that, according to Lemma 5, we can suppose that y_E is not a vertex u_j with j odd. Indeed, for each vertex u_{2i+1}, the hyperedges that contain the previous vertex in the legitimate order (if this vertex is a vertex x_j or $\overline{x_j}$ this is true for either of them) also contain u_{2i+1}. As any hyperedge containing x_{2i} or $\overline{x_{2i}}$ also contains u_{6i+1} which is the next vertex Avoider should have played according to the legitimate order, it benefits her if Enforcer finally claims u_{6i+1} instead of x_{2i} or $\overline{x_{2i}}$ according to Lemma 5.

Denote by k the smallest integer such that $y_E \notin S_k$, and by k' the largest integer such that $y_A \in S_{k'}$. We consider $k = 4n + 1$ if $y_E \in S_{4n}$, with $S_{4n+1} = \emptyset$. Note that all the vertices outside $S_{k'}$ have already been played or are paired, and that $S_{k'} \setminus S_k$ is then the set of vertices perturbed by the move of Enforcer. As all the sets S_i have an odd number of vertices, we know that the number of remaining vertices outside S_k is odd, as an even number of moves have been played in it. By assumption, Avoider was following a winning strategy for the legitimate order. She can then consider an arbitrary sequence of moves for Enforcer following the legitimate order and her answers according to her strategy until all the vertices in $S_{k'} \setminus S_k$ are played. According to these moves, we will denote by x_j^E the vertex among $(x_j, \overline{x_j})$ claimed by Enforcer and by x_j^A the vertex played by Avoider.

Avoider claims y_A, the vertex that Enforcer should have claimed according to the legitimate order, and will consider one strategy in S_k and another one outside S_k:

- In $H \setminus S_k$, she plays according to a pairing strategy, that is presented in the next paragraph.
- In S_k, Avoider considers the strategy she would have played if all the vertices outside S_k were played according to the legitimate order, with the vertices x_j^E claimed by Enforcer and the vertices x_j^A claimed by Avoider.

The pairing we define is the following one: (u_{6i-4}, x_{2i-1}^A), (u_{6i-3}, u_{6i-6}), (x_{2i-1}^E, u_{6i-1}), (u_{6i-2}, x_{2i}^A), (u_{6i+1}, x_{2i}^E). This pairing concerns all the vertices that have to be played after y_A in the legitimate order that are not in S_k, and we consider only pairs containing at least one vertex outside S_k. Note that, by construction, exactly one vertex of this pairing is already played, and exactly one paired with a vertex in S_k. Therefore, to make the pairing contain only vertices not played and outside S_k, some modifications are done. These modifications are presented in Figure 1. By applying Lemma 6, Avoider can ensure that Enforcer plays at least one in each of these pairs.

y_A	changes
u_{6i-4}	$u_{6i-3} \longleftrightarrow x_{2i-1}^A$
x_{2i-1} or $\overline{x_{2i-1}}$	$x_{2i-1}^* \longleftrightarrow u_{6i-1}$
u_{6i-2}	$u_{6i-1} \longleftrightarrow x_{2i}^A$
u_{6i}	$u_{6i+3} \longleftrightarrow x_{2i}^A$

y_E	changes
u_{6i-4}	no changes
x_{2i-1} or $\overline{x_{2i-1}}$	$x_{2i-1}^* \longleftrightarrow u_{6i-4}$
u_{6i-2}	no changes
u_{6i}	no changes
x_{2i} or $\overline{x_{2i}}$	$x_{2i}^* \longleftrightarrow u_{6i+1}$, $u_{6i-2} \longleftrightarrow u_{6i}$

Figure 1 Changes of the matching. x_j^* refers to the variable in $\{x_k, \overline{x}_k\}$ that has not been played, arrows represent new pairs of vertices in the matching.

▷ Claim 14. The pairing strategy ensures that Enforcer plays at least once in each hyperedge A_i, B_i or C_i containing all their vertices in $S_{k'}$ and at least one outside S_k.

Proof. First, if the hyperedge contains no vertex whose pairing has been modified because of their belonging to y_A or y_E, it contains two paired vertices. We show in bold text the paired vertices:

$$\begin{aligned} A_{2i} &= (x^A_{2i}, \boldsymbol{x^E_{2i}}, \boldsymbol{u_{6i+1}}, u_{6i+3}) & C^E_{6i-2} &= (u_{6i-2}, u_{6i-1}, \boldsymbol{u_{6i+1}}, \boldsymbol{x^E_{2i}}) \\ C^A_{6i} &= (\boldsymbol{u_{6i}}, u_{6i+1}, \boldsymbol{u_{6i+3}}, x^A_{2i}) & B_{2i-1} &= (x^A_{2i-1}, \boldsymbol{x^E_{2i-1}}, \boldsymbol{u_{6i-1}}) \\ C^E_{6i} &= (u_{6i}, \boldsymbol{u_{6i+1}}, u_{6i+3}, \boldsymbol{x^E_{2i}}) & C^A_{6i-4} &= (\boldsymbol{u_{6i-4}}, u_{6i-3}, u_{6i-1}, \boldsymbol{x^A_{2i-1}}) \\ C^A_{6i-2} &= (\boldsymbol{u_{6i-2}}, u_{6i-1}, u_{6i+1}, \boldsymbol{x^A_{2i}}) & C^E_{6i-4} &= (u_{6i-4}, u_{6i-3}, \boldsymbol{u_{6i-1}}, \boldsymbol{x^E_{2i-1}}) \end{aligned}$$

For the first hyperedges of the matching, there are two paired vertices.

Recall first that if Enforcer was supposed to play in $\{x_{2i}, \overline{x_{2i}}\}$, Avoider pairs this vertex with u_{6i+1} and considers $y_A = u_{6i+2}$. The only one hyperedge among the A_i, B_i and C_i that was concerned with this change is A_{2i}, in which two vertices are now paired. In any other case, the following vertices are paired:

- If $y_A = u_{6i-4}$, only the hyperedges C^E_{6i-4} and C^A_{6i-4} are concerned by the changes. In the former x^E_{2i-1} is paired with u_{6i-1}, in the latter x^A_{2i-1} is paired with u_{6i-3}.
- If $y_A \in \{x_{2i-1}, \overline{x_{2i-1}}\}$, the only hyperedges concerned by the change is B_{2i-1}. In it, the other vertex in $\{x_{2i-1}, \overline{x_{2i-1}}\}$ is paired with u_{6i-1}
- If $y_A = u_{6i-2}$, only the hyperedges C^E_{6i-2} and C^A_{6i-2} are concerned by the changes. In the former x^E_{2i} is paired with u_{6i+1}, in the latter x^A_{2i} is paired with u_{6i-1}.
- If $y_A = u_{6i}$, only the hyperedges C^E_{6i} and C^A_{6i} are concerned by the changes. In the former x^E_{2i} is paired with u_{6i+1}, in the latter x^A_{2i} is paired with u_{6i+3}.

For the last hyperedges that contain vertices of the matching, the following happens:

- If $y_E = u_{6i-4}$, the pairing stops at u_{6i-3}. The only two hyperedges that contain at least one vertex in S_k and one vertex outside S_k are C^+_{6i-4} and C^-_{6i-4}, in which Enforcer has claimed y_E.
- If $y_E = x_{2i-1}$ or $\overline{x_{2i-1}}$, the pairing stops after the second vertex in $\{x_{2i-1}, \overline{x_{2i-1}}\}$. The only one hyperedge containing at least one vertex in S_k and one outside S_k is B_{2i-1}, in which Enforcer has already claimed y_E.
- If $y_E = u_{6i-2}$, the pairing stops at u_{6i-1}. The only two hyperedges that contain at least one vertex in S_k and one vertex outside S_k are C^+_{6i-2} and C^-_{6i-2}, in which Enforcer has claimed y_E.
- If $y_E = u_{6i}$, the pairing stops at u_{6i+1}. The three hyperedges that contain both vertices in S_k and vertices outside S_k are C^+_{6i}, C^-_{6i} and A_{2i}. In C^+_{6i}, C^-_{6i}, Enforcer has claimed y_E, and in A_{2i}, Enforcer will play one of x^E_{2i} or u_{6i+1} as these two vertices are paired together.
- If $y_E = x_{2i}$ or $\overline{x_{2i}}$, the pairing stops at u_{6i+1}. The three hyperedges that contain vertices inside S_k and outside S_k are C^+_{6i}, C^-_{6i} and A_{2i}. As the second vertex in $\{x_{2i}, \overline{x_{2i}}\}$ is paired with u_{6i+1}, either Enforcer has claimed both x_{2i} and $\overline{x_{2i}}$, and any of these three hyperedges contains at least one of them; or Enforcer has claimed u_{6i+1} which is in these three hyperedges.

If the pairing stops because it goes until the end (i.e. $k = 4n + 1$), one vertex is not paired. According to Lemma 6, as Enforcer plays the last move in H, Avoider can force him to play it and still play once in each pair of the pairing.

Finally, in any hyperedge A_i, B_i or C_i containing at least one vertex of the matching, Enforcer has played at least one vertex. ◁

Now, we can prove that the strategy we defined for Avoider is a winning strategy. In all the hyperedges A_i, B_i or C_i, Enforcer played at least once. Indeed, if Enforcer has respected the order until he plays in one of these hyperedges, there is nothing to do. Otherwise, by Claim 14, Avoider can force Enforcer to play in it as this hyperedge is considered in a set of hyperedges in which Enforcer has not respected the order, and thus contains two paired vertices.

In the hyperedges D_j, as the strategy in the legitimate order is a winning strategy, it forces at least one vertex x_i or $\overline{x_i}$ to be claimed by Enforcer in D_j (since all the vertices u_k of odd indices are played by Avoider in the legitimate order). Then, by construction, if when this vertex has to be claimed the order was respected, Enforcer has played it. If the order was not respected, then Avoider has paired this vertex with the next vertex u_k of odd index, forcing Enforcer to play one of them. In both cases, Enforcer has played in D_j. ◀

5 Uniform hypergraphs and applications

The construction provided in Section 3.1 provided a hypergraph in which any hyperedge have size at most six. We prove here that the PSPACE-hardness can be generalized to uniform hypergraphs.

5.1 From 6-hypergraphs to 6-uniform hypergraphs

A hypergraph $H = (V, E)$ is a *k-hypergraph* if any hyperedge $e \in E$ has size at most k. It is said to be *k-uniform* if any hyperedge $e \in E$ has size exactly k.

▶ **Lemma 15.** *Let $H = (V, E)$ be a k-hypergraph. Let $m = \min_{e \in E} |e|$. If $m < k$, there exists a k-hypergraph $H' = (V', E')$ where $\min_{e \in E} |e| = m + 1$, having $|E'| \leq 2|E|$ and $|V'| \leq |V| + 2$ such that Avoider has a winning strategy in the Avoider-Enforcer game on H if and only if she has one in H'.*

Proof. Let $H = (V, E)$ be a k-hypergraph. Let $m = \min_{e \in E} |e|$. We define $H' = (V', E')$ as follows. We start from $V' = V$. We add two vertices $\{a_1, a_2\}$ in V'. For each hyperedge $e \in E$, we add hyperedges in E' as follows:

- If $|e| > m$, we add a copy of e in E'.
- If $|e| = m$, we add two hyperedges $e_1 = e \cup \{a_1\}$ and $e_2 = e \cup \{a_2\}$ in E'.

We have $|V'| = |V| + 2$, $|E'| \leq 2|E|$ and $\min_{e \in E'} |e| = m + 1$.

Now, if Avoider (Enforcer resp.) had a winning strategy $\mathcal{S}$ in E, we can define a strategy $\mathcal{S}'$ in E' as follows:

- If the opponent plays a vertex in V, or if it is the first move of the player, play as in $\mathcal{S}$.
- If the opponent plays a vertex in $\{a_1, a_2\}$, or if there is no vertex in V available, play an available vertex in $\{a_1, a_2\}$.

Following this strategy, Avoider (Enforcer resp.) has played exactly the same vertices in H' as he (she resp.) would have played in H according to $\mathcal{S}$ with, in addition, exactly one of $\{a_1, a_2\}$.

Therefore, if Avoider had a winning strategy in H, then for each $e \in E$, there exists one vertex $v \in e$, that Enforcer has played. Let $e' \in E'$ be a hyperedge. If e' is a copy of some hyperedge $e \in E$, then Enforcer has played in it, as he would have played in e according to $\mathcal{S}$. If e' is a hyperedge e_1 or e_2 created from a hyperedge $e \in E$, as Enforcer would have played in e according to $\mathcal{S}$, he has played the same vertex in e'.

If Enforcer had a winning strategy in H, following this strategy, there exists a hyperedge $e \in E$ in which Avoider has claimed all the vertices. If $|e| \geq m+1$, Avoider has also played all the vertices of $e \in E'$, so Enforcer has won. If $|e| = m$, as the strategy $\mathcal{S}'$ forces Avoider to play at least one of $\{a_1, a_2\}$, suppose without loss of generality that she has claimed a_1. Then, she has claimed a_1 and all the vertices of e, so she has filled up the edge e_1. Therefore, this strategy is a winning strategy for Enforcer.

Finally, H' has the same outcome as H and $\min_{e \in E'} |e| = m+1$. ◀

▶ **Corollary 16.** *Avoider-Enforcer is* PSPACE*-complete even restricted to* 6*-uniform hypergraphs*

Proof. The construction provided in the proof of Theorem 2 builds a 6-hypergraphs in which all hyperedges have size at least two. Therefore, by applying Lemma 15 four times, with $m = 2, 3, 4, 5$, we obtain a hypergraph having at most eight more vertices and eight times more hyperedges. Thus, this construction is still polynomial and the hypergraph obtained is 6-uniform. ◀

5.2 Avoider-Avoider games

We prove Corollary 4, i.e. that Avoider-Avoider games are PSPACE-complete, even restricted to 7-uniform hypergraphs.

Proof. Consider the construction provided in the proof of Corollary 16. Consider H' the hypergraph obtained by adding a vertex v_0 in H and adding it in all the hyperedges of H. Note that, as any hyperedge of H has size six, any hyperedge of H' have size exactly seven. According to Lemma 5 (note that the result also apply to Avoider-Avoider games), both players have an optimal strategy in which v_0 will be played last, and as the graph has an odd number of vertices, the first player will play it. Therefore, the second player cannot fill up a hyperedge and plays as Enforcer would in the Avoider-Enforcer game. By applying the same strategy as in Avoider-Enforcer, if Avoider wins in Avoider-Enforcer, the game ends by a draw, otherwise the second player wins. ◀

References

1 Noga Alon, József Balogh, Béla Bollobás, and Tamás Szabó. Game domination number. *Discrete mathematics*, 256(1-2):23–33, 2002.

2 V Anuradha, Chinmay Jain, Jack Snoeyink, and Tibor Szabó. How long can a graph be kept planar? *the electronic journal of combinatorics*, 15(1), 2008.

3 János Barát and Miloš Stojaković. On winning fast in Avoider-Enforcer games. *the electronic journal of combinatorics*, 17, 2008.

4 József Beck. Ramsey games. *Discrete mathematics*, 249(1-3):3–30, 2002.

5 József Beck. *Combinatorial games: tic-tac-toe theory*, volume 114. Cambridge University Press Cambridge, 2008.

6 Edouard Bonnet, Serge Gaspers, Antonin Lambilliotte, Stefan Rümmele, and Abdallah Saffidine. The Parameterized Complexity of Positional Games. In *ICALP*, Varsovie, Poland, July 2017.

7 Boštjan Brešar, Sandi Klavžar, and Douglas F Rall. Domination game and an imagination strategy. *SIAM Journal on Discrete Mathematics*, 24(3):979–991, 2010.

8 Jesper Makholm Byskov. Maker-Maker and Maker-Breaker games are PSPACE-complete. *BRICS Report Series*, 11(14), 2004.

9 Erik D Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 18–33. Springer, 2001.

10 Eric Duchene, Valentin Gledel, Aline Parreau, and Gabriel Renault. Maker-Breaker domination game. *Discrete Mathematics*, 343(9):111955, 2020.

11 P Erdös and J.L Selfridge. On a combinatorial game. *Journal of Combinatorial Theory, Series A*, 14(3):298–301, 1973.

12 Florian Galliot, Sylvain Gravier, and Isabelle Sivignon. Maker-Breaker is solved in polynomial time on hypergraphs of rank 3, 2022.

13 Andrzej Grzesik, Mirjana Mikalački, Zoltán Lóránt Nagy, Alon Naor, Balázs Patkós, and Fiona Skerman. Avoider-Enforcer star games. In *The Seventh European Conference on Combinatorics, Graph Theory and Applications*, pages 375–379. Springer, 2013.

14 R.I. Hales and A.W. Jewett. Regularity and positional games. *Trans. Am. Math. Soc*, 106:222–229, 1963.

15 Dan Hefetz. *Positional games on graphs*. PhD thesis, Citeseer, 2007.

16 Dan Hefetz, Michael Krivelevich, Miloš Stojaković, and Tibor Szabó. Fast winning strategies in positional games. *Electronic Notes in Discrete Mathematics*, 29:213–217, 2007.

17 Dan Hefetz, Michael Krivelevich, Miloš Stojaković, and Tibor Szabó. Avoider-Enforcer: The rules of the game. *Journal of Combinatorial Theory, Series A*, 117(2):152–163, 2010.

18 Dan Hefetz, Michael Krivelevich, Miloš Stojaković, and Tibor Szabó. *Positional games*, volume 44. Springer, 2014.

19 Dan Hefetz, Michael Krivelevich, and Tibor Szabó. Avoider-Enforcer games. *Journal of Combinatorial Theory, Series A*, 114(5):840–853, 2007.

20 Dan Hefetz, Michael Krivelevich, and Tibor Szabó. Bart–Moe games, JumbleG and discrepancy. *European Journal of Combinatorics*, 28(4):1131–1143, 2007.

21 Gal Kronenberg, Adva Mond, and Alon Naor. h-games played on vertex sets of random graphs. *arXiv preprint*, 2019. `arXiv:1901.00351`.

22 Xiaoyun Lu. A matching game. *Discret. Math.*, 94(3):199–207, 1991. `doi:10.1016/0012-365X(91)90025-W`.

23 Xiaoyun Lu. A note on biased and non-biased games. *Discrete Applied Mathematics*, 60(1):285–291, 1995.

24 Tillmann Miltzow and Miloš Stojaković. Avoider-Enforcer Game is NP-hard. *arXiv preprint*, 2022. `arXiv:2208.06687`.

25 Rajko Nenadov, Angelika Steger, and Miloš Stojaković. On the threshold for the Maker-Breaker H-game. *Random Structures & Algorithms*, 49(3):558–578, 2016.

26 Md Lutfar Rahman and Thomas Watson. 6-Uniform Maker-Breaker Game Is PSPACE-Complete. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 57:1–57:15, 2021.

27 Thomas J. Schaefer. On the Complexity of Some Two-Person Perfect-Information Games. *Journal of computer and system Sciences*, 16:185–225, 1978.

28 Gustavus J Simmons. The Game of SIM. In *Mathematical Solitaires & Games*, pages 50–50. Routledge, 1968.

29 Wolfgang Slany. Endgame problems of Sim-like graph Ramsey avoidance games are PSPACE-complete. *Theoretical computer science*, 289(1):829–843, 2002.

30 Larry J Stockmeyer and Albert R Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9, 1973.

A Particular Avoider-Enforcer games

Several games have been proven to be PSPACE-complete in the Maker-Breaker convention, thanks to the proofs of Schaefer or of Rahman and Watson. Due to the similarities between the two conventions, some reductions may be adapted to prove that these games are PSPACE-complete in the Avoider-Enforcer convention. In particular, we prove in this section that the domination game and the vertex H-Game are PSPACE-complete in Avoider-Enforcer convention.

A.1 Avoider-Enforcer domination game

The Maker-Breaker domination game was introduced by Duchêne et al. in 2020 [10] and follows the study of domination games on graphs, which were investigated since 2002 [1, 7]. In Maker-Breaker, two players, namely Dominator and Staller alternately claim an unclaimed vertex of the graph, Dominator wins if he manages to claim all the vertices from a dominating set. Otherwise, Staller wins. They proved that determining whether Dominator or Staller has a winning strategy is PSPACE-complete using a reduction from the general Maker-Breaker game. The Avoider-Enforcer domination game can be similarly defined, with Anti-Staller winning if Anti-Dominator claims a dominating set and Anti-Dominator winning otherwise. We prove here that determining the winner of the Avoider-Enforcer domination game is PSPACE-complete. Note that the proof is very similar to the reduction from Maker-Breaker games to Maker-Breaker domination game.

▶ **Problem 17** (Avoider-Enforcer domination game)**.**
Input: A graph G
Output: True if and only if Anti-Dominator wins the Avoider-Enforcer domination game on G.

▶ **Theorem 18.** *The* Avoider-Enforcer domination game *problem is* PSPACE*-complete*

Proof. First, Avoider-Enforcer domination game is in PSPACE, as the number of moves in a game is the number of vertices, and as determining if a set is a dominating set or not can be done in polynomial time, the game is in PSPACE.

Let $H = (V_H, E_H)$ be a hypergraph. Without loss of a generality, we can suppose that each vertex is in at least one hyperedge, otherwise the vertices in no hyperedge will be played first according to Lemma 5, and we can just remove them, up to change the first player to go. We construct the following graph $G = (V, E)$ as follows:

- For each vertex u_i in V_H, we add a vertex v_i in V.
- For each hyperedge C in E_H, we add two vertices v_C^1 and v_C^2 in V.
- If a vertex u_i of V_H belongs to a hyperedge C of E_H, we add the edges $v_i v_C^1$ and $v_i v_C^2$ to E.

Note that the graph created here is bipartite

Suppose Avoider (Enforcer resp.) has a winning strategy $\mathcal{S}$ in H. We define a strategy $\mathcal{S}'$ for Anti-Staller (Anti-Dominator resp.) in G as follows.

- If Avoider (Enforcer resp.) plays the first move in H, he claims first a vertex v_i such that u_i is the first vertex claimed in $\mathcal{S}$.
- If the opponent claims a vertex v_i, he claims a vertex v_j such that u_j is the answer to the vertex u_i in $\mathcal{S}$.

- If a player plays a vertex v_C^k for $k \in \{1,2\}$, he claims the vertex $v_C^{k'}$ for $k' \neq k \in \{1,2\}$.

Now if Avoider had a winning strategy in H, by applying the strategy $\mathcal{S}'$, for any vertex v_C^i, Anti-Staller has not played all the v_js adjacent to it. Therefore, Anti-Dominator has claimed one of them and all the v_C^is are dominated. Moreover, all the vertex u_js are in at least one edge C of H, and thus v_j is dominated by Anti-Dominator, as she has played one of (v_C^1, v_C^2). Finally, Anti-Dominator has dominated the graph and Anti-Staller has won.

Reciprocally, if Enforcer had a winning strategy in H, by applying the strategy $\mathcal{S}'$, Anti-Dominator knows that there exists a pair of vertices (v_C^1, v_C^2), such that Anti-Staller has played all the v_js adjacent to them. As Anti-Staller has played one of them, and all its neighbors, this vertex is not dominated. Therefore, Anti-Dominator has won. ◀

In Figure 2 we provide an example of reduction. Note that the construction provided is exactly the same as the one used to prove the PSPACE-completeness in Maker-Breaker in [10].

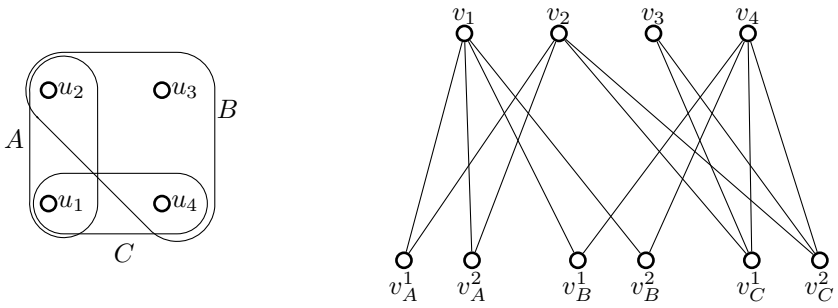


Figure 2 Reduction from Avoider-Enforcer game to Avoider-Enforcer domination game.

▶ Remark 19. Up to add all the edges between the v_is, the game is still PSPACE-complete on split graphs.

A.2 Avoider-Enforcer vertex H-Game

The vertex H-Game has been introduced by Kronenberg, Mond and Naor in [21] on random graphs. It is presented in several conventions, but we will focus here on the Avoider-Enforcer one. The game is played as follows:

Let H be a graph. Avoider and Enforcer play on the vertex set of another graph G. Alternately, Avoider and Enforcer claim an unclaimed vertex of G. Avoider wins if the set of vertices she has claimed do not contain H as a subgraph (not necessarily induced). Otherwise, Enforcer wins.

▶ **Problem 20** (Avoider-Enforcer vertex H-Game)**.**
Input: A graph G
Output: True if and only if Avoider wins the Avoider-Enforcer vertex H-Game played on G.

We first need to define some graphs and operations that will be helpful to describe the graphs of this section:

- We will denote by I_k the graph being an independent set of size k, i.e. containing k vertices and no edge.

- If G and H are two graphs, we denote by $G \bowtie H$ their join, i.e. if $G = (V_G, E_G)$ and $H = (V_H, E_H)$, we have $G \bowtie H = (V, E)$ with $V = V_G \cup V_H$ and $E = E_G \cup E_H \cup \{(v_G, v_H) | v_G \in V_G, v_H \in V_H\}$.
- If G and H are two graphs, we denote by $G \boxtimes H$ their strong product, i.e. if $G = (V_G, E_G)$ and $H = (V_H, E_H)$, we have $G \boxtimes H = (V, E)$ with $V = \{x_{u,v} | u \in V_G, v \in V_H\}$ and $E = \{(x_{u_1,v_1}, x_{u_2,v_2}) | (u_1 = u_2 \text{ or } (u_1, u_2) \in E_G) \text{ and } (v_1 = v_2 \text{ or } (v_1, v_2) \in E_H)\}$.

Remark that for any graph G, $G \boxtimes P_2$ (where P_2 design the path of length 2) is obtained by taking two copies of G and connecting each vertex to its copy and its copy's neighbors.

We prove here that determining the winner of the Avoider-Enforcer vertex H-Game is a PSPACE-complete problem for several graphs H .

▶ **Theorem 21.** *Let H_0 be a graph containing at least one edge or at least 6 vertices, and let $k \geq 6$. Consider $H = I_k \bowtie H_0$. The* Avoider-Enforcer vertex H-game *problem is* PSPACE*-complete.*

Note that complete bipartite graphs $K_{n,m}$, with $n, m \geq 6$, are of this type. Indeed, $K_{n,m} = I_k \bowtie H_0$ for $H_0 = I_m$ and $k = n$.

Proof. First, the Avoider-Enforcer vertex H-game is in PSPACE. Indeed, as it is a positional game, if $G = (V, E)$ is a graph, the game ends after at most $|V|$ moves. After that, determining whether a graph H is a subgraph of a graph G can be done in polynomial space.

We do our reduction from Avoider-Enforcer on 6-uniform hypergraphs (proven to be PSPACE-complete in Corollary 16). Let H_0 be a graph containing at least one edge or at least 6 vertices, and let $k \geq 6$. Let $H = I_k \bowtie H_0$. To avoid confusion while describing the strategies in the two games, we will call the players of the Avoider-Enforcer vertex H-Game Alice and Bob, with Alice avoiding creating a subgraph H and Bob forcing her to create one.

Let $H' = (V', E')$ be a 6-uniform hypergraph. Let H'_0 be the strong product $H_0 \boxtimes P_2$. We build $G = (V, E)$ an instance of the Avoider-Enforcer vertex H-Game as follows:

- **Step 1**: For any vertex $v'_i \in V'$, we add a vertex $v_i \in V$.
- **Step 2**: For any hyperedge $C \in E'$, we add $2(k-6)$ vertices $v^C_1, \ldots, v^C_{2(k-6)}$ (note that if $k = 6$, no vertex is added during this step).
- **Step 3**: For any hyperedge $C \in E'$, we add a copy H^C_0 of the graph H'_0 in G, and we connect any vertex of H^C_0 to all the vertices v_i such that $v'_i \in C$ and to all the vertices v^C_j for $1 \leq j \leq 2k-6$.

▷ Claim 22. If Avoider has a winning strategy in H', Alice has a winning strategy in G.

Proof. Suppose Avoider has a winning strategy $\mathcal{S}'$ in H'. We define Alice's strategy $\mathcal{S}$ in G as follows:

- She starts by claiming the vertex v_i, corresponding to the vertex v'_i that Avoider would have claimed in H' according to $\mathcal{S}'$.
- If Bob claims a vertex v_i, she answers with the vertex v_j corresponding to the vertex v'_j that Avoider would have claimed by $\mathcal{S}'$ in H' if Enforcer has claimed v'_i.
- In H^C_0, as it is a strong product $H_0 \boxtimes P_2$, Alice considers the pairing between any vertex of H_0 and its copy in the strong product. If Bob plays in one pair of this pairing, she claims the second vertex of this pair.
- For any hyperedge C in H', Alice considers the set of vertices v^C_j for $1 \leq j \leq 2k-6$. If Bob claims one of them, she also claims one. As there is an even number of them, this is always possible.

At the end of the game, by the matching strategy, for each hyperedge C in H', Alice will have played exactly a copy of H_0 in each H_0^C, exactly $k-6$ vertices among the v_j^C for $1 \leq j \leq 2(k-6)$, and the vertices v_i corresponding to the v'_i that Avoider would have played according to $\mathcal{S}'$ in H'.

Now, consider any copy H_1 of H in G. Suppose that Alice has played all the vertices of H_1.

Suppose that H_0 has at least one edge. We first prove that H_1 cannot contain two vertices v_C and $v_{C'}$ for $C \neq C'$ that have been created by the Steps 2 or Step 3 of our construction. Suppose it does. Consider a decomposition of $H_1 = I_k \bowtie H_0^1$. By construction, the v_Cs and the $v_{C'}$s are not adjacent. Therefore, as these two components are fully connected one to the other, they must either be both in I_k or both in H_0^1.

Let $e = (u_1, u_2)$ be an edge of H_0^1. v_C and v'_C cannot be both adjacent to u_1, otherwise, by construction, we would have $C = C'$. Therefore, as only vertices created during Step 1 can be adjacent to both v_C and $v_{C'}$, any vertex in I_k must be a vertex v_i. Which is not possible otherwise, Alice would have play $k \geq 6$ vertices v_i adjacent to a same v_C, which means that, according to $\mathcal{S}'$ she would have played $k \geq 6$ vertices in the same hyperedge in H', which contradicts the fact that $\mathcal{S}'$ was a winning strategy for Avoider in H'.

Now, as all the vertices of H_1 are either v_is or were created by considering the same hyperedge C, and $|H_1| = |H_0| + k$, by the pairing, we know that exactly 6 of them are v_is. As there are no edges between the v_is, they must all be on the same side of the join, and therefore, they are all connected to a same vertex. By construction, this is only the case if these six vertices are in a same hyperedge of H' which contradicts that $\mathcal{S}'$ was a winning strategy for Avoider in H', as these six vertices would then form a hyperedge.

Suppose now that H_0 has no edges and has $k' \geq 6$ vertices.

This means that H can be written $I_k \bowtie I_{k'}$ for $k, k' \geq 6$ (note that H is a complete bipartite graph). Once again, consider two vertices v_C and $v_{C'}$ for $C \neq C'$ in H_1. As they cannot be adjacent, they must be both in I_k or both in $I_{k'}$. Thus, v_C and $v_{C'}$ have $\min(k, k') \geq 6$ common neighbors. This implies that, if $C \neq C'$, at least one of their common neighbor is not a vertex v_i created during Step 1, otherwise Alice would have played six vertices in the same hyperedge of H'. This is not possible by construction. So once again, H_1 cannot contain v_C and $v_{C'}$ created from different hyperedges C and C' in H'. Now, if Alice has played all the vertices of H_1, by construction as $|H_1| = k + k'$, and as her pairing strategy ensures her to play $k-6$ vertices created during step 2 and k' created during step 3, necessarily, she has played six vertices v_i creating during step 1. As there are no edges between these six vertices, they must all be in the same independent set I_k or $I_{k'}$. Thus, they have a common neighbor. This common neighbor must then be a vertex v_C creating during step 3 as only them are connected to the v_is. Finally, these six vertices corresponds to six vertices v'_is that are in the same hyperedge C of H'. Once again, this contradicts the fact that $\mathcal{S}'$ was a winning strategy for Avoider in H'. ◁

▷ Claim 23. If Enforcer has a winning strategy $\mathcal{S}'$ in H', Bob has a winning strategy in G.

Proof. Let $\mathcal{S}'$ be a winning strategy for Enforcer in H'. We consider a strategy $\mathcal{S}$ for Bob in G as follows:

- If Alice claims a vertex v_i, he answers with the vertex v_j that corresponds to the vertex v'_j that Enforcer would have claimed in response to v'_i in $\mathcal{S}'$.
- In H_0^C, as it is a strong product $H_0 \boxtimes P_2$, Bob considers the pairing between any vertex and its copy in the strong product. If Alice plays in one pair of the pairing, he plays the second vertex of the pair.

- For any edge $C \in H'$, Bob pairs the vertices v_j^C for $1 \leq j \leq 2k-6$. If Alice claims one of them, he claims one of them too.
- If at a certain moment of the game, it is Bob's turn and the remaining vertices are all in some H_0^C or vertices v_j^Cs, he applies the pairing strategy, so that Alice plays once in any pair of the matching by Lemma 6.

Consider the graph at the end of the game. As $\mathcal{S}$ was a winning strategy for Enforcer in H', there exists a hyperedge $C \in H'$ in which Avoider has claimed the six vertices. Up to a renaming of the vertices, denote by $v'_1, \ldots, v'_6$ be these six vertices. Alice has then claimed $v_1, \ldots, v_6$ in G. According to the pairing strategy, Bob knows that Alice will play exactly on $k-6$ vertices from the v_j^C, denote them $v_7, \ldots, v_k$, and exactly one copy H_1 of H_0 from the vertices of H_0^C. Now, by construction, the vertices $v_1, \ldots v_k$ are a stable set and all the edges exist between any v_i $(1 \leq i \leq k)$ and any vertex v of H_1. Thus, the subgraph formed by these vertices, which were all played by Alice, is isomorphic to $I_k \bowtie H_0 = H$. Thus, Bob has won. ◁

Finally, Alice has a winning strategy in the Avoider-Enforcer vertex H-Game played on G if and only if Avoider has one in the Avoider-Enforcer game played on H', and determining the winner of the Avoider-Enforcer vertex H-Game is PSPACE-complete. ◀

Parameterized Lower Bounds for Problems in P via Fine-Grained Cross-Compositions

Klaus Heeger
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

André Nichterlein
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Rolf Niedermeier
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

We provide a general framework to exclude parameterized running times of the form $O(\ell^\beta + n^\gamma)$ for problems that have polynomial running time lower bounds under hypotheses from fine-grained complexity. Our framework is based on cross-compositions from parameterized complexity. We (conditionally) exclude running times of the form $O(\ell^{\gamma/(\gamma-1)-\varepsilon} + n^\gamma)$ for any $1 < \gamma < 2$ and $\varepsilon > 0$ for the following problems:

- Longest Common (Increasing) Subsequence: Given two length-n strings over an alphabet Σ (over $\mathbb{N}$) and $\ell \in \mathbb{N}$, is there a common (increasing) subsequence of length ℓ in both strings?
- Discrete Fréchet Distance: Given two lists of n points each and $k \in N$, is the Fréchet distance of the lists at most k? Here ℓ is the maximum number of points which one list is ahead of the other list in an optimum traversal.
- Planar Motion Planning: Given a set of n non-intersecting axis-parallel line segment obstacles in the plane and a line segment robot (called rod), can the rod be moved to a specified target without touching any obstacles? Here ℓ is the maximum number of segments any segment has in its vicinity.

Moreover, we exclude running times $O(\ell^{2\gamma/(\gamma-1)-\varepsilon} + n^\gamma)$ for any $1 < \gamma < 3$ and $\varepsilon > 0$ for:

- Negative Triangle: Given an edge-weighted graph with n vertices, is there a triangle whose sum of edge-weights is negative? Here ℓ is the order of a maximum connected component.
- Triangle Collection: Given a vertex-colored graph with n vertices, is there for each triple of colors a triangle whose vertices have these three colors? Here ℓ is the order of a maximum connected component.
- 2nd Shortest Path: Given an n-vertex edge-weighted digraph, vertices s and t, and $k \in \mathbb{N}$, has the second longest s-t-path length at most k? Here ℓ is the directed feedback vertex set number.

Except for 2nd Shortest Path all these running time bounds are tight, that is, algorithms with running time $O(\ell^{\gamma/(\gamma-1)} + n^\gamma)$ for any $1 < \gamma < 2$ and $O(\ell^{2\gamma/(\gamma-1)} + n^\gamma)$ for any $1 < \gamma < 3$, respectively, are known. Our running time lower bounds also imply lower bounds on kernelization algorithms for these problems.

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

Keywords and phrases FPT in P, Kernelization, Decomposition

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.35

Related Version *Full Version*: https://arxiv.org/abs/2301.00797 [37]

Funding *Klaus Heeger*: Supported by DFG project NI 369/16 "FPTinP".

Acknowledgements In memory of Rolf Niedermeier, our colleague, friend, and mentor, who sadly passed away before this paper was finished.
We thank the anonymous reviewers for their thoughtful and constructive feedback.

© Klaus Heeger, André Nichterlein, and Rolf Niedermeier;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

In recent years, many results in Fine-Grained Complexity showed that many decade-old textbook algorithms for polynomial-time solvable problems are essentially optimal: Consider as an example Longest Common Subsequence (LCS) where, given two input strings with n characters each, the task is to find a longest string that appears as subsequence in both input strings. The classic $O(n^2)$-time algorithm is often taught in introductory courses to dynamic programming [18]. Bringmann and Künnemann [14] and Abboud et al. [1] independently showed that an algorithm solving LCS in $O(n^{2-\varepsilon})$ time for any $\varepsilon > 0$ would refute the Strong Exponential Time Hypothesis (SETH). Such conditional lower bounds have been shown for many polynomial-time solvable problems in the recent years [50].

One approach to circumvent such lower bounds is "FPT in P" [3, 34]. For Longest Common Subsequence there is a (quite old) parameterized algorithm running in $O(kn + n \log n)$ time, where k is the length of the longest common subsequence [40]. Thus, if k is small (e. g. $O(n^{0.99})$), then the $O(n^2)$ barrier can be broken (without refuting the SETH). A natural question is whether we can do better. As $k \leq n$, an algorithm running in $O(k^{1-\varepsilon}n)$ time for any $\varepsilon > 0$ would break the SETH. However, there are no obvious arguments excluding a running time of $O(k^2+n)$. In fact, such additive running times are not only desirable (as again, for small k this would be faster than even $O(kn)$) but also quite common in parameterized algorithmics by employing kernelization: For Longest Common Subsequence the question would be whether there are linear-time applicable data reduction rules that shrink the input to size $O(k)$. Then we could simply apply the textbook algorithm to solve Longest Common Subsequence in overall $O(k^2 + n)$ time. Kernelization is well-studied in the parameterized community [5, 31] and also effective in practice for polynomial-time solvable problems such as Maximum Matching [43] or Minimum Cut [39].

Bringmann and Künnemann [15] showed in an extensive study that such an $O(k^2 + n)$-time algorithm (and indeed many other parameterized algorithms for Longest Common Subsequence) would refute the SETH. This also implies that no such kernelization algorithm as mentioned above is likely to exist. The results of Bringmann and Künnemann [15] are based on very carefully crafted reductions.

In this work, we follow a different route to obtain similar results for various problems. We provide an easy-to-apply, general framework to (conditionally) exclude algorithms with running time $O(k^\beta + n^\gamma)$ for problems admitting conditional running time lower bounds. Indeed we show for various string (including Longest Common Subsequence) and graph problems as well as problems from computational geometry tight trade-offs between β and γ. This shows that the trivial trade-offs are often the best one can hope for.

1.1 Related work

Fine-grained complexity is an active field of research with hundreds of papers. We refer to the survey of Vassilevska Williams [50] for an overview of the results and employed hypotheses.

Over the last couple of years there has been a lot of work in the direction of "FPT in P" for various problems such as Maximum Matching [19, 22, 30, 38, 41, 43, 44, 47], Hyperbolicity [19, 28], and Diameter [3, 19]. Parameterized lower bounds are rare in this line of work. Certain linear-time reductions can be used to exclude any kind of meaningful FPT-running times; this is also known as General-Problem-Hardness [9]. Using various carefully crafted reductions, Bringmann and Künnemann [15] show parameterized running time lower bounds (under SETH) for Longest Common Subsequence with respect to

seven different parameters. In a similar fashion, Duraj et al. [24] show that solving Longest Common Increasing Subsequence in $O((n\ell)^{1-\epsilon})$ time where ℓ is the solution size for some $\epsilon > 0$ would refute SETH.

Fluschnik et al. [29] provide lower bounds for *strict* kernelization (i. e. kernels where the parameter is not allowed to increase) for subgraph detection problems such as Negative Weight Triangle and Triangle Collection. Conceptually, they use the *diminisher*-framework [17, 27] which was originally developed to exclude polynomial-size strict kernels under the assumption P $\neq$ NP. The basic idea is to iteratively apply a diminisher (an algorithm that reduces the parameter at a cost of increasing the instance size) and an (assumed) strict kernel (to shrink and control the instance size) to an instance I of an NP-hard problem. After a polynomial number of rounds, this overall polynomial-time algorithm will return a constant size instance which is equivalent to I, thus arriving at P = NP. Fluschnik et al. [29] applied the same idea to polynomial-time solvable problems. In contrast, we rely and adjust the composition-framework by Bodlaender et al. [11] which was developed to exclude (general) polynomial-size kernels under the stronger assumption NP $\not\subseteq$ coNP / poly.

The composition framework works as follows. Consider the example of the NP-hard problem Negative-Weight Clique: Given an edge-weighted graph G and an integer k, does G contain a negative-weight k-clique, that is, a clique on k vertices where the sum of the edge-weights of the edges within the clique is negative.

Let $(G_1, k), (G_2, k), \ldots, (G_t, k)$ be several instances of Negative-Weight Clique with the same k. Clearly, the graph G obtained by taking the disjoint union of all G_i contains a negative-weight k-clique if and only if some G_i contains a negative-weight k-clique. Moreover, the largest connected component of G has order $\max_{i\in[t]}\{|V(G_i)|\}$. Now assume that Negative-Weight Clique has a kernel of size $O(\ell^c)$ for some constant c where ℓ is the order of a largest connected component. By choosing $t = k^{c+1}$, it follows that kernelizing the instance (G, k) yields an instance of size less than ℓ, that is, less bits than the number of instances encoded in G. Given the NP-hardness of Negative-Weight Clique such a compression seems challenging; indeed it would imply NP $\subseteq$ coNP / poly [32], which in turn results in a collapse of the polynomial hierarchy. Compositions and their extension cross-composition [12] are extensively employed in the parameterized complexity literature. Moreover, to exclude kernels whose size is bounded by polynomials of a specific degree adjustments have been made to the composition framework [20].

Parameter trade-offs. For several of our running time lower bounds we have tight upper bounds that are derived from a simple case distinction argument.

▶ **Observation 1.1** (folklore). *If a problem $\mathcal{P}$ admits an $\widetilde{O}(\ell^\beta n^\gamma)$-time algorithm*[1]*, then it admits for every $\lambda > 0$ an $\widetilde{O}(\ell^{\beta+\frac{\gamma\cdot\beta}{\lambda}} + n^{\gamma+\lambda})$-time algorithm.*

Proof. If $\ell \leq n^{\frac{\lambda}{\beta}}$, then the $\widetilde{O}(\ell^\beta n^\gamma)$-time algorithm runs in $\widetilde{O}(n^{\gamma+\lambda})$ time. Otherwise $n \leq \ell^{\frac{\beta}{\lambda}}$. Then the $\widetilde{O}(\ell^\beta n^\gamma)$-algorithm then in $\widetilde{O}(\ell^{\beta+\frac{\gamma\cdot\beta}{\lambda}})$ time. ◀

1.2 Our Results & Technique

We provide a composition-based framework to establish parameterized running time lower bounds and apply the framework to Longest Common Subsequence, Longest Common (Weakly) Increasing Subsequence, Discrete Fréchet Distance, Planar Motion

[1] The $\widetilde{O}$ hides polylogarithmic factors.

Table 1 Overview of achievable running times. The upper part of the table lists the results for four problems that can be solved in $O(n^2)$ time but under SETH or 3SUM-hypothesis not in $O(n^{2-\varepsilon})$ time for any $\varepsilon > 0$. The lower part lists results for three graph problems that, based on the APSP-hypothesis, do not admit $O(n^{3-\varepsilon})$-time algorithms. The parameterized upper and lower bounds are visualized in Figure 1.

problems	Longest Common Subsequence	$\ell \hat{=}$solution size
	Longest Common (Weakly) Increasing Subsequence	$\ell \hat{=}$solution size
	Discrete Fréchet Distance	$\ell \hat{=}$maximum shift
	Planar Motion Planning	$\ell \hat{=}$max. number of segments in the vincinty of any segment

results	upper bounds	lower bounds
	$O(n^2)$ [6, 7, 35, 52]	no $O(n^{2-\varepsilon})$ assuming SETH / 3SUM [1, 13, 14, 33]
	$\widetilde{O}(\ell n)$ [40, 45, 49]	
	$\widetilde{O}(\ell^{\gamma/(\gamma-1)} + n^\gamma)$ for each $\gamma > 1$	no $O(\ell^{\gamma/(\gamma-1)-\varepsilon} + n^\gamma)$ for any $\gamma < 2$ [15, 24]
	(Observation 1.1)	(Corollaries 4.6, 4.8, and 4.10)

problems	Negative Triangle	$\ell \hat{=}$size of maximum component
	Triangle Collection	$\ell \hat{=}$size of maximum component
	2nd Shortest Path (only lower bounds)	$\ell \hat{=}$directed feedback vertex set size

results	upper bounds	lower bounds
	$O(n^3/2^{\Theta(\log^{0.5} n)})$ [16, 53]	no $O(n^{3-\varepsilon})$ assuming APSP [51]
	$\widetilde{O}(\ell^2 n)$ (folklore)	
	$\widetilde{O}(\ell^{2\gamma/(\gamma-1)} + n^\gamma)$ for each $\gamma > 1$	no $O(\ell^{2\gamma/(\gamma-1)-\varepsilon} + n^\gamma)$ for any $\gamma < 3$
	(Observation 1.1)	(Corollaries 4.12 and 4.14 and Proposition 4.15)

Planning, Negative Triangle, and 2nd Shortest Path (see Section 1.3 for the problem definitions). Using similar ideas we obtain running time lower bounds for Triangle Collection. For all these problems except 2nd Shortest Path parameterized by the directed feedback vertex set there exist matching running time upper bounds. We refer to Table 1 for an overview on the specific results and the parameterization. Moreover, we visualize in Figure 1 the trade-offs in the running times that are (im-)possible.

Framework. We adjust the cross-composition framework to obtain lower bounds for polynomial time solvable problems. As an example, consider Negative-Weight Triangle, that is, Negative-Weight Clique with k fixed to three. Assuming the APSP-hypothesis, Negative-Weight Clique cannot be solved in $O(n^{3-\varepsilon})$ time [51]. The first difference to the cross-composition framework is that we start with one instance G of Negative-Weight Triangle which we then decompose into many small instances as follows: Partition the vertices $V(G)$ of G into z many sets $V_1, \ldots, V_z$ of size n/z, where z is chosen depending on the running time we want to exclude (see the proof of Lemma 3.3 in Section 3 for the actual formula specifying z). Then, we create z^3 instances of Negative-Weight Clique: for each $(i, j, k) \in [z]^3$ take the graph $G[V_i \cup V_j \cup V_k]$. Clearly, we have that G contains a negative-weight triangle if and only if at least one of the created instances contains a negative-weight triangle.

Next, we apply the composition as explained above (the disjoint union) for Negative-Weight Clique to obtain an instance G' with $n' = z^3 \cdot n/z = z^2 n$ vertices. Note that the size ℓ of a largest connected component in G' is $3n/z$. Hence, an algorithm running in

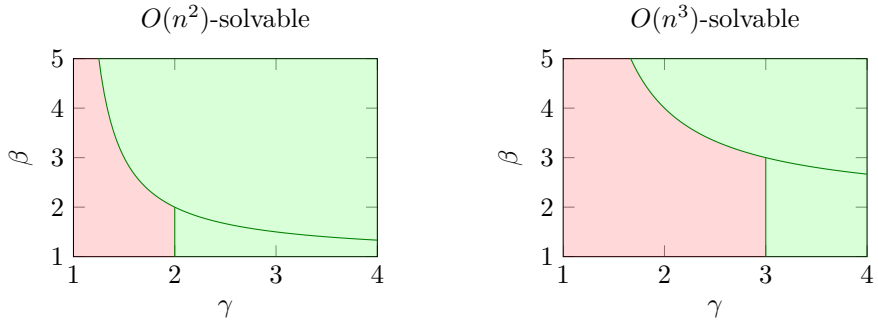


Figure 1 Overview on the possible (in green) and unlikely (in red) trade-offs in running times of the form $O(n^\gamma + \ell^\beta)$. Left: First category for $O(n^2)$-time solvable problems (upper part in Table 1); right: second category for $O(n^3)$-time solvable problems (lower part in Table 1).

time $O(n^\gamma + \ell^\beta)$ for Negative-Weight Triangle solves G' in time $O(z^{2\gamma}n^\gamma + (3n/z)^\beta)$. By carefully choosing z as a function in n, β, and γ, we get that this is in $O(n^{3-\varepsilon})$ for various combinations of γ and β.

The property that Negative-Weight Triangle can be decomposed as above is not unique to the problem. In fact, this has been observed already: "Many problems, like SAT, have a simple self-reduction proving that the "Direct-OR" version is hard, assuming the problem itself is hard" [2]. Our framework formalizes this notion of decomposition (see Section 2 for a definition) and adjusts the cross-composition definition. We furthermore show that commonly used "hard" problems such as Orthogonal Vectors, 3-Sum, and Negative-Weight k-Clique are decomposable. Thus, it remains to show cross-compositions in order to apply our framework and obtain lower bounds.

1.3 Preliminaries and Notation

Problem definitions. For $\ell \in \mathbb{N}$ we set $[\ell] := \{1, 2, \ldots, \ell\}$.

Orthogonal Vectors
Input: Two size-n sets $A, B \subseteq \{0,1\}^d$ for some $d \in \mathbb{N}$.
Question: Are there $a \in A$ and $b \in B$ so that a and b are orthogonal, i.e., for each $i \in [d]$, the i-th coordinate of a or the i-th coordinate of b is zero.

We denote the restriction of Orthogonal Vectors to instances with $d \leq O(\log n)$ as Orthogonal Vectors *with logarithmic dimension*.

3-Sum
Input: An array A of n integers.
Question: Are there $i, j, h \in [n]$ such that $A[i] + A[j] + A[h] = 0$?

Negative-Weight k-Clique
Input: An edge-weighted graph G on n vertices.
Question: Does G contain a k-clique of negative weight?

Longest Common Subsequence
Input: Two strings x^1 and x^2 of length n over an alphabet Σ and $k \in \mathbb{N}$.
Question: Decide whether there is a common subsequence of length k of x^1 and x^2.

Longest Common (Weakly) Increasing Subsequence

Input: Two strings x^1 and x^2 of length n over $\mathbb{N}$ and $k \in \mathbb{N}$.

Question: Decide whether there is a common subsequence y of length k of x^1 and x^2 with $y[i] < y[i+1]$ ($y[i] \leq y[i+1]$) for all $i \in [k-1]$.

Discrete Fréchet Distance

Input: Two lists $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$ of points in the plane and $k \in \mathbb{Q}$.

Question: Is the Fréchet distance of P and Q at most k, that is, there are two surjective, non-decreasing functions $f_P, f_Q : [2n] \to [n]$ with $f_P(1) = 1 = f_Q(1)$, $f_P(2n) = n = f_Q(2n)$ and $\max_{i \in [2n]} \operatorname{dist}(p_{f_P(i)}, q_{f_Q(i)}) \leq k$?

Planar Motion Planning

Input: A set of n non-intersecting, non-touching, axis-parallel line segment obstacles in the plane and a line segment robot (a rod or ladder), a given source, and a given goal.

Question: Can the rod be moved (allowing both translation and rotation) from the source to the goal without colliding with the obstacles?

2nd Shortest Path

Input: An n-vertex edge-weighted directed graph G, vertices s and t, and $k \in N$.

Question: Has the 2nd-shortest s-t-path length at most k?

Triangle Collection

Input: A vertex-colored graph G on n vertices.

Question: For each combination of three colors, does G contain a triangle whose vertices are colored with three colors?

Hypotheses. The conditional lower bounds in this work are based on SETH, 3-Sum-, and the APSP-Hypothesis (see Vassilevska Williams [50] for more details).

▶ **Hypothesis 1** (SETH). For every $\varepsilon > 0$ there exists a $k \in \mathbb{N}$ such that k-SAT cannot be solved in $O(2^{(1-\varepsilon)n})$ time, where n is the number of variables.

▶ **Hypothesis 2** (3SUM). 3-Sum on n integers in $\{-n^4, \ldots, n^4\}$ cannot be solved in $O(n^{2-\varepsilon})$ time for any $\varepsilon > 0$.

▶ **Hypothesis 3** (APSP). All Pairs Shortest Path on n-vertex graphs with polynomial edge weights cannot be solved in $O(n^{3-\varepsilon})$ time for any $\varepsilon > 0$.

Parameterized Complexity. In many of the above problems, n is *not* the input size but a parameter and the input size is bounded by $n^{O(1)}$. A parameterized problem is a set of instances $(I, p) \in \Sigma^* \times \Sigma^*$, where Σ denotes a finite alphabet, I denotes the classical instance and p the parameter. A *kernelization* is a polynomial-time algorithm that maps any instance (I, p) to an equivalent instance (I', p') (the *kernel*) such that $|I'| + p' \leq f(p)$ for some computable function f. If f is a polynomial, then (I', p') is a polynomial-size kernel.

In this work we restrict ourselves to the following: Either $p = n$ (p is a single parameter) or $p = (n, \ell)$ (p is a combined parameter). Moreover, both n and ℓ are always nonnegative integers, n is related to the input size but ℓ is not (ℓ can be seen as "classical" parameter).

Due to space restrictions we defer proofs of results marked with a $\star$ to a full version [37].

2 Framework

Our framework has the following three steps (see Section 1.2 for a high-level description).

1. Start with an instance $(I, n_{\mathcal{P}})$ of a "hard" problem $\mathcal{P}$ and decompose it into the disjunction of t instances $(I_1, n_1), \ldots, (I_t, n_t)$ of $\mathcal{P}$. In Section 3, we provide such decompositions for the frequently used hard problems 3-Sum, Orthogonal Vectors, and Negative Weight k-Clique.
2. Compose $(I_1, n_1), \ldots, (I_t, n_t)$ into one instance $(J, (n, \ell))$ of the "target" problem using an OR-cross-composition. This step has to be done for the application at hand.
3. Apply the assumed $\widetilde{O}(n^\gamma + \ell^\beta)$-time algorithm to J. If the combination of γ and β is small enough, then the resulting algorithm will be faster than the lower bound for $\mathcal{P}$.

To give a more formal description of our framework, we first define decompositions and cross-compositions. Note that all mentioned problems are parameterized problems.

▶ **Definition 2.1** (OR-decomposition). *For $\alpha > 1$ an α-OR-decomposition for a problem $\mathcal{P}$ is an algorithm that, given $\lambda > 0$ and an instance (I, n) of $\mathcal{P}$, computes for some $\alpha' < \alpha$ in $\widetilde{O}(n^{\alpha'})$ time $t \in \widetilde{O}(n^{\alpha\lambda/(\alpha+\lambda)})$ many instances $(I_1, n_1), \ldots, (I_t, n_t)$ of $\mathcal{P}$ such that*

- *$(I, n) \in \mathcal{P}$ if and only if $(I_i, n_i) \in \mathcal{P}$ for some $i \in [t]$, and*
- *$n_i \in \widetilde{O}(n^{\alpha/(\alpha+\lambda)})$ for all $i \in [t]$.*

We say a problem $\mathcal{P}$ is *α-OR-decomposable* if there exists an α-OR-decomposition for it. For some problems it is easier to show OR-decomposability than others. Thus, using appropriate reductions to transfer OR-decomposability can be desirable (we do so in Section 3 when showing that 3-Sum is 2-OR-decomposable). Quasi-linear time reductions that do not increase the parameter to much are one option. To this end, we say a reduction that given an instance $(I^{\mathcal{P}}, n^{\mathcal{P}})$ of $\mathcal{P}$ produces an instance $(I^{\mathcal{Q}}, n^{\mathcal{Q}})$ of $\mathcal{Q}$ is *parameter-preserving* if $n^{\mathcal{Q}} \in \widetilde{O}(n^{\mathcal{P}})$.

▶ **Proposition 2.2.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two problems such that there are quasi-linear-time parameter-preserving reductions from $\mathcal{P}$ to $\mathcal{Q}$ and from $\mathcal{Q}$ to $\mathcal{P}$. Then for any $\alpha > 1$, $\mathcal{P}$ is α-OR-decomposable if and only if $\mathcal{Q}$ is.*

Proof. Assume that $\mathcal{P}$ is α-OR-decomposable (the case that $\mathcal{Q}$ is α-OR-decomposable is symmetric). We now give an α-OR-decomposition for $\mathcal{Q}$. Given an instance $(I^{\mathcal{Q}}, n^{\mathcal{Q}})$ and $\lambda > 0$, we first reduce $(I^{\mathcal{Q}}, n^{\mathcal{Q}})$ to an instance $(I^{\mathcal{P}}, n^{\mathcal{P}})$. Afterwards, we apply the α-OR-decomposition from $\mathcal{P}$, resulting in instances $(I_1^{\mathcal{P}}, n_1^{\mathcal{P}}), \ldots, (I_t^{\mathcal{P}}, n_t^{\mathcal{P}})$ of $\mathcal{P}$. Finally, we reduce each instance $(I_i^{\mathcal{P}}, n_i^{\mathcal{P}})$ to an instance $(I_i^{\mathcal{P}}, n_i^{\mathcal{P}})$. This clearly is an α-OR-decomposition for $\mathcal{Q}$. ◀

For the second step of our framework, we introduce fine-grained OR-cross-compositions:

▶ **Definition 2.3** (fine-grained OR-cross-composition). *For $\nu \geq 1, \mu \geq 0$ an (ν, μ)-OR-cross-composition from a problem $\mathcal{P}$ to a problem $\mathcal{Q}$ is an algorithm $\mathcal{A}$ which takes as an input t instances $(I_1^{\mathcal{P}}, n_1^{\mathcal{P}}), \ldots, (I_t^{\mathcal{P}}, n_t^{\mathcal{P}})$ of $\mathcal{P}$, runs in $\widetilde{O}(t \cdot n_{\max}^\nu + \sum_{i=1}^t |I_i^{\mathcal{P}}|)$ time with $n_{\max} \coloneqq \max_{i \in [t]} n_i^{\mathcal{P}}$, and computes an instance $(I^{\mathcal{Q}}, (n^{\mathcal{Q}}, \ell^{\mathcal{Q}}))$ of $\mathcal{Q}$ such that*

1. *$(I^{\mathcal{Q}}, (n^{\mathcal{Q}}, \ell^{\mathcal{Q}})) \in \mathcal{Q}$ if and only if $(I_i^{\mathcal{P}}, n_i^{\mathcal{P}}) \in \mathcal{P}$ for some $i \in [t]$, and*
2. *$n^{\mathcal{Q}} \in \widetilde{O}(t \cdot n_{\max}^\nu)$ and $\ell^{\mathcal{Q}} \in \widetilde{O}(n_{\max}^\mu)$.*

We say a problem $\mathcal{P}$ *(ν, μ)-OR-cross-composes* into a problem $\mathcal{Q}$ if there exists an (ν, μ)-OR-cross-composition from $\mathcal{P}$ to $\mathcal{Q}$.

▶ **Theorem 2.4.** *Let $\alpha > \nu \geq 1$, $\gamma > 1$, and $\mu > 0$ with $\alpha > \nu \cdot \gamma$. Let $\mathcal{P}$ be an α-OR-decomposable problem with parameter $n_{\mathcal{P}}$ that (ν, μ)-OR-cross-composes into a problem $\mathcal{Q}$ with parameters $n_{\mathcal{Q}}$ and $\ell_{\mathcal{Q}}$. If there is an $\widetilde{O}(n_{\mathcal{Q}}^{\gamma} + \ell_{\mathcal{Q}}^{\beta})$-time algorithm for $\mathcal{Q}$ and*

$$0 < \beta < \frac{\gamma \cdot (\alpha - \nu)}{(\gamma - 1) \cdot \mu},$$

then $\mathcal{P}$ can be solved in $O(n_{\mathcal{P}}^{\alpha - \varepsilon})$ time for some $\varepsilon > 0$ time.

Proof. Let $(I_{\mathcal{P}}, n_{\mathcal{P}})$ be an instance of $\mathcal{P}$. Our algorithm to solve $(I_{\mathcal{P}}, n_{\mathcal{P}})$ runs in the following steps:

1. Apply the α-OR-decomposition (with λ specified below) to obtain the instances (I_1, n_1), $\ldots, (I_t, n_t)$ with $\max_{i \in [t]} n_i \leq q := n_{\mathcal{P}}^{\alpha/(\alpha+\lambda)}$ and $t := n_{\mathcal{P}}^{\alpha\lambda/(\alpha+\lambda)} = q^{\lambda}$. Note that, by definition, this step runs in $\widetilde{O}(n_{\mathcal{P}}^{\alpha'})$ time for some $\alpha' < \alpha$.
2. Apply the (ν, μ)-OR-cross-composition to compute the instance $(I_{\mathcal{Q}}, (n_{\mathcal{Q}}, \ell_{\mathcal{Q}}))$ for $\mathcal{Q}$ from $(I_1, n_1), \ldots, (I_t, n_t)$. Note that the running time and $n_{\mathcal{Q}}$ is in $\widetilde{O}(t \cdot q^{\nu} + \sum_{i=1}^{t} |I_i^{\mathcal{P}}|) = \widetilde{O}(q^{\lambda+\nu} + n_{\mathcal{P}}^{\alpha'})$. Moreover, $\ell_{\mathcal{Q}} \in \widetilde{O}(q^{\mu})$.
3. Apply the algorithm with running time $\widetilde{O}(n_{\mathcal{Q}}^{\gamma} + \ell_{\mathcal{Q}}^{\beta})$. This requires $\widetilde{O}(q^{(\lambda+\nu)\gamma} + q^{\mu \cdot \beta})$ time.

It remains to show that all three steps run in $O(n_{\mathcal{P}}^{\alpha-\varepsilon})$ time for some $\varepsilon > 0$. To this end, we now specify $\lambda = \beta\mu/\gamma - \nu$. Note that $\lambda + \nu = \beta\mu/\gamma < \mu \cdot \beta$ and thus it suffices to show that the third step runs in $\widetilde{O}(n_{\mathcal{P}}^{\alpha-\epsilon})$ for some $\epsilon > 0$. The last step runs in $\widetilde{O}(q^{\mu\cdot\beta}) = \widetilde{O}(n_{\mathcal{P}}^{\alpha\mu\cdot\beta/(\alpha+\lambda)})$ time. The exponent is

$$\frac{\alpha\mu\beta}{\alpha + \lambda} = \frac{\alpha\mu\beta}{\alpha + \beta\mu/\gamma - \nu} = \frac{\alpha\beta\gamma\mu}{\gamma(\alpha - \nu) + \beta\mu} = \frac{\alpha\gamma\mu}{\gamma(\alpha - \nu)/\beta + \mu}.$$

By assumption we have $\beta < \frac{\gamma \cdot (\alpha - \nu)}{(\gamma - 1) \cdot \mu}$ and thus the exponent is

$$\frac{\alpha\gamma\mu}{\gamma(\alpha - \nu)/\beta + \mu} < \frac{\alpha\gamma\mu}{\gamma(\alpha - \nu)/\left(\frac{\gamma \cdot (\alpha - \nu)}{(\gamma - 1) \cdot \mu}\right) + \mu} = \frac{\alpha\gamma\mu}{(\gamma - 1) \cdot \mu + \mu} = \alpha.$$

There is still one thing left to do: We must ensure that $\lambda > 0$. This will not always be the case as when $\beta \to 0$, λ gets negative. However, an $\widetilde{O}(n_{\mathcal{Q}}^{\gamma} + \ell_{\mathcal{Q}}^{\beta})$-time algorithm also implies for any $\beta' > \beta$ an $\widetilde{O}(n_{\mathcal{Q}}^{\gamma} + \ell_{\mathcal{Q}}^{\beta'})$-time algorithm. Thus, we can simply pick some larger β' such that the corresponding λ' is larger than 0. To do so, let $\beta_{\max} := \frac{\gamma \cdot (\alpha - \nu)}{(\gamma - 1) \cdot \mu}$ the upper bound for β. Note that $\alpha > \nu \cdot \gamma$ implies that

$$\lambda_{\max} := \frac{\beta_{\max}\mu}{\gamma} - \nu = \frac{\frac{\gamma \cdot (\alpha - \nu)}{(\gamma - 1) \cdot \mu}\mu}{\gamma} - \nu = \frac{\alpha - \nu}{\gamma - 1} - \nu = \frac{\alpha - \nu - \nu \cdot (\gamma - 1)}{\gamma - 1} = \frac{\alpha - \nu \cdot \gamma}{\gamma - 1} > 0$$

Thus, we can pick $\beta < \beta' < \beta_{\max}$ such that $\lambda' := \frac{\beta'\mu}{\gamma} - \nu > 0$. ◀

Note that if for $\mathcal{P}$ there is a (conditional) running time lower bound of $\Omega(n^{\alpha})$, then Theorem 2.4 excludes (conditionally) running times of the form $\widetilde{O}(n_{\mathcal{P}} + \ell^{\beta})$ for any $\beta \in \mathbb{R}$ as $\lim_{\gamma \to 1} \frac{\gamma \cdot (\alpha - \nu)}{(\gamma - 1) \cdot \mu} = \infty$. This running time also excludes linear-time computable polynomial-size kernels. More precisely, we get the following.

▶ **Corollary 2.5.** *Let $\alpha > \nu \geq 1$, $\gamma > 1$, and $\mu > 0$ with $\alpha > \nu \cdot \gamma$. Let $\mathcal{P}$ be an α-OR-decomposable problem with parameter $n_{\mathcal{P}}$ that (ν, μ)-OR-cross-composes into a problem $\mathcal{Q}$ with parameters $n_{\mathcal{Q}}$ and $\ell_{\mathcal{Q}}$. Assume that there is an $\widetilde{O}(n_{\mathcal{Q}}^{\xi})$ algorithm for deciding $\mathcal{Q}$ and*

that $n_{\mathcal{Q}}$ is upper bounded by the input size. If there exists an $\widetilde{O}(\ell_{\mathcal{Q}}^{\beta})$-size $\widetilde{O}(n_{\mathcal{Q}}^{\gamma})$-time kernel for $\mathcal{Q}$ for some $\gamma > 1$, $\beta \in \mathbb{R}$, and

$$0 < \beta < \frac{\gamma \cdot (\alpha - \nu)}{(\gamma - 1) \cdot \mu \cdot \xi},$$

then $\mathcal{P}$ can be solved in $O(n_{\mathcal{P}}^{\alpha-\varepsilon})$ time for some $\varepsilon > 0$.

Proof. An $\widetilde{O}(\ell_{\mathcal{Q}}^{\beta})$-size $\widetilde{O}(n_{\mathcal{Q}}^{\gamma})$-time kernel together with an $\widetilde{O}(n_{\mathcal{Q}}^{\xi})$-time algorithm solving $\mathcal{Q}$ yields an $\widetilde{O}(n_{\mathcal{Q}}^{\gamma} + \ell_{\mathcal{Q}}^{\beta\xi})$ algorithm for $\mathcal{Q}$. The corollary now follows from Theorem 2.4. ◀

Our general approach to apply our framework is follows. Start with a problem $\mathcal{P}$ that (under some hypothesis) cannot be solved in $O(n^{\alpha-\varepsilon})$ time for $\varepsilon > 0$. Then construct an α-decomposition for $\mathcal{P}$ followed by a (1,1)-OR-cross-composition into the target problem.

3 OR-decomposable problems

In order to apply our framework, we first need some OR-decomposable problems. We will observe that three fundamental problems from fine-grained complexity, namely Orthogonal Vectors, 3-SUM and Negative-Weight k-Clique, are OR-decomposable. These problems are also our source for running time lower bounds: Note that the former two problems cannot be solved in $O(n^{2-\varepsilon})$ time unless the SETH respectively 3-Sum-hypothesis fail [50]. We moreover use that Negative-Weight 3-Clique (= Negative Triangle) cannot be solved in $O(n^{3-\varepsilon})$ time unless APSP-hypothesis fails [51].

We will use that "Many problems, like SAT, have a simple self-reduction proving that the "Direct-OR" version is hard, assuming the problem itself is hard" [2]. This self-reduction is based on partitioning the instance into many small ones, with at least one of them containing the small desired structure (i.e., a pair of orthogonal vectors, three numbers summing to 0, or a negative triangle).

Orthogonal Vectors. We now show that Orthogonal Vectors is 2-OR-decomposable.

▶ **Lemma 3.1.** *Orthogonal Vectors parameterized by the number of vectors is 2-OR-decomposable.*

Proof. Let (I, n) be an instance of Orthogonal Vectors and $\lambda > 0$. Set $\epsilon := \frac{2}{2+\lambda}$. Partition A into $z := \lceil n^{1-\epsilon} \rceil$ many sets $A_1, \ldots, A_z$ of at most $\lceil n^{\epsilon} \rceil$ vectors each. Symmetrically, partition B into $B_1, \ldots, B_z$ of at most $\lceil n^{\epsilon} \rceil$ vectors each. We assume without loss of generality that $|A_i| = |B_j| = \lceil n^{\epsilon} \rceil =: n'$ (this can be achieved e.g. by adding the all-one vector). For each pair $(i, j) \in [z]^2$, create an instance $(I_{(i,j)}, n') = ((A_i, B_j), n')$ of Orthogonal Vectors. We claim that this constitutes a 2-OR-decomposition.

The number of vectors n' of each instance $I_{(i,j)}$ is $n' = O(n^{2/(2+\lambda)})$. The number of created instances is $z^2 = O(n^{2\cdot(1-\epsilon)}) = O(n^{2\cdot(1-2/(2+\lambda))}) = O(n^{(4+2\lambda-4)/(2+\lambda)}) = O(n^{2\lambda/(2+\lambda)})$. The running time to compute the decomposition is $\widetilde{O}(z^2 \cdot n') = \widetilde{O}(n^{2-\epsilon})$.

It remains to show that (I, n) is a "Yes"-instance if and only if $(I_{(i,j)}, n')$ is a "Yes"-instance for some $(i, j) \in [z]^2$. First assume that (I, n) is a "Yes"-instance. Then there exists some $a \in A$ and $b \in B$ such that a and b are orthogonal. Let $i \in [z]$ such that $a \in A_i$ and $j \in [z]$ such that $b \in B_j$. Then a and b are orthogonal vectors in $(I_{(i,j)}, n')$, showing that $(I_{(i,j)}, n')$ is a "Yes"-instance.

Finally, assume that there exists $(i^*, j^*) \in [z]^2$ such that $(I_{(i^*,j^*)}, n')$ is a "Yes"-instance. Then there exists $a \in A_{i^*}$ and $b \in B_{j^*}$ which are orthogonal. Consequently, a and b are orthogonal vectors in I, implying that (I, n) is a "Yes"-instance. ◀

▶ Remark 3.2. Note that the above decomposition does not change the dimension d. Thus, even restricted versions of Orthogonal Vectors with $d \in O(\log n)$ are 2-OR-decomposable. Further, we can assume that all constructed instances of Orthogonal Vectors have the same number of vectors and the same dimension.

Negative-Weight k-Clique. We now show that Negative-Weight k-Clique is k-OR-decomposable.

▶ **Lemma 3.3.** *For any $k \geq 3$, Negative-Weight k-Clique parameterized by the number of vertices is k-OR-decomposable.*

Proof. The proof follows the ideas from Lemma 3.1. Let $(I = (G, w), n)$ be an instance of Negative-Weight k-Clique and $\lambda > 0$. Set $\epsilon := \frac{k}{k+\lambda}$. Partition the set $V(G)$ of vertices into $z := \lceil n^{1-\epsilon} \rceil$ many sets $V_1, \ldots, V_z$ of size at most $\lceil n^\epsilon \rceil$. We assume without loss of generality that $|V_i| = \lceil n^\epsilon \rceil$ for all $i \in [z]$ (this can be achieved e.g. by adding isolated vertices). Let $n_{(i_1,\ldots,i_k)} := |\{i_1, \ldots, i_k\}| \cdot |V_1|$. For each tuple $(i_1, \ldots, i_k) \in [z]^k$, create an instance $(I_{(i_1,\ldots,i_k)} = G[A_{i_1} \cup \cdots \cup A_{i_k}], n_{(i_1,\ldots,i_k)})$ of Negative-Weight k-Clique. We claim that this constitutes a k-OR-decomposition.

Each instance $(I_{(i,j)}, n_{(i_1,\ldots,i_k)})$ has at most $O(k \cdot n^\epsilon) = O(n^{k/(k+\lambda)})$ vertices (note that k is a constant here). The number of created instances is $z^k = O(n^{k\cdot(1-\epsilon)}) = O(n^{k\cdot(1-k/(k+\lambda))}) = O(n^{(k^2+k\lambda-k^2)/(k+\lambda)}) = O(n^{k\lambda/(k+\lambda)})$. Further, the summed size of all created instances and therefore also the running time is $\widetilde{O}(z^k \cdot k \cdot n^{2\epsilon}) = \widetilde{O}(n^{k-k\epsilon+2\epsilon}) = \widetilde{O}(n^{k-(k-2)\epsilon}) = \widetilde{O}(n^{k'})$ for some $k' < k$.

It remains to show that (I, n) is a "Yes"-instance if and only if $(I_{(i_1,\ldots,i_k)}, n_{(i_1,\ldots,i_k)})$ is a "Yes"-instance for some $(i_1, \ldots, i_k) \in [z]^k$. First assume that (I, n) is a "Yes"-instance. Thus, G contains a negative-weight clique $C = \{v_1, \ldots, v_k\}$. Let i_j such that $v_j \in V_{i_j}$. Then C is a negative-weight k-clique in $(I_{(i_1,\ldots,i_k)}, n_{(i_1,\ldots,i_k)})$.

Finally, assume that there exists $(i_1, \ldots, i_k) \in [n]^k$ such that $(I_{(i_1,\ldots,i_k)}, n_{(i_1,\ldots,i_k)})$ is a "Yes"-instance. Then there exists a clique in $G[V_{i_1} \cup \cdots \cup V_{i_k}]$. Since G contains $G[V_{i_1} \cup \cdots \cup V_{i_k}]$, also G contains a negative-weight k-clique. ◀

3-Sum. Showing that 3-Sum is 2-OR-decomposable requires some more work. We defer these parts to a full version.

▶ **Lemma 3.4** ($\star$). *3-Sum parameterized by the number of numbers is 2-OR-decomposable.*

Applying the framework. The above results make our framework easier to apply. To apply Theorem 2.4 we only need to provide a suitable OR-cross composition from one of the three problems discussed above. We thus arrive at the following.

▶ **Proposition 3.5.** *Let $\mathcal{Q}$ be a problem with parameters $n_\mathcal{Q}$ and $\ell_\mathcal{Q}$. If Orthogonal Vectors resp. 3-Sum parameterized by n $(1,1)$-OR-cross-composes into Q, then an $O(\ell_\mathcal{Q}^\beta + n_\mathcal{Q}^\gamma)$-time algorithm for $\mathcal{Q}$ for any $2 > \gamma > 1$ and $\beta < \gamma/\gamma-1$ refutes the SETH respectively the 3-Sum-Hypothesis.*

4 Applications

We now apply our framework from Theorem 2.4 to several problems from different areas such as string problems (Section 4.1), computational geometry (Section 4.2), and subgraph isomorphism (Section 4.3).

4.1 String problems

We start with Longest Common Subsequence that can be solved in $O(n^2)$ time algorithm [18]. Assuming SETH, there is no algorithm solving Longest Common Subsequence in $O(n^{2-\varepsilon})$ time for any $\varepsilon > 0$ [14, 1]. However, Longest Common Subsequence can be solved in $O(kn + n \log n)$ time, where k is the length of the longest common subsequence [40]. Bringmann and Künnemann [15] proved that this running time is essentially optimal under SETH. Here we reprove this fact using our framework. We refer to Bringmann and Künnemann [15] for a more extensive literature review on Longest Common Subsequence.

A *string* over an alphabet Σ is an element from Σ^*. We access the i-th element of a string x via $x[i]$. A *subsequence* of a string x is a string y such that there is an injective, strictly increasing function f with $y[i] = x[f(i)]$ for all i. A *common subsequence* of two strings x and x' is a string which is a subsequence of both x and x'. For two strings x and y, we denote their concatenation (i.e. the string starting with x and ending with y) by $x \circ y$.

▶ **Lemma 4.1.** *Orthogonal Vectors with logarithmic dimension parameterized by the number of vectors $(1,1)$-OR-cross-composes into Longest Common Subsequence parameterized by the length of the input strings and the length k of the longest common subsequence.*

Proof. Let $(I_1^{\mathrm{OV}}, n_1), \ldots, (I_t^{\mathrm{OV}}, n_t)$ be instances of Orthogonal Vectors with logarithmic dimension. We denote by d_i the dimension of I_i^{OV}. Abboud, Backurs, and Williams [1] gave a reduction from Orthogonal Vectors to Longest Common Subsequence which, given an instance of Orthogonal Vectors with n vectors of dimension d, constructs in $n \cdot d^{O(1)}$ time an equivalent instance of Longest Common Subsequence with strings of length $n \cdot d^{O(1)}$. We apply this reduction to each instance (I_i^{OV}, n_i) of OV, giving us an instance $I_i^{\mathrm{LCS}} = ((x_i^1, x_i^2), (n_i^{\mathrm{LCS}}, k_i))$ with $k_i = O(n_i \cdot d_i^{O(1)})$ and $n_i^{\mathrm{LCS}} = n_i \cdot d_i^{O(1)}$. We assume that $k_i = k_j$ for every i, j (this can be achieved by appending identical sequences of appropriate length to strings x_i^1 and x_i^2), and set $k := k_i$. Further, we assume that the alphabets used for I_i^{LCS} and I_j^{LCS} are disjoint for $i \neq j$. We define $x^1 := x_1^1 \circ x_2^1 \circ \cdots \circ x_t^1$ and $x^2 := x_t^2 \circ x_{t-1}^2 \circ \cdots \circ x_1^2$. The OR-cross-composition constructs the instance (x^1, x^2, k) (the parameter is (n^{LCS}, k) with $n^{\mathrm{LCS}} = \sum_{i=1}^t n_i^{\mathrm{LCS}}$).

We now show correctness of the reduction. First assume that (I_i^{OV}, n_i) is a "Yes"-instance for some $i \in [t]$. Then x_i^1 and x_i^2 contain a subsequence y of length $k_i = k$. This subsequence y is also a subsequence of x^1 and x^2, so (x^1, x^2, k) is a "Yes"-instance. Vice versa, assume that (x^1, x^2, k) is a "Yes"-instance, i.e., x^1 and x^2 contain a subsequence y of length k. Let $i \in [t]$ such that the first letter of y is contained in x_i^1. We claim that all letters from y are contained in x_i^1. Since the first letter $y[1]$ of y is contained in x_i^1, no letter of y can be contained in x_j^1 for $j < i$. For $j > i$, note that (as any letter from x_j^2 only appears in x_j^2 and x_j^1) any letter from x_j^1 appears only before x_i^2 in x_2 and thus would have to appear before $y[1]$ in y, a contradiction to $y[1]$ being the first letter of y. Thus, y is contained in x_i^1 and x_i^2. Consequently, (x_i^1, x_i^2, k) is a "Yes"-instance, implying that (I_i, n_i) is a "Yes"-instance.

Computing the OR-cross-composition can be done in $t \cdot \max_{i \in t} n_i \cdot (\max_{i \in [t]} d_i)^{O(1)} = \widetilde{O}(t \max_{i \in [t]} n_i)$ time and $k = O(\max_{i \in t} n_i \cdot (\max_{i \in t} d_i)^{O(1)}) = \widetilde{O}(\max_{i=1} n_i)$ (we use here that $d_i \in O(\log n_i)$). Further, we have $n^{\mathrm{LCS}} = \sum_{i=1}^t n_i^{\mathrm{LCS}} = O(\sum_{i=1}^t n_i \cdot d_i^{O(1)}) = \widetilde{O}(\sum_{i=1}^t)$. Thus, it is an $(1,1)$-OR-cross-composition. ◀

We remark that the use of alphabets of non-constant size in the above composition is probably necessary as it excludes a linear-time computable kernel of polynomial size. In contrast, Longest Common Subsequence *with* constant alphabet size parameterized by solution size admits a linear-time computable polynomial-size kernel (note that the SETH-based $O(n^{2-\epsilon})$ lower bound also holds for binary alphabets [15]):

▶ **Proposition 4.2.** *Longest Common Subsequence with constant alphabet size parameterized by solution size k admits a linear-time computable polynomial-size kernel.*

Proof. Consider the following reduction rule which directly leads a polynomial kernel. A *substring* of a string x is a string y such that $y = (x[i], x[i+1], \ldots, x[j])$ for some $i < j$.

▶ **Reduction Rule 4.3.** If, for some $t \in \mathbb{N}$ and $i \in \{1, 2\}$, a string x^i contains a substring x' of length at least $(k+1)^t$ over only t different letters and not containing a substring x'' of length $(k+1)^{t-1}$ over at most $t-1$ different letters, then we can delete all but the first $(k+1)^t$ letters of x'.

First, we argue that the reduction rule is safe. Let $I = (x^1, x^2, k)$ be an instance of Longest Common Subsequence. We assume without loss of generality that the reduction rule can be applied to x^1, i.e., x^1 contains a substring x' of length $(k+1)^t$ which contains only t different letters, and all substrings of x' containing only $t' < t$ different letters have length at most $(k+1)^{t'}$. Let $I_{\text{red}} = (x^1_{\text{red}}, x^2, k)$ be the instance arising from applying Reduction Rule 4.3. Clearly, if I is a "No"-instance, then also I_{red} is also a "No"-instance.

Now assume that I is a "Yes"-instance, i.e., x^1 and x^2 contain a common subsequence z of length k. By assumption x' contains no substring of length $(k+1)^{t-1}$ which contains at most $t-1$ different letters. Thus, in the first $(k+1)^{t-1}-1$ letters of x', each of the t letters appears at least once. More generally, for any $i \in [k]$, each letter appears at least once among $x'[(i-1)\cdot(k+1)^{t-1}+1], \ldots, x'[i\cdot(k+1)^{t-1}]$. Consequently, each string of length k over the t appearing letters is a subsequence of x'. Hence, z is also a subsequence of x^1_{red} (and thus a common subsequence of x^1_{red} and x^2), showing that I_{red} is also a "Yes"-instance. Next, we analyze the time needed to exhaustively apply Reduction Rule 4.3.

▷ Claim 4.4. For constant alphabet size, Reduction Rule 4.3 can be exhaustively applied in linear time.

Proof. We process x^1 and x^2 from left to right. For each subset S of the alphabet, we have a variable storing the number of consecutive letters from S are before the current letter. If this number exceeds $(k+1)^{|S|}$ for some letter, then we delete this letter. Note that whenever we found a string of length $(k+1)^{|S|}$ containing only letters from S, then this string does not contain a substring of length $(k+1)^{|S|-1}$ containing only letters from $S \setminus \{\ell\}$ for some $\ell \in S$ as in this case, the number stored for $S \setminus \{\ell\}$ would have exceeded $(k+1)^{|S|-1}$.

The above procedure clearly runs in linear time as the alphabet size is constant. ◁

After the exhaustive application of Reduction Rule 4.3, the size of the instance is clearly $\mathcal{O}(k^{|\Sigma|})$. As $|\Sigma|$ is constant, this is a polynomial-sized kernel. ◀

Analogously to Longest Common Subsequence, we derive similar hardness results for the related problems Longest Common Weakly Increasing Subsequence and Longest Common Increasing Subsequence. Both problems can be solved in slightly subquadratic time [7, 23]. For Longest Common Increasing Subsequence, our lower bound was already shown by Duraj et al. [24]. We refer to Duraj et al. [24] and Duraj [23] for a broader literature review on Longest Common Subsequence.

▶ **Lemma 4.5** (⋆). *Orthogonal Vectors with logarithmic dimension $(1,1)$-OR-cross-composes into Longest Common (Weakly) Increasing Subsequence parameterized by the length n of the strings and the length k of the longest common (weakly) increasing subsequence.*

Combining Proposition 3.5 and Lemmas 4.1 and 4.5, we get the following result:

▶ **Corollary 4.6.** *Unless the SETH fails, there is no $\widetilde{O}(n^\gamma + k^\beta)$-time algorithm for* Longest Common Subsequence *or* Longest Common (Weakly) Increasing Subsequence *parameterized by solution size k for $1 < \gamma < 2$ and $\beta < \frac{\gamma}{\gamma-1}$.*

We remark that the running time lower bounds are tight; the tight upper bound follows from the known $O(kn+n\log n)$ time algorithm for Longest Common Subsequence [40] or the $O(nk \log\log n + n \log n)$ time algorithm for Longest Common (Weakly) Increasing Subsequence [45] and Observation 1.1.

4.2 Computational Geometry

We now turn to problems from computational geometry. We will denote the Euclidean distance between two points p and q in the plane by $\mathrm{dist}(p, q)$. For a list of points P, a *tour* through P is a surjective, non-increasing function $f_P : [2n] \to [n]$ such that $f_P(1) = 1$ and $f_P(2n) = n$. The input of Discrete Fréchet Distance contains two lists of points $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_n)$. A pair (f_P, f_Q) of tours through P and Q is called a *traversal*.

Discrete Fréchet Distance can be solved in $O(n^2 \cdot \log\log n/\log^2 n)$ time [6]. Bringmann [13] gave a linear-time reduction from Orthogonal Vectors[2] to Discrete Fréchet Distance, showing that assuming SETH, Discrete Fréchet Distance cannot be solved in $O(n^{2-\epsilon})$ time for any $\epsilon > 0$. We use this reduction to get a $(1, 1)$-OR-composition for the parameter $\max_{i\in[2n]} |f_P(i) - f_Q(i)|$, i.e., the maximum number of time steps which P may be ahead of Q (or vice versa) in the optimal solution. We will call the parameter $\max_{i\in[2n]} |f_P(i) - f_Q(i)|$ the *maximum shift* of the traversal (f_P, f_Q). For a more extensive literature review on Discrete Fréchet Distance, we refer to Agarwal et al. [6].

▶ **Lemma 4.7 (⋆).** *Orthogonal Vectors admits a $(1, 1)$-OR-cross-composition into Discrete Fréchet Distance parameterized by the length of the input lists of points and the maximum shift in a solution.*

Combining Proposition 3.5 and Lemma 4.7, yields the following:

▶ **Corollary 4.8.** *Unless the SETH fails, there is no $\widetilde{O}(n^\gamma + \ell^\beta)$-time algorithm Discrete Fréchet Distance parameterized by maximum shift ℓ for $1 < \gamma < 2$ and $\beta < \frac{\gamma}{\gamma-1}$.*

It is easy to extend the $O(n^2)$-algorithm for Discrete Fréchet Distance [25] to run in $O(n\ell)$ time, where ℓ is the minimal (over all solutions) maximum shift of an optimal solution [8]. This together with Observation 1.1 shows that the running time lower bound from Corollary 4.8 is tight.

We now give lower bounds for another problem from computational geometry, Planar Motion Planning, based not on the hardness of Orthogonal Vectors but on the hardness of 3-Sum. Planar Motion Planning can be solved in $O(n^2)$ time [52]. Assuming the 3-Sum conjecture, Planar Motion Planning cannot be solved in $O(n^{2-\epsilon})$ for any $\epsilon > 0$ [33]. We say a segment is in the *vicinity* of another segment if they have distance at most the length of the rod.

▶ **Lemma 4.9.** *3-Sum $(1, 1)$-OR-cross-composes into Planar Motion Planning parameterized by the maximum number of segments any segment has in its vicinity.*

[2] Bringmann [13] reduces from Satisfiability, but his reduction implicitly reduces Satisfiability to Orthogonal Vectors and then Orthogonal Vectors to Discrete Fréchet Distance.

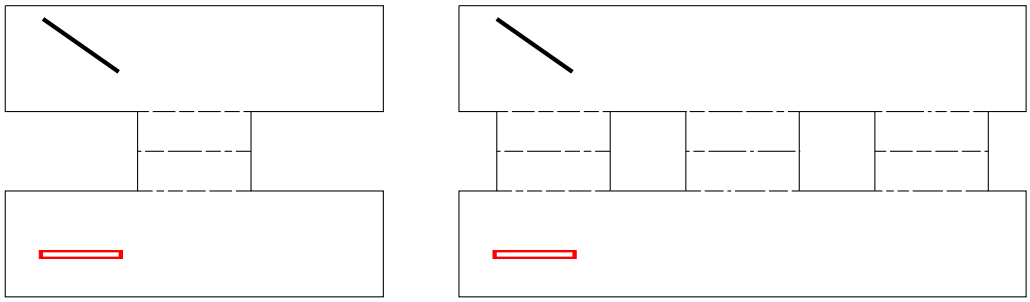


Figure 2 Left: Exemplary illustration of an instance of Planar Motion Planning constructed by the reduction from 3-Sum from Gajentaan and Overmars [33]. Right: An example for the OR-cross-composition of three instances of 3-Sum into Planar Motion Planning. The goal position of the rod is surrounded by red.

Proof sketch. Let $I_1^{\text{3-Sum}}, \ldots, I_t^{\text{3-Sum}}$ be instances of 3-Sum. We denote by n_i the number of numbers of $I_1^{\text{3-Sum}}$. Gajentaan and Overmars [33] gave a reduction from 3-Sum to Planar Motion Planning which, given an instance of 3-Sum with n numbers, constructs in $\widetilde{O}(n)$ time an instance of Planar Motion Planning where the rod initially is in a large "upper" rectangle and has to reach a "lower" rectangle through a narrow passage (see Figure 2 for a proof by picture). We apply this reduction to each instance $I_i^{\text{3-Sum}}$ of 3-Sum, giving us an instance I_i^{PMP} with $\widetilde{O}(n_i)$ many segments. From these instances, we construct an instance of Planar Motion Planning as follows: We identify the large rectangles in which the rod starts from each I_i^{PMP}. The narrow passages are copied next to each other (if the large starting rectangle is not wide enough, then we make it wider.

The correctness of the reduction is obvious.

Any segment from an instance $I_i^{\text{3-Sum}}$ has distance at most the length of the rod except for the bounding boxes. By splitting the segments of the bounding box into many small segments not longer than the rod, we get that for each segment s there are at most $O(n)$ segments whose distance to s is at most the length of the rod. ◀

Combining Proposition 3.5 and Lemma 4.9 yields the following:

▶ **Corollary 4.10.** *Unless the 3SUM-hypothesis fails, there is no $\widetilde{O}(n^\gamma + \ell^\beta)$-time algorithm Planar Motion Planning parameterized by the maximum number ℓ of segments any segment has in its vicinity for $\beta < \frac{\gamma}{\gamma-1}$.*

The lower bound for the running time is tight by the $O(K^2 \log n)$ time algorithm from Sifrony and Sharir [49] (where K is the number of segment pairs whose distance is less than the length of the rod), the observation that $K^2 \leq n \cdot \ell$, and Observation 1.1.

4.3 Graph Problems

We now turn to graph problems. First, we consider Minimum Weight k-Clique parameterized by the maximum size of a connected component.

▶ **Proposition 4.11.** *Minimum Weight k-Clique $(1,1)$-OR-cross-composes into Minimum Weight k-Clique parameterized by the maximum size of a connected component.*

Proof. Let $G_1, \ldots, G_t$ be instances of Minimum Weight k-Clique. The cross-composition just computes the disjoint union G of $G_1, \ldots, G_t$.

Clearly, the cross-composition is correct, runs in linear time, and the maximum size of a connected component is bounded by the maximum size of $G_1, \ldots, G_s$. ◀

As Negative Triangle is the special case of Minimum Weight k-Clique, combining Proposition 3.5 and Proposition 4.11 yields the following lower bound:

▶ **Corollary 4.12.** *Unless the APSP-hypothesis fails, there is no $\widetilde{O}(n^\gamma + \ell^\beta)$-time algorithm for Negative Triangle parameterized by the maximum size ℓ of a connected component for $1 < \gamma < 3$ and $\beta < \frac{2\cdot\gamma}{\gamma-1}$.*

An algorithm running in $O(\ell^2 n)$ where ℓ is the maximum size of a connected component is trivial. This together with Observation 1.1 implies that the running time lower bound is tight.

Next, we turn to 2nd Shortest Path which can be solved in $\widetilde{O}(mn)$ [46] or in $O(Mn^\omega)$ time (where M is the largest edge weight) [36]. If the graph is undirected [42] or one aims to find a 2nd shortest walk [26], then there is a quasi-linear-time algorithm. For unweighted directed graphs, the problem can be solved in $\widetilde{O}(m\sqrt{n})$ time [48]. An ϵ-approximation can be computed in $\widetilde{O}(\frac{m}{\epsilon})$ time [10].

▶ **Lemma 4.13.** *Negative Triangle $(1,1)$-OR-cross-composes into 2nd Shortest Path parameterized by directed feedback vertex number.*

Proof. Let $(I_1^{\mathrm{NT}}, n_1), \ldots, (I_t^{\mathrm{NT}}, n_t)$ be instances of Negative Triangle. We assume without loss of generality that each instance of Negative Triangle has the same number n of vertices, i.e., $n_i = n$ for all $i \in [t]$, and that the largest absolute value of an edge weight is the same for all instances. Vassilevska Williams and Williams [51] gave a linear-time reduction from Negative Triangle to 2nd Shortest Path which, given an instance $(I_i^{\mathrm{NT}} = (G_i, w_i), n)$ of Negative Triangle, creates an instance I_i^{2SP} of 2nd Shortest Path as follows: For each vertex $v \in V(G_i)$, add three vertices a^v, b^v, and c^v. Further, add two vertices s^i and t^i as well as a path $s^i, p_1^i, p_2^i, \ldots, p_n^i, t^i$, all of whose edges have weight 0. Further, there are edges (p_i, a^v), (a^v, b^u), (b^u, c^v), and (c^v, p_i) for every $i \in [n], u, v \in V(G_i)$. All these edges have positive weight (depending on the edge-weights in $E(G_i)$). Identifying s^i with s^j, t^i with t^j, and p_ℓ^i with p_ℓ^j for all $i, j \in [t]$ and $j \in [\ell]$ then results in one instance of 2nd Shortest Path with a directed feedback vertex set of size n, namely $\{p_1, \ldots, p_n\}$. As the constructed instance is equivalent to the disjunction of $I_1^{\mathrm{2SP}}, \ldots, I_t^{\mathrm{2SP}}$ (it is never beneficial to leave $s, p_1, \ldots, p_n, t$ more than once as all edges leaving this path have positive weight), we have a $(1,1)$-OR-cross-composition. ◀

Combining Proposition 3.5 with Lemma 4.13 yields the following:

▶ **Corollary 4.14.** *Unless the APSP-hypothesis fails, there is no $\widetilde{O}(n^\gamma + \ell^\beta)$-time algorithm for 2nd Shortest Path parameterized by directed feedback vertex set ℓ parameterized by the maximum size ℓ of a connected component for $1 < \gamma < 3$ and $\beta < \frac{2\gamma}{\gamma-1}$.*

In contrast to the other problems studied in this paper, we do not know whether the running time lower bounds for 2nd Shortest Path are tight.

4.4 Triangle Collection

Last, we consider the Triangle Collection problem. Triangle Collection can trivially be solved in $O(n^3)$ time, but does not admit an $O(n^{3-\epsilon})$ time algorithm assuming SETH, the 3-Sum conjecture, *or* the APSP conjecture [4]. For this problem, we were unable to apply

our framework directly, i.e., to find an OR-cross-composition from an OR-decomposable problem to Triangle Collection. However, we can still get a lower bound in a very similar fashion by combining decomposition and composition into one step. The difference is that the decomposition of Triangle Collection that we use in the proof of the following result does not admit the "OR"-property.

▶ **Proposition 4.15.** *Unless Orthogonal Vectors, 3-Sum, or APSP-hypothesis fails, there is no $\widetilde{O}(n^\gamma+\ell^\beta)$-time algorithm for Negative Triangle parameterized by the maximum size ℓ of a connected component for $1<\gamma<3$ and $\beta<\frac{2\cdot\gamma}{\gamma-1}$.*

Proof. Fix $1<\gamma<3$. Let G be in an instance of Triangle Collection. Partition $V(G)$ into $z := n^{\lambda/3+\lambda}$ sets $V_1,\dots,V_z$ of size $q := n^{3/(3+\lambda)}$, where $\lambda := \beta/\gamma - 1$. For each $(i,j,k)\in[z]^3$, let $G_{(i,j,k)}$ be the graph induced by $V_i\cup V_j\cup V_k$. Finally, let H be the union of all $G_{(i,j,k)}$. Note that H corresponds to the output of the OR-decomposition in our framework. Clearly G has a triangle collection if and only if H has. Further, H can be computed in $\widetilde{O}(q^2\cdot z^3) = \widetilde{O}(n^{3(\lambda+2)/(3+\lambda)})$ time (note that each of the z^3 graphs $G_{(i,j,k)}$ has size $O(q^2)$ as a q-vertex graph may have $O(q^2)$ many edges).

Now assume that there is a $\widetilde{O}(n^\gamma+\ell^\beta)$-algorithm for Triangle Collection with $\beta<\frac{2\cdot\gamma}{\gamma-1}$ and apply this algorithm to H. This clearly solves H. The running time for this step is $\widetilde{O}((z^3\cdot q)^\gamma + q^\beta) = \widetilde{O}(n^{\gamma\cdot(3\cdot(\lambda+1)/(3+\lambda))} + n^{\beta\cdot 3/(3+\lambda)})$ time. Note that $\gamma\cdot 3\cdot(\lambda+1)/(3+\lambda) = 3\cdot\frac{\beta/\gamma}{2+\beta/\gamma} < 3$. Further, it holds that

$$\beta\cdot 3/(3+\lambda) = 3\cdot\frac{\beta}{2+\beta/\gamma} = 3\cdot\frac{\beta\gamma}{2\gamma+\beta} = 3\cdot\frac{\gamma}{2\gamma/\beta+1} < 3\cdot\frac{\gamma}{2\gamma/\left(\frac{2\gamma}{(\gamma-1)}\right)+1} = 3\cdot\frac{\gamma}{(\gamma-1)+1} = 3$$

where we used the assumption $\beta<\frac{2\cdot\gamma}{\gamma-1}$ for the inequality. Consequently, we can solve Triangle Collection in $O(n^{3-\epsilon})$ time for some $\epsilon>0$. ◀

As an $O(n\ell^2)$ time algorithm for Triangle Collection is trivial, it follows from Observation 1.1 that the running time lower bound is tight.

5 Conclusion

We introduced a framework for extending conditional running time lower bounds to parameterized running time lower bounds and applied it to various problems. Beyond the clear task to apply the framework to further problems there are further challenges for future work. For example, can we get "AND-hard" problems so that we can use AND-cross compositions similar to the ones used to exclude compression [21]? Moreover, can the framework be adapted to cope with dynamic, counting, or enumerating problems?

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *ACM Transactions on Algorithms*, 18(1):6:1–6:22, 2022. doi:10.1145/3450524.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

4 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. *SIAM Journal on Computing*, 47(3):1098–1122, 2018. doi:10.1137/15M1050987.

5 Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. *CoRR*, abs/2012.12594, 2020. arXiv:2012.12594.

6 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014. doi:10.1137/130920526.

7 Anadi Agrawal and Pawel Gawrychowski. A faster subquadratic algorithm for the longest common increasing subsequence problem. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *LIPIcs*, pages 4:1–4:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.4.

8 Jérémy Barbay. Adaptive computation of the discrete fréchet distance. In *Proceeding of the 25th International Symposium on String Processing and Information Retrieval (SPIRE 2018)*, volume 11147 of *Lecture Notes in Computer Science*, pages 50–60. Springer, 2018. doi:10.1007/978-3-030-00479-8_5.

9 Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences*, 103:61–77, 2019. doi:10.1016/j.jcss.2019.02.004.

10 Aaron Bernstein. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 742–755. SIAM, 2010. doi:10.1137/1.9781611973075.61.

11 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.

12 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.

13 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2014)*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.

14 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS 2015)*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

15 Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1216–1235. SIAM, 2018. doi:10.1137/1.9781611975031.79.

16 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. *ACM Transactions on Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.

17 Yijia Chen, Jörg Flum, and Moritz Müller. Lower bounds for kernelizations and other preprocessing procedures. *Theory of Computing Systems*, 48(4):803–839, 2011. doi:10.1007/s00224-010-9270-y.

18 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: http://mitpress.mit.edu/books/introduction-algorithms.

19 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms*, 15(3):33:1–33:57, 2019. doi:10.1145/3310228.

20 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *Journal of the ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

21 Andrew Drucker. New limits to classical and quantum instance compression. *SIAM Journal on Computing*, 44(5):1443–1479, 2015. doi:10.1137/130927115.

22 Guillaume Ducoffe. Maximum matching in almost linear time on graphs of bounded clique-width. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*, volume 214 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.15.

23 Lech Duraj. A sub-quadratic algorithm for the longest common increasing subsequence problem. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154 of *LIPIcs*, pages 41:1–41:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.41.

24 Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10):3968–3992, 2019. doi:10.1007/s00453-018-0485-7.

25 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical report, Technische Universität Wien, 1994.

26 David Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998. doi:10.1137/S0097539795290477.

27 Henning Fernau, Till Fluschnik, Danny Hermelin, Andreas Krebs, Hendrik Molter, and Rolf Niedermeier. Diminishable parameterized problems and strict polynomial kernelization. *Computability*, 9(1):1–24, 2020. doi:10.3233/COM-180220.

28 Till Fluschnik, Christian Komusiewicz, George B. Mertzios, André Nichterlein, Rolf Niedermeier, and Nimrod Talmon. When can graph hyperbolicity be computed in linear time? *Algorithmica*, 81(5):2016–2045, 2019. doi:10.1007/s00453-018-0522-6.

29 Till Fluschnik, George B. Mertzios, and André Nichterlein. Kernelization lower bounds for finding constant-size subgraphs. In *Proceedings of the 14th Conference on Computability in Europe (CiE 2018)*, volume 10936 of *Lecture Notes in Computer Science*, pages 183–193. Springer, 2018. doi:10.1007/978-3-319-94418-0_19.

30 Fedor V. Fomin, Daniel Lokshtanov, Michal Pilipczuk, Saket Saurabh, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

31 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Kernelization: Theory of Parameterized Preprocessing*. Cambridge University Press, 2019. doi:10.1017/9781107415157.

32 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. *Journal of Computer and System Sciences*, 77(1):91–106, 2011.

33 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5:165–185, 1995. doi:10.1016/0925-7721(95)00022-2.

34 Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science*, 689:67–95, 2017. doi:10.1016/j.tcs.2017.05.017.

35 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016. doi:10.1016/j.dam.2015.10.040.

36 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Transactions on Algorithms*, 16(1):15:1–15:25, 2020. doi:10.1145/3365835.

37 Klaus Heeger, André Nichterlein, and Rolf Niedermeier. Parameterized lower bounds for problems in p via fine-grained cross-compositions. *CoRR*, abs/10.48550, 2023.

38 Falko Hegerfeld and Stefan Kratsch. On adaptive algorithms for maximum matching. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *LIPIcs*, pages 71:1–71:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.71.

39 Monika Henzinger, Alexander Noe, Christian Schulz, and Darren Strash. Finding all global minimum cuts in practice. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *LIPIcs*, pages 59:1–59:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.59.

40 Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675, 1977. doi:10.1145/322033.322044.

41 Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*, volume 96 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

42 Naoki Katoh, Toshihide Ibaraki, and Hisashi Mine. An efficient algorithm for K shortest simple paths. *Networks*, 12(4):411–427, 1982. doi:10.1002/net.3230120406.

43 Tomohiro Koana, Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. *ACM Journal of Experimental Algorithmics*, 26:1.3:1–1.3:30, 2021. doi:10.1145/3439801.

44 Stefan Kratsch and Florian Nelles. Efficient and adaptive parameterized algorithms on modular decompositions. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112 of *LIPIcs*, pages 55:1–55:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.55.

45 Martin Kutz, Gerth Stølting Brodal, Kanela Kaligosi, and Irit Katriel. Faster algorithms for computing longest common increasing subsequences. *Journal of Discrete Algorithms*, 9(4):314–325, 2011. doi:10.1016/j.jda.2011.03.013.

46 Eugene L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Sci.*, 18:401–405, 1971/72. doi:10.1287/mnsc.18.7.401.

47 George B. Mertzios, André Nichterlein, and Rolf Niedermeier. The power of linear-time data reduction for maximum matching. *Algorithmica*, 82(12):3521–3565, 2020. doi:10.1007/s00453-020-00736-0.

48 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Transactions on Algorithms*, 8(4):33:1–33:11, 2012. doi:10.1145/2344422.2344423.

49 Shmuel Sifrony and Micha Sharir. A new efficient motion-planning algorithm for a rod in two-dimensional polygonal space. *Algorithmica*, 2:367–402, 1987. doi:10.1007/BF01840368.

50 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3447–3487. World Scientific, 2018.

51 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *Journal of the ACM*, 65(5):27:1–27:38, 2018. doi:10.1145/3186893.

52 Gert Vegter. The visibility diagram: a data structure for visibility problems and motion planning. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (Swat 1990)*, volume 447 of *Lecture Notes in Computer Science*, pages 97–110. Springer, 1990. doi:10.1007/3-540-52846-6_81.

53 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

Dynamic Maintenance of Monotone Dynamic Programs and Applications

Monika Henzinger
Institute of Science and Technology Austria, Klosterneuburg, Austria

Stefan Neumann
KTH Royal Institute of Technology, Stockholm, Sweden

Harald Räcke
TU München, Germany

Stefan Schmid
TU Berlin, Germany
Fraunhofer SIT, Darmstadt, Germany

Abstract

Dynamic programming (DP) is one of the fundamental paradigms in algorithm design. However, many DP algorithms have to fill in large DP tables, represented by two-dimensional arrays, which causes at least quadratic running times and space usages. This has led to the development of improved algorithms for special cases when the DPs satisfy additional properties like, e.g., the Monge property or total monotonicity.

In this paper, we consider a new condition which assumes (among some other technical assumptions) that the *rows* of the DP table are *monotone*. Under this assumption, we introduce a novel data structure for computing $(1+\epsilon)$-approximate DP solutions in *near-linear time and space* in the static setting, and with *polylogarithmic update times* when the DP entries change dynamically. To the best of our knowledge, our new condition is incomparable to previous conditions and is the first which allows to derive *dynamic* algorithms based on existing DPs. Instead of using two-dimensional arrays to store the DP tables, we store the rows of the DP tables using monotone piecewise constant functions. This allows us to store length-n DP table rows with entries in $[0, W]$ using only $\text{polylog}(n, W)$ bits, and to perform operations, such as $(\min, +)$-convolution or rounding, on these functions in polylogarithmic time.

We further present several applications of our data structure. For bicriteria versions of *k-balanced graph partitioning* and *simultaneous source location*, we obtain the first dynamic algorithms with subpolynomial update times, as well as the first static algorithms using only near-linear time and space. Additionally, we obtain the currently fastest algorithm for *fully dynamic knapsack*.

2012 ACM Subject Classification Theory of computation → Dynamic programming; Theory of computation → Dynamic graph algorithms; Theory of computation → Packing and covering problems

Keywords and phrases Dynamic programming, dynamic algorithms, data structures

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.36

Related Version *Full Version*: https://arxiv.org/abs/2301.01744

Funding *Monika Henzinger*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.
Stefan Neumann: This research is supported by the the ERC Advanced Grant REBOUND (834862) and the EC H2020 RIA project SoBigData++ (871042).
Stefan Schmid: Research supported by Austrian Science Fund (FWF) project I 5025-N (DELTA), 2020-2024.

© Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

Dynamic programming (DP) is one of the fundamental paradigms in algorithm design. In the DP paradigm, a complex problem is broken up into simpler subproblems and then the original problem is solved by combining the solutions for the subproblems. One of the drawbacks of DP algorithms is that in practice they are often slow and memory-intensive: for inputs of size n their running time is typically $\Omega(n^2)$, and when the DP table is stored using a two-dimensional array they also need space $\Omega(n^2)$.

This motivated researchers to develop more efficient DP algorithms with near-linear time and space. Indeed, such improvements are possible under a wide range of conditions on the DP tables [1, 4, 6, 8, 12, 16, 20–22, 24], such as the Monge property, total monotonicity, certain convexity and concavity properties, or the Knuth–Yao quadrangle-inequality; we discuss these properties in more detail in the full version [17]. When these properties hold, typically one does not have to compute the entire DP table but instead only has to compute $O(n)$ DP entries which reveal the optimal solution.

However, we are not aware of any property for DPs that yields efficient *dynamic* algorithms, i.e., algorithms that provide efficient update operations when the input changes. One might find this somewhat surprising because, from a conceptual point of view, many dynamic algorithms hierarchically partition the input and maintain solutions for subproblems; this is quite similar to how many DP schemes are derived. Indeed, this conceptual similarity is exploited by many "hand-crafted" algorithms (e.g., [9, 18]) which start with a DP scheme and then show how to maintain it dynamically under input changes. However, such algorithms are often quite involved and their analysis often requires sophisticated charging schemes.

Hence, it is natural to ask whether there exists a *general criterion* which, if satisfied, guarantees that a given DP can be updated efficiently under input changes.

Our Contributions. The main contribution of our paper is the introduction of a general criterion which allows to approximate all entries of a DP table up to a factor of $1 + \epsilon$. We show that if our criterion is satisfied by a DP (with suitable parameters) then:

- In the dynamic setting, we can maintain a $(1 + \epsilon)$-approximation of the entire DP table using polylogarithmic update time (see Theorem 10).
- In the static setting, we can compute a $(1+\epsilon)$-approximation of the DP table in near-linear time and space (see Theorem 9).

Our criterion essentially asserts that the *rows* of the DP tables should be *monotone* and that the dependency graph of the DP should be a DAG, where the sets of reachable nodes are small, among some other technical conditions (see Definition 8 for the formal definition). Our criterion is incomparable to the Monge property, total monotonicity or other criteria from the literature (see the full version [17] for a more detailed discussion).

To obtain our results, we introduce a novel data structure for maintaining DPs which satisfy our criterion. Our data structure is based on the idea of storing the DP rows using *monotone piecewise constant* functions. The monotonicity of the DP rows will allow us to ensure that our functions only contain very few pieces. Then we show that we can perform operations on such functions very efficiently, with the running times only depending on the number of pieces. This is crucial because it allows us to compute an *entire* $(1+\delta)$-approximate DP row in time just $\text{polylog}(W)$, even when the DP has $\Omega(n)$ columns, assuming that the DP entries are from $[0, W]$. Note that if $W \leq \text{poly}(n)$ then this decreases the running time for computing an entire row from $\Omega(n)$ to just $\text{polylog}(n)$. Additionally, this also allows us to store each row *using only* $\text{polylog}(W)$ *space* rather than storing it in an array of size $\Omega(n)$. We present our criterion and the details of our data structure in Section 2.

As applications of our data structure, we obtain new static and dynamic algorithms for various problems. We present new algorithms for *k-balanced partitioning*, *simultaneous source location* and for *fully dynamic knapsack*. Next, we describe these results in detail; we discuss more related work in the full version [17].

Our Results for Fully Dynamic 0-1 Knapsack. First, we provide a novel algorithm for fully dynamic 0-1 knapsack. In this problem, the input consists of a knapsack size $B \in \mathbb{R}_+$ and a set of n items, where each item $i \in [n]$ has a weight $w_i \in \mathbb{R}_+$ and a price $p_i \in [1, \infty)$. The goal is to find a set of items I that maximizes $\sum_{i \in I} p_i$ while satisfying the constraint $\sum_{i \in I} w_i \leq B$. In the dynamic version of the problem, items are inserted and deleted. More concretely, we consider the following update operations: *insert*(p_i, w_i), in which the price and weight of item i are set to $p_i \in [1, \infty)$ and $w_i \in \mathbb{R}_+$, respectively, and *delete*(i), where item i is removed from the set of items.

Our main result is a dynamic $(1+\epsilon)$-approximation algorithm with worst-case update time $\epsilon^{-2} \cdot \log^2(nW) \cdot \text{polylog}(1/\epsilon, \log(nW))$, where $W = \sum_i p_i$. Our algorithm improves upon a recent result by Eberle, Megow, Nölke, Simon and Wiese [11] that also maintained a $(1+\epsilon)$-approximate solution with update time $O(\epsilon^{-9} \log^4(nW))$.

▶ **Theorem 1.** *Let $\epsilon > 0$. There exists an algorithm for fully dynamic knapsack that maintains a $(1+\epsilon)$-approximate solution with worst-case update time $\frac{1}{\epsilon^2} \log^2(nW) \,\text{polylog}\left(\frac{1}{\epsilon} \log(nW)\right)$.*

We will also show that we can return the maintained solution I in time $O(|I|)$ and that we can answer queries whether a given item $i \in [n]$ is contained in I in time $O(1)$. This matches the query times of [11].

To obtain this result, we first derive a slightly slower algorithm as a simple application of our data structure for maintaining DPs with monotone rows. Then we use this algorithm together with additional ideas to obtain the theorem (see Section 3).

Since our dynamic algorithm is based on a DP, it is possible that the solution changes significantly after each update. However, in the full version [17] we prove a lower bound, showing that every dynamic $(1+\epsilon)$-approximation algorithm for knapsack must either make a lot of changes to the solution after each update or store many (potentially substantially different) solutions between which it can switch after each update. This implies that maintaining a single explicit solution with polylogarithmic update times is not possible and the property of our algorithm cannot be avoided.

Our Results for k-Balanced Partitioning. Our most technically challenging result is for *k-balanced graph partitioning*. In this problem, the input consists of an integer k and an undirected weighted graph $G = (V, E, \text{cap})$ with n vertices, where $\text{cap} : E \to W_\infty$ is a weight function on the edges with weights in $W_\infty := [1, W] \cup \{0, \infty\}$. The goal is to find a partition $V_1, \ldots, V_k$ of the vertices such that $|V_i| \leq \lceil |V| / k \rceil$ for all i and the weight of the edges which are cut by the partition is minimized. More formally, we want to minimize $\mathit{cut}(V_1, \ldots, V_k) := \sum_{i=1}^{k} \sum_{\{u,v\} \in E \cap (V_i \times (V \setminus V_i))} \text{cap}(u, v)$.

We note that this problem is highly relevant in theory [3,13–15] and in practice [5,10,19,23], where algorithms for balanced graph partitioning are often used as a preprocessing step for large scale data analytics. Obtaining practical improvements for this problem is of considerable interest in applied communities [5] and, for instance, the popular METIS heuristic [19] has 1,400+ citations.

Since the above problem is NP-hard to approximate within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ even on trees [15], we consider bicriteria approximation algorithms. Given an undirected weighted graph $G = (V, E, \text{cap})$, a partition $V_1, \ldots, V_k$ of V is a *bicriteria* (α, β)*-approximate*

solution if $|V_i| \leq \beta\lceil n/k \rceil$ for all i and $cut(V_1, \ldots, V_k) \leq \alpha \cdot cut(\text{OPT})$, where $\text{OPT} = (V_1^*, \ldots, V_k^*)$ is the optimal solution with $|V_i^*| \leq \lceil n/k \rceil$ for all i. We note that the previously mentioned hardness result implies that any algorithm that computes a bicriteria $(\alpha, 1+\epsilon)$-approximation for any $\alpha \geq 1$ and whose running time depends only polynomially on n, must have a running time depending super-polynomially on $1/\epsilon$, unless $\mathsf{P} = \mathsf{NP}$.[1]

Our main result for the static setting is presented in the following theorem. It gives the first algorithm with polylogarithmic approximation ratio for this problem *with near-linear running time*. More concretely, we compute a bicriteria $(O(\log^4 n), 1+\epsilon)$-approximation in near-linear time for constant k. For comparison, the best approximation ratio achieved by a polynomial-time algorithm [15] is a bicriteria $(O(\log^{1.5} n \log\log n), 1+\epsilon)$-approximation with running time $\Omega(n^4)$.

▶ **Theorem 2.** *Let $\epsilon > 0$ and $k \in \mathbb{N}$. Let $G = (V, E, \text{cap})$ be an undirected weighted graph with n vertices and m edges and edge weights in W_∞. Then for the k-balanced partition problem we can compute:*

- *An $(O(\log^4 n), 1+\epsilon)$-approximation in time $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(m \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$.*[2]
- *A $(1+\epsilon, 1+\epsilon)$-approximation in time $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(n \cdot h^2 \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$ if G is a tree of height h.*
- *A $(1, 1+\epsilon)$-approximation in time $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(n^4 \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$ if G is a tree.*

Furthermore, we extend our results to the dynamic setting in which the graph G is undergoing edge insertions and deletions. In the following theorem, we present *the first dynamic algorithm with subpolynomial update time* for this problem. We again consider bicriteria approximation algorithms with update and query times depending super-polynomially on $1/\epsilon$; this cannot be avoided since if we computed $(\alpha, 1)$-approximations for any $\alpha \geq 1$ or if we had a polynomial dependency on $1/\epsilon$, then the hardness result from above implies that our update and query times must be super-polynomial in n (unless $\mathsf{P} = \mathsf{NP}$).

▶ **Theorem 3.** *Let $\epsilon > 0$ and $k \in \mathbb{N}$. Let $G = (V, E, \text{cap})$ be an undirected weighted graph with n vertices that is undergoing edge insertions and deletions. Then for the k-balanced partition problem we can maintain:*

- *An $(n^{o(1)}, 1+\epsilon)$-approximate solution with amortized update time $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot n^{o(1)} \cdot O'(\log^2(W))$ and query time $(k/\epsilon)^{O(1/\epsilon^2)}$ if G is unweighted.*
- *A $(1+\epsilon, 1+\epsilon)$-approximate solution with worst-case update time $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(h^3 \cdot \log^2(W))$ and query time $(k/\epsilon)^{O(1/\epsilon^2)}$ if G is a tree of height h.*

Our approach is inspired by the DP of Feldmann and Foschini [15]. However, the DP rows in the algorithm of [15] are not monotone and, hence, their DP cannot directly be sped up by our approach. Therefore, we first simplify and generalize the exact DP of Feldmann and Foschini to make it monotone. The DP we obtain eventually is still slightly too complex to fit into our black-box framework, but we show that the ideas from our framework can still be used to obtain the result.

Again, it is possible that the solution maintained by our algorithm changes substantially after each update. Similar to above we show in the full version [17] that this cannot be avoided when considering subpolynomial update times.

---

[1] If we had an algorithm that computes a bicriteria $(\alpha, 1+\epsilon)$-approximation in time $\text{poly}(n, 1/\epsilon)$ then we could set $\epsilon = 1/(2n)$ which implies that all partitions have size $\lceil n/k \rceil$. Thus we can compute a bicriteria $(\alpha, 1)$-approximate solution in time $\text{poly}(n)$ which contradicts the hardness result, unless $\mathsf{P} = \mathsf{NP}$.

[2] We use the notation $O'(\cdot)$ to suppress factors in $\text{poly}(\log n, k, \log(1/\epsilon), \log\log(W))$.

Our Results for Simultaneous Source Location. Next, we provide efficient algorithms for the simultaneous source location problem by Andreev, Garrod, Golovin, Maggs and Meyerson [2]. In this problem, the input consists of an undirected graph $G = (V, E, \text{cap}, d)$ with a capacity function $\text{cap}\colon E \to W_\infty$ on the edges and a demand function $d\colon V \to W_\infty$ on the vertices. The goal is to select a minimum set $S \subseteq V$ of *sources* that can simultaneously supply all vertex demands. More concretely, a set of sources S is *feasible* if there exists a flow from the vertices in S that supplies demand $d(v)$ to all vertices $v \in V$ and that does not violate the capacity constraints on the edges. The objective is to find a feasible set of sources of minimum size.

We will again consider bicriteria approximation algorithms. Let S^* be the optimal solution for the simultaneous source location problem. Then we say that S is a *bicriteria* (α, β)*-approximate solution* if $|S| \leq \alpha\,|S^*|$ and if S is a feasible set of sources when all edge capacities are increased by a factor β.

The following theorem summarizes our main results. It presents *the first near-linear time algorithm* for simultaneous source location that computes a $(1+\epsilon)$-approximate solution while only exceeding the edge capacities by a $O(\log^4 n)$ factor. In comparison, the best algorithm with arbitrary polynomial processing time computes a bicriteria $(1, O(\log^2 n \log\log n))$-approximate solution in time $\Omega(n^3)$ [2].

▶ **Theorem 4.** *Let $\epsilon > 0$. Let $G = (V, E, \text{cap}, d)$ be an undirected weighted graph with n vertices and m edges. Then for the simultaneous source location problem we can compute:*

- *A $(1+\epsilon, O(\log^4(n)))$-approximation in time*[3] $\tilde{O}(\frac{1}{\epsilon^2}m)$.
- *A $(1+\epsilon, 1)$-approximation in time $\tilde{O}(\frac{1}{\epsilon^2}h^2 \cdot n)$ if G is a tree of height h.*

Next, we turn to dynamic versions of the problem. We consider the following update operations: *SetDemand*(v, d): updates the demand of vertex v to $d(v) = d$, *SetCapacity*$((u, v), c)$: updates the capacity of the edge (u, v) to $\text{cap}(u, v) = c$, *Remove*(u, v): removes the edge (u, v), *Insert*$((u, v), c)$: inserts the edge (u, v) with capacity $\text{cap}(u, v) = c$.

We obtain *the first dynamic algorithms with subpolynomial update times* for this problem, which exceed the edge capacities only by a small subpolynomial factor.

▶ **Theorem 5.** *Let $\epsilon > 0$. Let $G = (V, E, \text{cap}, d)$ be a graph with n vertices and m edges that is undergoing the update operations given above. Then for the simultaneous source location problem we can maintain:*

- *A $(1+\epsilon, n^{o(1)})$-approximation with amortized update time $n^{o(1)}/\epsilon^2$ and preprocessing time $O(n^2/\epsilon^2)$ if all edge capacities are 1.*
- *A $(1+\epsilon, O(\log^4(n)))$-approximation with worst-case update time $\tilde{O}(1/\epsilon^2)$ and preprocessing time $\tilde{O}(m)$ if we only allow the update operation SetDemand(v, d).*
- *A $(1+\epsilon, O(\log^2(n)\log\log(n)))$-approximation with worst-case update time $\tilde{O}(1/\epsilon^2)$ and preprocessing time $\text{poly}(n)$ if we only allow the update operation SetDemand(v, d).*
- *A $(1+\epsilon, 1)$-approximate solution with worst-case update time $\tilde{O}(h^3/\epsilon^2)$ and preprocessing time $O(n^2/\epsilon^2)$ if G is a tree of height h.*

To obtain these results, we use a similar DP approach as the one used by Andreev et al. [2]. Interestingly, the DP function that we use essentially computes *the inverse function* of the one used by Andreev et al. After making these changes, the theorems become straightforward applications of our data structure for maintaining DPs with monotone rows.

[3] We write $\tilde{O}(f(n, \epsilon, W))$ to denote running times of the form $f(n, \epsilon, W) \cdot \text{polylog}(n, \epsilon, \log W)$.

Organization of Our Paper. In Section 2 we provide the details of our condition for DPs with monotone rows. In Section 3 we present our results for 0-1 Knapsack which nicely illustrate the applicability of our black-box framework from Section 2. All other results, including full proofs and a technical overview of our more involved results for k-Balanced Graph Partitioning and for Simultaneous Source Location are presented in the full version [17].

Open Problems and Future Work. In the future, it will be interesting to use our framework to obtain more dynamic algorithms based on existing DPs. We believe that this is interesting both in theory and in practice. Furthermore, it is intriguing to ask whether our criterion from Definition 8 can be generalized. Indeed, our approach was built around approximating monotone functions using piecewise constant functions, which can be viewed as piecewiese degree-0 polynomials. An interesting question is whether we can obtain a more general criterion if we approximate DP rows using pieces of higher-degree polynomials, such as splines. Results in this direction might be possible; for example, in the full version [17] we give a side result for the case when the functions contain a small number of non-monotonicities and derive a dynamic algorithm for the ℓ_∞-necklace problem.

2 Maintaining Monotone Dynamic Programming Tables

In this section, we introduce our notion of DP tables with *monotone rows* and the additional technical assumptions that we are making. Then we present our data structure for efficiently maintaining DP tables that satisfy our assumptions. In our data structure, we will store the rows of the DP using piecewise constant functions, which we will introduce first.

List Representation of Piecewise Constant Functions. Let $t \in \mathbb{R}$, $W \in [1, \infty)$ and set $W_\infty := \{0\} \cup [1, W] \cup \{+\infty\}$. A function $f\colon [0, t] \to W_\infty$ is *piecewise constant with p pieces* if there exist real numbers $0 = x_0 < x_1 < x_2 < \cdots < x_p = t$ and numbers $y_1, \ldots, y_p \in W_\infty$ such that on each interval $[x_{i-1}, x_i)$, f is constant and has value y_i. More formally, for all $i \in \{1, \ldots, p\}$ we have $f(x) = y_i$ for all real numbers $x \in [x_{i-1}, x_i)$ and $f(x_p) = y_p$. Note that we need the condition $f(x_p) = y_p$ such that f is defined on the whole domain.

In the *list representation* of a piecewise constant function f, we use a doubly linked list to store the pairs $(x_1, y_1), \ldots, (x_p, y_p)$. We also store the pairs (x_i, y_i) in a binary search tree that is sorted by the x_i-values, which allows us to compute a function value $f(x)$ in time $O(\log p)$ for all $x \in [0, t]$. In the following, we assume that all piecewise constant functions we consider are stored in the list representation with an additional binary search tree.

One of the main observations we use is that many operations on piecewise constant functions are efficient if there are only few pieces. The following lemma shows that several operations can be computed in time almost linear in the number of pieces of the function, rather than in time depending on the size of the domain of f.[4] For $\delta > 0$ and $y \in W_\infty$, we write $\lceil y \rceil_{1+\delta}$ to denote the smallest power of $1+\delta$ that is at least y, i.e., $\lceil y \rceil_{1+\delta} = \min\{(1+\delta)^i : (1+\delta)^i \geq y,\ i \in \mathbb{N}\}$; we follow the convention that $\lceil 0 \rceil_{1+\delta} = 0$ and $\lceil \infty \rceil_{1+\delta} = \infty$.

▶ **Lemma 6.** *Let $t \in \mathbb{R}$ and $c \in \mathbb{R}_+$. Let $g, h : [0, t] \to W_\infty$ be monotone and piecewise constant functions with p_g and p_h pieces, resp. Then we can compute the following functions:*

- *$f_{\min}(x) := \min\{g(x), h(x)\}$ with at most $p_g + p_h$ pieces in time $O((p_g + p_h)\log(p_g + p_h))$;*

[4] We note that computing the operations themselves can be done in linear time. However, since we also store the pairs (x_i, y_i) of the list representations in a binary search tree, the running times in the lemma include an additional logarithmic factor.

- $f_{\text{shift}}(x) := g(x-c)$ *for* $x \geq c$, $f_{\text{shift}}(x) = g(0)$ *for* $x < c$ *with at most* p_g *pieces in time* $O(p_g \log(p_g))$*;*
- $f_{\text{add}}(x) := g(x) + h(x)$*, with at most* $p_g + p_h$ *pieces in time* $O((p_g + p_h)\log(p_g + p_h))$*;*
- $f_{\text{round}}(x) := \lceil g(x) \rceil_{1+\delta}$ *for* $\delta > 0$ *with at most* $2+\lceil \log_{1+\delta}(W) \rceil$ *pieces in time* $O(p_g \log(p_g))$.

Note that if we set $\tilde{f} = \lceil f \rceil_{1+\delta}$ then $\tilde{f}$ is a $(1+\delta)$-approximation of f in the following sense. For $\alpha > 1$, we say that a function $\tilde{f} \colon [0,t] \to W_\infty$ *α-approximates* a function $f \colon [0,t] \to W_\infty$ if for all $x \in [0,t]$,

$$f(x) \leq \tilde{f}(x) \leq \alpha \cdot f(x). \tag{1}$$

Furthermore, if f is monotone then the rounded function $\tilde{f}$ contains at most $O(\log_{1+\delta}(W))$ pieces. This will be crucial later because this ensures that, if we perform a single rounding operation for each row of our DP table, the resulting function will have few pieces and operations on the function can be performed efficiently.

Next, consider functions $f_1, f_2 : [0,t] \to W_\infty$. A function $f : [0,t] \to W_\infty$ is the $(\min,+)$-*convolution* $f_1 \oplus f_2$ if for all $x \in [0,t]$, $f(x) = (f_1 \oplus f_2)(x) := \min_{\bar{x} \in [0,x]} f_1(\bar{x}) + f_2(x - \bar{x})$. Such convolutions are highly useful for the computation of many DPs. The following lemma shows that we can efficiently compute the convolution of piecewise constant functions.

▶ **Lemma 7.** *Let* $f_1, f_2 : [0,t] \to W_\infty$ *be piecewise constant functions with at most* p *pieces and assume that one of them is monotonically decreasing. Then we can compute the function* $f : [0,t] \to W_\infty$ *with* $f = f_1 \oplus f_2$ *in time* $O(p^2 \log p)$ *and* f *is a piecewise constant function with* $O(p^2)$ *pieces. Furthermore, after computing* f*, for any* $x \in [0,t]$ *we can return a value* $\bar{x}^* \in [0,t]$ *such that* $f(x) = f_1(\bar{x}^*) + f_2(x - \bar{x}^*)$ *in time* $O(\log p)$.

Now observe that Lemma 7 has a drawback for our approach: The number of pieces (i.e., the complexity of the functions) grows quadratically with every application. An important property which can be used to mitigate this issue is that the result of the convolution is still a monotone function, as we show in the full version [17]. Later, to keep the number of pieces in our functions small, after each convolution that we perform via Lemma 7 (and that might grow the number of pieces quadratically), we perform a rounding operation $\lceil \cdot \rceil_{1+\delta}$ (see Lemma 6). This loses a factor $1+\delta$ in approximation but guarantees that the resulting function has $O(\log_{1+\delta}(W))$ pieces. This will be crucial to ensure that our functions have only few pieces.

Maintaining DPs With Monotone Rows. Next, we introduce our DP scheme formally. We consider DP tables with a finite set of rows $\mathcal{I}$ and a set of columns $\mathcal{J}$, with entries taking values in W_∞. We will consider DP tables as functions $\mathsf{DP} \colon \mathcal{I} \times \mathcal{J} \to W_\infty$.[5] Further, we will associate the i'th row of the DP with a function $\mathsf{DP}(i,\cdot) \colon \mathcal{J} \to W_\infty$, and we store each such function $\mathsf{DP}(i,\cdot)$ using piecewise constant functions from above.

Next, we introduce the dependency graph for the *rows* of our DP. More concretely, the *dependency graph* $D = (\mathcal{I}, E_D)$ is a directed graph that has the rows $\mathcal{I}$ as vertices and a directed edge (i', i) between two rows if for some columns $j, j' \in \mathcal{J}$ the entry $\mathsf{DP}(i', j')$ is required to compute $\mathsf{DP}(i, j)$. We write $\text{In}(i) = \{i' \in \mathcal{I} \colon (i', i) \in E_D\}$ to denote the set of rows i' that are required to compute row i. For the rest of the paper we will assume that the dependency graph is a DAG, which is the case for all applications that we study. We will also write $\text{Reach}(i)$ to denote the set of vertices that are reachable from row i in D.

[5] Even though our definition may suggest that we only consider two-dimensional DP tables, we do not require an order on $\mathcal{I}$ and we allow $\mathcal{I}$ to be any finite set. For example, in our DP for balanced graph partitioning we will set $\mathcal{I}$ to 3-tuples corresponding to the parameters of a four-dimensional DP.

Since we assume that the dependency graph is a DAG, we can compute the i'th DP row as soon as we have computed the solutions for the DP rows in $\mathrm{In}(i)$. We assume that this is done via a *procedure* $\mathcal{P}_i$ that takes as input the DP rows $\mathsf{DP}(i', \cdot)$ for all $i' \in \mathrm{In}(i)$ and returns the row $\mathsf{DP}(i, \cdot) = \mathcal{P}_i(\{\mathsf{DP}(i', \cdot) \colon i' \in \mathrm{In}(i)\})$.

Next, we come to our condition which encodes when our scheme applies. In the definition and for the rest of the paper, we write ADP to refer to an approximate DP table, which approximates the exact DP table DP. Let $\beta > 1$. We say that ADP *β-approximates* DP if $\mathsf{DP}(i,j) \leq \mathsf{ADP}(i,j) \leq \beta \mathsf{DP}(i,j)$ for all $i \in \mathcal{I}, j \in \mathcal{J}$.

▶ **Definition 8.** *A DP table is* (h, α, p)-well-behaved *if it satisfies the following conditions:*

1. *(Monotonicity:) For all $i \in \mathcal{I}$, the function $\mathsf{DP}(i, \cdot)$ is monotone.*
2. *(Dependency graph:) The dependency graph is a DAG and $|\mathrm{Reach}(i)| \leq h$ for all $i \in \mathcal{I}$.*
3. *(Sensitivity:) Suppose $\beta > 1$ and for all $i' \in \mathrm{In}(i)$, we obtain a β-approximation $\mathsf{ADP}(i', \cdot)$ of $\mathsf{DP}(i', \cdot)$. Then applying $\mathcal{P}_i$ on the $\mathsf{ADP}(i', \cdot)$ yields a β-approximation of $\mathsf{DP}(i, \cdot)$, i.e.,*

$$\mathsf{DP}(i, \cdot) \leq \mathcal{P}_i(\{\mathsf{ADP}(i', \cdot) \colon i' \in \mathrm{In}(i)\}) \leq \beta \cdot \mathsf{DP}(i, \cdot).$$

4. *(Pieces:) For each procedure $\mathcal{P}_i$ there exists an approximate procedure $\tilde{\mathcal{P}}_i$ such that:*
   (a) $\tilde{\mathcal{P}}_i(\{\mathsf{ADP}(i', \cdot) \colon i' \in \mathrm{In}(i)\})$ is an α-approximation of $\mathcal{P}_i(\{\mathsf{ADP}(i', \cdot) \colon i' \in \mathrm{In}(i)\})$,
   (b) $\tilde{\mathcal{P}}_i$ can be computed as the composition of a constant number of operations from Lemma 6 and and at most one application of Lemma 7, and
   (c) $\tilde{\mathcal{P}}_i$ returns a monotone piecewise constant function with at most p pieces.

The definition is motivated in the following way: our operations on the piecewise constant functions have efficient running times when the functions are monotone and have few pieces. This is ensured by Properties (1), 4(b), and 4(c). Next, rounding errors cannot compound too much if each row can only reach h other rows and the sensitivity condition is satisfied. This is ensured by Properties (2), (3), and 4(a).

Even though the definition might look slightly technical at first glance, it applies in many settings. In particular, Property (2) is satisfied when the dependency graph is a rooted tree of height h in which all edges point towards the root; this is the case in all of our applications. The other conditions are immediately satisfied by our DP for 0-1 Knapsack in Section 3 and the DP for simultaneous source location (see [17]). However, our DP for balanced graph partitioning violates Property (4b) of Definition 8. Hence, in the full version [17] we will also consider a weaker assumption which, however, will not allow for nice black-box results, such as Theorems 9 and 10 below.

Next, we state our main results. They imply that we obtain static $(1+\epsilon)$-approximation algorithms running in near-linear time and space for $(\tilde{O}(1), \ln(1+\epsilon)/\tilde{O}(1), \tilde{O}(1))$-well-behaved DPs. They also show that under this assumption, we can dynamically maintain $(1+\epsilon)$-approximate DP solutions with polylogarithmic update times.

Our main theorem for static algorithms is as follows.

▶ **Theorem 9.** *Consider an (h, α, p)-well-behaved DP. Then we can compute an approximate DP table ADP which α^{h+1}-approximates DP in time and space $O(|\mathcal{I}| \cdot p^2 \log(p))$.*

Later, we will apply the theorem to DPs with dependency trees of logarithmic heights $h = O(\log n)$, we will set the approximation ratio to $\alpha = \ln(1+\epsilon)/(h+1)$, and the number of pieces to $p = \mathrm{polylog}(W)$. This will yield our desired algorithms with near-linear running time $\tilde{O}(|\mathcal{I}|)$ and space usage. Note that this is a big improvement upon the brute-force running times and space usages of $\Omega(|\mathcal{I}| \cdot |\mathcal{J}|)$.

The proof of the theorem follows from observing that when moving from one vertex to another in the dependency graph, we lose a multiplicative α-factor in the approximation ratio; as each vertex can only reach h other vertices, this will compound to at most α^{h+1}. Combining the assumptions on the functions $\tilde{\mathcal{P}}_i$ and the results from Lemmas 6 and 7, we get that each row $\mathsf{ADP}(i,\cdot)$ can be computed in time $O(p^2\log(p))$ which gives $O(|I|\cdot p^2\log(p))$ total time.

We also give the following extension to the dynamic setting which shows that if one of the DP rows changes, we can update *the entire table* efficiently.

▶ **Theorem 10.** *Consider an (h,α,p)-well-behaved DP and suppose that row i is changed. Then we can update our approximate DP table* ADP *such that after time $O(h\cdot p^2\log(p))$ it is an α^{h+1}-approximation of* DP*.*

As before, we will typically use the theorem with $h=O(\log n)$, $\alpha=\ln(1+\epsilon)/(h+1)$ and $p=\mathrm{polylog}(W)$. This will then result in our desired polylogarithmic update times. Note that this is a significant speedup compared to storing the DP tables using two-dimensional arrays: in that case even updating *a single row* would take time $\Omega(|\mathcal{J}|)$, which in many applications would already be linear in the size of the input.

The theorem follows from observing that after a row i changes, we only have to update those rows which can be reached from i in the dependency graph. But these can be at most h and each of them can be updated in time $O(p^2\log(p))$ by Lemmas 6 and 7.

3 Fully Dynamic Knapsack

In *0-1 knapsack*, the input consists of a knapsack size $B\in\mathbb{R}_+$ and a set of n items, where each item $i\in[n]$ has a weight $w_i\in\mathbb{R}_+$ and a price $p_i\in[1,\infty)$. The goal is to find a set of items I that maximizes $\sum_{i\in I}p_i$ while satisfying the constraint $\sum_{i\in I}w_i\le B$. For a set of items $I\subseteq[n]$, we refer to the sum $\sum_{i\in I}w_i$ as *the weight of I*.

For the rest of this section we set $W=\sum_i p_i$ and $t=\sum_{i\in[n]}w_i$.

Next, we first derive a dynamic algorithm with update time $\tilde{O}(\log^3(n)\log^2(W)/\epsilon^2)$ which is based on our framework for DPs with monotone rows. Then we will use this algorithm as a subroutine to obtain a faster algorithm with update time $\tilde{O}(\log^2(nW)/\epsilon^2)$ in Section 3.2; this will prove Theorem 1.

▶ **Theorem 1.** *Let $\epsilon>0$. There exists an algorithm for fully dynamic knapsack that maintains a $(1+\epsilon)$-approximate solution with worst-case update time $\frac{1}{\epsilon^2}\log^2(nW)\,\mathrm{polylog}\left(\frac{1}{\epsilon}\log(nW)\right)$.*

Below we will also show that we can return the maintained solution I in time $O(|I|)$ and that we answer queries whether a given item $i\in[n]$ is contained in I in time $O(1)$. This matches the query times of [11].

3.1 Knapsack via Convolution of Monotone Functions

First, we give a brief recap of the knapsack approach by Chan [7]. We consider the more general problem of approximating the function $f_J\colon[0,t]\to\mathbb{R}_+$, where $J\subseteq[n]$ is a set of items and

$$f_J(x)=\max\left\{\sum_{i\in I}p_i\colon\ \sum_{i\in I}w_i\le x,\ \ I\subseteq J\right\}. \tag{2}$$

Intuitively, the value $f_J(x)$ corresponds to the best possible knapsack solution if we can only pick items which are contained in J and if the weight of the solution can be *at most* x. Therefore, $f_{[n]}(B)$ corresponds to the optimum solution of the global knapsack instance.

Note that for each $J \subseteq [n]$, $f_J(x)$ is a monotonically increasing piecewise constant function: Indeed, consider $x' \leq x$. Any solution $I \subseteq J$ that is feasible for x' (i.e., the weight of I is at most x') is also a feasible solution for x. Thus, $f_J(x') \leq f_J(x)$ and, therefore, f_J is monotonically increasing. Furthermore, f_J is piecewise constant since each function value $f_J(x)$ corresponds to a solution $I \subseteq J$ and the number of choices for $I \subseteq J$ is finite.

Next, note that if we have two disjoint subsets $J_1, J_2 \subseteq [n]$ then it holds that $f_{J_1 \cup J_2}$ is the $(\max, +)$-convolution of f_{J_1} and f_{J_2}, i.e., for all x it holds that

$$f_{J_1 \cup J_2}(x) = \max_{\bar{x}} f_{J_1}(\bar{x}) + f_{J_2}(x - \bar{x}).$$

This can be seen by observing that for each x, the optimum solution I for the instance $J_1 \cup J_2$ with weight at most x can be split into two disjoint solutions $I_1 \subseteq J_1$ and $I_2 \subseteq J_2$ such that I_1 has weight $\bar{x}$ and I_2 has knapsack weight at most $x - \bar{x}$ (for suitable choice of $\bar{x} \in [0, x]$). We conclude that if we have two knapsack instances over disjoint sets of items J_1 and J_2, then we compute the solution for the knapsack instance with items $J_1 \cup J_2$ by computing the $(\max, +)$-convolution of f_{J_1} and f_{J_2}.

The Exact DP. The previous paragraphs imply a simple way of computing the exact solution of a knapsack instance: For each item $i \in [n]$, compute the function $f_{\{i\}}$ and then recursively merge the solutions for sets of size 2^j, $j = 1, \ldots, \lceil \log n \rceil$, by computing $(\max, +)$-convolutions until we have computed the global solution $f_{[n]}$. We perform the recursive merging of the solutions using a balanced binary tree, resulting in a tree of height $O(\log n)$.

More concretely, we build a rooted balanced binary tree T with n leaf nodes, where all edges point towards the root. We have one leaf $f_{\{i\}}$ for each item i. Each internal node u in T is associated with a function f_{J_u} as per Equation (2), where J_u is the set of all items in the subtree rooted at u. To simplify notation, we will also refer to f_{J_u} as f_u.

Now we consider the exact computation of the DP. This will reveal the procedures $\mathcal{P}_i$ from Definition 8. As base case, for each $i \in [n]$, the i'th leaf of T contains the function $f_{\{i\}}$, which is a piecewise constant function that has value 0 on the interval $[0, w_i)$ and value p_i on the interval $[w_i, t]$.

Next, in each internal node u of T with children u_1 and u_2, we set f_u to the $(\max, +)$-convolution of f_{u_1} and f_{u_2}. By induction it can be seen that for every node u in T, it holds that $J_u = J_{u_1} \cup J_{u_2}$ and thus J_u is the set of all items whose corresponding leaf is contained in the subtree T_u. Hence, for the root r of T it holds that $f_r = f_{[n]}$ and $f_r(B)$ is the optimal solution for the global knapsack instance.

In the following, we check that our DP satisfies Properties (1–3) of Definition 8.

First, note that the tree T from above is also the dependency graph of our DP. Hence, our DP has a row for every vertex of T and thus $O(n)$ rows in total. Furthermore, since T has height $O(\log n)$ and all edges point towards the root, every vertex can reach at most $h = O(\log n)$ vertices. Hence, Property (2) of Definition 8 is satisfied.

Second, we observe that in both cases above, the function $f_{\{i\}}$ and f_u which correspond to the rows of our DP table are monotonically increasing (we argued this above for all functions f_J). Thus, Property (1) is satisfied.

Third, observe that Property (3) is also satisfied since $(\max, +)$-convolution satisfies our sensitivity condition.

We conclude that the first three properties of Definition 8 are satisfied. Unfortunately, this does not yet imply that we can obtain efficient algorithms: Note that if we compute the exact DP bottom-up then we compute one convolution per node and thus the total running time of this approach is $O(n \cdot t(p))$, where p is an upper bound on the number of pieces in our functions and $t(p)$ is the time it takes to compute a $(\max, +)$-convolution of two functions with p pieces. However, observe that computing the convolutions can potentially take a large amount of time because the number of pieces of the functions might grow quadratically after each convolution (see Lemma 7). We will resolve this issue below using rounding.

The Approximate DP. Next, we consider approximations which will reveal the functions $\tilde{\mathcal{P}}_i$ from Definition 8.

First, note that we need to compute $(\max, +)$-convolutions of monotonically increasing functions efficiently. We observe that this can be done efficiently using our subroutine from Lemma 7 for the $(\min, +)$-convolution of monotonically decreasing functions: Indeed, suppose that f is the $(\max, +)$-convolution of two monotonically increasing functions g and h, then for all x it holds that

$$f(x) = \max_{\bar{x}}\{g(\bar{x}) + h(x - \bar{x})\} = -\min_{\bar{x}}\{-g(\bar{x}) + (-h(x - \bar{x}))\}.$$

Now observe that $-g$ and $-h$ are monotonically decreasing functions and, therefore, $f = -((-g) \oplus (-h))$, where $\oplus$ denotes the $(\min, +)$-convolution. Thus, we can use the efficient routine for $(\min, +)$-convolution from Lemma 7 with the same running time.[6]

Now we can define the subroutines $\tilde{\mathcal{P}}_i$. Let $\delta > 0$ be a parameter that we set later. Whenever we compute a function f_u via a $(\max, +)$-convolution, we use the efficient subroutine from Lemma 7. After computing the convolution, we set $f_u = \lceil f_u \rceil_{1+\delta}$ via the subroutine from Lemma 6.

Observe that this approach satisfies Property (4a) of Definition 8 with $\alpha = 1 + \delta$. Furthermore, Property (4b) is satisfied since we only use a single convolution and a single rounding step. Finally, Property (4c) is also satisfied because the resulting function is monotone and has $p = O(\log_{1+\delta}(W))$ after the rounding.

The above arguments show that our DP is (h, α, p)-well-behaved for $h = \lceil \log n \rceil$, $\alpha = 1+\delta$, $\delta = \ln(1+\epsilon)/\lceil \log n \rceil$ and $p = O(\log_{1+\delta}(W)) = O(\log(W)/\delta)$. Hence, Theorem 10 immediately implies the following lemma.

▶ **Lemma 11.** *Let $\epsilon > 0$. There exists an algorithm that computes a $(1 + \epsilon)$-approximate solution for 0-1 knapsack in time $n \cdot \frac{1}{\epsilon^2} \log^2(n) \log^2(W) \cdot \text{polylog}(\frac{1}{\epsilon} \log(nW))$.*

We note that we can return our solution I in time $|I| \log(n) \cdot \text{polylog}(\frac{1}{\epsilon} \log(nW))$ as follows. Recall that our global objective function value is achieved by $f_r(B)$ and that $f_r(B) = f_{u_1}(\bar{x}^*) + f_{u_2}(B - \bar{x}^*)$, where u_1 and u_2 are the nodes below the root node r of the dependency tree. Now using the second part of Lemma 7 we can get the value of $\bar{x}^*$ in time $O(\log p)$. If $f_{u_1}(\bar{x}^*) > 0$ we recurse on $f_{u_1}(\bar{x}^*)$ and if $f_{u_2}(B - \bar{x}^*) > 0$ we also recurse on $f_{u_2}(B - \bar{x}^*)$. At some point we will reach a leaf node i and we include i in the solution iff $f_{\{i\}}(x) > 0$. Note that since we only recurse for function values which are strictly larger than zero, for each item that we include into the solution we have to follow a single path in the dependency tree of height $O(\log n)$ and our work in each internal node is bounded by $O(\log p)$. This gives the total time of $O(|I| \log(n) \log(p))$ and our claim follows from our choice of p above.

[6] We note that, formally, Lemma 7 can only be applied on functions with non-negative values. However, this can be achieved by adding a number C to $-g$ and $-h$, which is an upper bound on the values taken by g and h, and at the end we subtract the constant function $2C$, i.e., we set $f = -((-g+C) \oplus (-h+C)) - 2C$.

Extension to the Dynamic Setting. Next, we extend our result to the dynamic setting. For the sake of simplicity, we assume that n is an upper bound on the maximum number of available items (items in S) and given to our algorithm in the beginning.[7] We consider update operations that insert and delete items from the set. More concretely, we consider the following update operations:

- *insert*(p_i, w_i), in which i is added to S by setting the price and weight of item i to $p_i \in W_\infty$ and $w_i \in \mathbb{R}_+$, respectively, and
- *delete*(i), where item i is removed from the set of items.

Our implementation is as follows. In the preprocessing phase, we build the same tree T as above and use the subroutine from above to compute the function $f_{\{i\}}$. For the operation *delete*(i), we set $p_i = 0$ and $w_i = 0$, which changes exactly one row of our DP table. For the operation *insert*(p_i, w_i), we set the price and weight of item i to p_i and w_i, resp., which again changes a single row in our DP table. After changing such a row, we recompute the global DP solution via Theorem 10. Since the DP is (h, α, p)-well-behaved with the same parameters as above, the theorem implies the following proposition.

▶ **Proposition 12.** *Let $\epsilon > 0$. There exists an algorithm for the fully dynamic knapsack problem that maintains a $(1+\epsilon)$-approximate solution with worst-case update time $\frac{1}{\epsilon^2}\log^3(n)\log^2(W) \cdot \text{polylog}\left(\frac{1}{\epsilon}\log(nW)\right)$.*

Observe that with the same procedure as for the static algorithm, we can return our solution I in time $|I|\log(n) \cdot \text{polylog}(\frac{1}{\epsilon}\log(nW))$. Furthermore, given an item $i \in [n]$, we can return whether $i \in I$ in time $\log(n) \cdot \text{polylog}(\frac{1}{\epsilon}\log(nW))$. This can be done by using the same query procedure as in the static setting, where we only recurse on the unique subtree in the depedency tree that contains the node for item i.

We note that the above proposition already improves upon the update time in the result of Eberle et al. [11] in terms of the dependency on $\frac{1}{\epsilon}$ but it has a worse dependency on $\log(nW)$. However, our query time is slower than the $O(1)$-time query operation in [11]. We will resolve these issues in the next subsection, where we will use the algorithm from Proposition 12 as a subroutine.

3.2 Dynamically Maintaining a Small Instance

Next, we we obtain a faster dynamic algorithm with update time $\tilde{O}(\frac{1}{\epsilon^2}\log^2(nW))$ by combining the algorithm from Proposition 12 and with ideas from Eberle et al. [11]. Our high-level approach is as follows. First, we partition the items into a small number of *price classes*. Then we take a few items of small weight from each price class. This will give a very small knapsack instance X for which we maintain an almost optimal solution using the subroutine from Proposition 12; since this instance is very small (i.e., $|X| \ll n$), the update time for maintaining this instance essentially becomes $O(\frac{1}{\epsilon^2}\log^2(W))$, i.e., we lose the $O(\log^3 n)$ term that made the update time in the proposition too costly. For the rest of the items which are not contained in X, we show that we can compute a good solution using fractional knapsack, which can be easily solved using a set of binary search trees. Then it remains to show that the combination of the two solutions is a $(1+\epsilon)$-approximation.

[7] It is possible to drop this assumption using an amortization argument. More concretely, every time the number of items is less than $n/2$ or more than n, we rebuild the data structure with a new value of n. Each rebuild can be done in time $O(nt(n))$, where $t(n)$ is our update time. Since this only happens after $\Omega(n)$ updates occured, we can amortize this cost over the updates that appeared since the last rebuild.

The main differences of our algorithm and the one by Eberle et al. [11] are as follows. Eberle et al. also partition the items into a small number of price classes. They also combine solutions for a small set of heavy items X and solutions based on fractional knapsack for the other items. However, they have to enumerate many different sets X and they also guess the approximate price of the fractional knapsack solution; more concretely, they enumerate $\Theta(\frac{1}{\epsilon^2}\log(W))$ choices for X and the number of guesses they have to make for the fractional knapsack solution is $\Theta(\frac{1}{\epsilon}\log(W))$. Thus they have to consider $\Theta(\frac{1}{\epsilon^3}\log^2(W))$ guesses and for each of them they have to compute approximate solutions, which takes time $\Theta(\frac{1}{\epsilon^4})$ for each X since they have to run a static algorithm from scratch. In our approach, we only have to consider a single set X which we maintain in our data structure from Proposition 12, which saves us a lot of time. Furthermore, the piecewise constant function, in which we store the solution for X, essentially "guides" our $\Theta(\frac{1}{\epsilon}\log(W))$ guesses for the weight of fractional knapsack solution. In our analysis we have to be slightly more careful to ensure that our guesses for the weight of the fractional knapsack solution guarantee the correct approximation ratio.

Definitions. We assume that $\epsilon < 1$ and that $1/\epsilon$ is an integer. More concretely, we run the algorithm with $\epsilon' = \max\{\frac{1}{i} : \frac{1}{i} \leq \epsilon, i \in \mathbb{N}\}$. Set $L = \lceil \log_{1+\epsilon}(W) \rceil$ and recall that we set $W = \sum_i p_i$.

We define the *price classes* $V_\ell = \{i : (1+\epsilon)^\ell \leq p_i < (1+\epsilon)^{\ell+1}\}$. In the following, we assume that all items from price class V_ℓ have price exactly $(1+\epsilon)^{\ell+1}$. We only lose a factor of $1+\epsilon$ by making this assumption. Furthermore, we set $V_\ell^{1/\epsilon}$ to the set of $1/\epsilon$ items from V_ℓ with smallest weights w_i (breaking ties arbitrarily). We also define $V'_\ell = V_\ell \setminus V_\ell^{1/\epsilon}$.

Next, we set $X = \bigcup_{\ell \geq 0} V_\ell^{1/\epsilon}$ and $Y = \bigcup_{\ell \geq 0} V'_\ell$ for all $\ell \geq 0$. Note that X and Y partition the set of items and $|X| = \frac{1}{\epsilon} \cdot L = O(\epsilon^{-2}\log(W))$.

Now our strategy is to use our algorithm from Proposition 12 to maintain a solution for the items in X. Then we show how we can combine the solution for X with a solution for Y that is based on fractional knapsack and a charging argument.

Data Structures. For each $\ell \in [L]$, we maintain V_ℓ sorted non-decreasingly by weight.

We also maintain the set X in a binary search tree, in which we sort the items by their index, and we maintain our data structure from Proposition 12 on the items in X.

Furthermore, let $U_\ell = \bigcup_{\ell' \leq \ell} V'_{\ell'}$ denote the set of all items that are not contained in X and of price class at most ℓ. For each ℓ, we maintain the set U_ℓ in a binary search tree T in which the items are stored as leaves and sorted by their density $\frac{p_i}{w_i}$. In each internal node u of T, we store the total weight of the items in the subtree T_u rooted at u and the total profit of the items in T_u. Observe that this allows us to answer queries of the type: "Given a budget b, what is the value of the optimal fractional[8] knapsack solution in U_ℓ with weight at most b?" in time $O(\log n)$.

Updates. Now consider an item insertion or deletion and suppose that the updated item is of price class V_ℓ. We first update the sets V_ℓ, $U_{\ell'}$ for $\ell' \leq \ell$ and the sets X and Y. Note that for each of these sets at most one item can be removed and inserted. Thus, these steps can be done in time $O(\ell \cdot \log(n)) = O(\epsilon^{-1}\log(W)\log(n))$.

[8] In fractional knapsack, we may use items fractionally. An optimal solution is achieved by sorting the items items by their density and greedily adding items to the solution until we have used up our budget b. This approach uses at most one item fractionally (namely, the one at which we use up our budget).

Next, if X changed in the previous step, then we also perform the corresponding updates in the data structure from Proposition 12. Since $|X| = O(\epsilon^{-2}\log(W))$ holds by construction of X, the update operations for the data structure maintaing the knapsack solution for X take a total time of

$$O\left(\epsilon^{-2}\log^3(|X|)\log^2(W)\cdot\operatorname{polylog}\left(\frac{1}{\epsilon}\log(|X|\,W)\right)\right)$$
$$= O\left(\epsilon^{-2}\log^2(W)\cdot\operatorname{polylog}\left(\frac{1}{\epsilon}\log(nW)\right)\right).$$

Furthermore, we can explicitly write down our solution I_X for the items in X in time $\epsilon^{-2}\log(W)\cdot\operatorname{polylog}(\frac{1}{\epsilon}\log(nW))$ since $|X| = O(\epsilon^{-2}\log(W))$. Also, for each $i \in I_X$, we can set a bit indicating that $i \in I_X$. Note that the time for writing down I_X and setting the bits is subsumed by the update time above.

Queries. *Returning the value of a solution:* We return the value of a global knapsack solution as follows.

Consider the data structure from Proposition 12 which maintains a solution for the items in X. Note that this solution is stored as a piecewise constant function with $p \leq L$ pieces and consider the list representation $(x_1, y_1), \ldots, (x_p, y_p)$ of this function.

Our strategy is as follows: For each $i = 1, \ldots, p$, we consider a solution which spends budget x_i on items in X and budget $B - x_i$ on items in Y. Then we take the maximum over all of the solutions we have considered. More concretely, for given $i = 1, \ldots, p$, we obtain our solution as follows. We pick ℓ_i such that $(1+\epsilon)^{\ell_i} = \lceil \epsilon \cdot y_i \rceil_{1+\epsilon}$ (see Lemma 13 below for a justification of this choice). Now we use the binary search tree for U_{ℓ_i} to find the highest profit that we can obtain from fractional knapsack on items in $U_{\ell_i} \subseteq Y$ if we can spend budget at most $b = B - x_i$. Let y_i' be the value of this query after removing any profit that we gain from the (at most one) fractionally cut item. We also store the density of the final item that is contained in the fractional knapsack solution. Now we return the maximum of $y_i + y_i'$ over all $i = 1, \ldots, p$.

Note that since the solution for X has at most $L = O(\epsilon^{-1}\log(W))$ pieces and for each of them we perform a single query in a binary search tree, the total time for return the solution value is $O(\epsilon^{-1}\log(W)\log(n))$. Note that this time is subsumed by the update time.

Returning the entire solution: Now we can return our global solution I in time $O(|I|)$ as follows. Observe that I is composed of the solution I_X for the items in X and of the items in the fractional knapsack solution. During our updates, we already stored the items in I_X and can write them down in time $O(|I_X|)$. Next, to return the items from the fractional knapsack solution, recall that we stored the density of the final item in the fractional knapsack solution. Thus, we only have to output the items ordered non-decreasingly by their density, while we are above the desired density-threshold. This can be done in time linear in the size of the fractional knapsack solution. This is essentially the same query procedure as in [11].

Returning whether an item is in the solution: Furthermore, observe that the above implies that we can answer whether an item $i \in [n]$ is contained in our solution in time $O(1)$: If $i \in X$ then we already stored a bit whether $i \in I_X$. If $i \notin X$ then we can check whether i is in the fractional knapsack solution by checking whether its density is above or below the threshold given by the final item in the fractional knapsack solution.

Analysis. We start by making some simplifications to OPT. We let OPT' denote the version of OPT in which for each $\ell \in [L]$, we pick the $|\mathrm{OPT} \cap V_\ell|$ items of smallest weight from V_ℓ. This only loses a factor of $1+\epsilon$. Next, define $\mathrm{OPT}'_X = \mathrm{OPT}' \cap X$ and $\mathrm{OPT}'_Y = \mathrm{OPT}' \cap Y$. Observe that by how we picked OPT', it holds that $\mathrm{OPT}'_Y \cap V_\ell \neq \emptyset$ iff $\left|\mathrm{OPT}' \cap V_\ell\right| > 1/\epsilon$.

Let p_X denote the total price of items in OPT'_X and let w_X denote the total weight of the items in OPT'_X. Let f denote the piecewise constant function that stores the solution for the items in X. Observe that by Proposition 12 we have that

$$p_X \leq f(w_X) \leq (1+\epsilon)p_X.$$

Also, the function value $f(w_X)$ is part of a piece (x_{i^*}, y_{i^*}) with $x_{i^*} \leq w_X$ and $y_{i^*} = f(w_X)$.

The next lemma justifies why we set ℓ_i such that $(1+\epsilon)^{\ell_i} = \lceil \epsilon \cdot y_i \rceil_{1+\epsilon}$ in our algorithm. To this end, let ℓ_{i^*} be such that $(1+\epsilon)^{\ell_{i^*}} = \lceil \epsilon \cdot y_{i^*} \rceil_{1+\epsilon}$ and let ℓ_Y be the price class of the most valuable item in OPT'_Y. In the lemma we show that $\ell_{i^*} \geq \ell_Y$. We will use this to show that our solution for X of profit y_{i^*} is valuable enough such that we can charge a fractionally cut item from fractional knapsack onto the solution from X and only lose a factor of $(1+\epsilon)^2$.

▶ **Lemma 13.** *It holds that $\ell_{i^*} \geq \ell_Y$.*

Proof. Since $\mathrm{OPT}'_Y \cap V'_{\ell_Y} \neq \emptyset$, $\left|\mathrm{OPT}' \cap V_{\ell_Y}\right| > 1/\epsilon$ and thus OPT'_X contains all $1/\epsilon$ items from $V^{1/\epsilon}_{\ell_Y}$. Hence, $p_X \geq \frac{1}{\epsilon} \cdot (1+\epsilon)^{\ell_Y}$. From above we get $f(w_X) = y_{i^*}$ and $f(w_X) \geq p_X$. By choice of ℓ_{i^*},

$$(1+\epsilon)^{\ell_{i^*}} = \lceil \epsilon \cdot y_{i^*} \rceil_{1+\epsilon} = \lceil \epsilon \cdot f(w_X) \rceil_{1+\epsilon} \geq \lceil \epsilon \cdot p_X \rceil_{1+\epsilon} \geq \left\lceil \epsilon \cdot \frac{1}{\epsilon}(1+\epsilon)^{\ell_Y} \right\rceil_{1+\epsilon} = (1+\epsilon)^{\ell_Y}.$$

This implies $\ell_{i^*} \geq \ell_Y$. ◀

Next, consider the the fractional knapsack solution that we obtain from our query. Note that this solution has a profit that is at least as large as the profit of OPT'_Y (since fractional knapsack is a relaxation of 0-1 knapsack). Furthermore, the fractional solution uses at most one item fractionally and this item is from $U_{\ell_{i^*}}$ and has value at most $(1+\epsilon)^{\ell_{i^*}} = \lceil \epsilon \cdot y_{i^*} \rceil_{1+\epsilon} \leq (1+\epsilon)\epsilon \cdot y_{i^*}$. Thus, we can charge this item on OPT'_X and lose a factor of at most $(1+\epsilon)^2$.

We conclude that the solution $y_{i^*} + y'_{i^*}$ is a $(1+\epsilon)^{O(1)}$-approximation of OPT. Combining this with our previous running time analysis, we obtain Theorem 1.

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

2 Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce M. Maggs, and Adam Meyerson. Simultaneous source location. *ACM Trans. Algorithms*, 6(1):16:1–16:17, 2009.

3 Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

4 Wolfgang W. Bein, Mordecai J. Golin, Lawrence L. Larmore, and Yan Zhang. The knuth–yao quadrangle-inequality speedup is a consequence of total monotonicity. *ACM Trans. Algorithms*, 6(1):17:1–17:22, 2009.

5 Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. Springer, 2016.

6 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of monge properties in optimization. *Discret. Appl. Math.*, 70(2):95–161, 1996.

7 Timothy M. Chan. Approximation schemes for 0-1 knapsack. In *SOSA*, volume 61, pages 5:1–5:12, 2018.

8 Timothy M. Chan. Near-optimal randomized algorithms for selection in totally monotone matrices. In *SODA*, pages 1483–1495, 2021.

9 Spencer Compton, Slobodan Mitrovic, and Ronitt Rubinfeld. New partitioning techniques and faster algorithms for approximate interval scheduling. *CoRR*, abs/2012.15002, 2020. arXiv:2012.15002.

10 Yihe Dong, Piotr Indyk, Ilya P. Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *ICLR*, 2020.

11 Franziska Eberle, Nicole Megow, Lukas Nölke, Bertrand Simon, and Andreas Wiese. Fully Dynamic Algorithms for Knapsack Problems with Polylogarithmic Update Time. In *FSTTCS*, volume 213, pages 18:1–18:17, 2021.

12 David Eppstein, Zvi Galil, and Raffaele Giancarlo. Speeding up dynamic programming. In *FOCS*, pages 488–496. IEEE Computer Society, 1988.

13 Guy Even, Joseph Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. *SIAM J. Comput.*, 28(6):2187–2214, 1999.

14 Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002.

15 Andreas Emil Feldmann and Luca Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, 2015.

16 Zvi Galil and Kunsoo Park. Dynamic programming with convexity, concavity, and sparsity. *Theor. Comput. Sci.*, 92(1):49–76, 1992.

17 Monika Henzinger, Stefan Neumann, Harald Räcke, and Stefan Schmid. Dynamic maintenance of monotone dynamic programs and applications. *CoRR*, abs/2301.01744, 2024. Full version of this paper. arXiv:2301.01744.

18 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In *SoCG*, volume 164 of *LIPIcs*, pages 51:1–51:14, 2020.

19 George Karypis and Vipin Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, 1997.

20 Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

21 Gary L. Miller, Richard Peng, Russell Schwartz, and Charalampos E. Tsourakakis. Approximate dynamic programming using halfspace queries and multiscale monge decomposition. In *SODA*, pages 1675–1682, 2011.

22 Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Mem. Math. Phys. Acad. Royale Sci.*, pages 666–704, 1781.

23 Peter Sanders and Christian Schulz. Think locally, act globally: Highly balanced graph partitioning. In *SEA*, volume 7933, pages 164–175, 2013.

24 F. Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *STOC*, pages 429–435, 1980.

Approximate Selection with Unreliable Comparisons in Optimal Expected Time

Shengyu Huang
Department of Computer Science, EPFL, Lausanne, Switzerland

Chih-Hung Liu
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

Daniel Rutschmann
Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Copenhagen, Denmark

Abstract

Given n elements, an integer $k \leq \frac{n}{2}$ and a parameter $\varepsilon \geq \frac{1}{n}$, we study the problem of selecting an element with rank in $(k - n\varepsilon, k + n\varepsilon]$ using *unreliable* comparisons where the outcome of each comparison is incorrect independently with a constant error probability, and multiple comparisons between the same pair of elements are independent. In this fault model, the fundamental problems of finding the minimum, selecting the k-th smallest element and sorting have been shown to require $\Theta\big(n \log \frac{1}{Q}\big)$, $\Theta\big(n \log \frac{k}{Q}\big)$ and $\Theta\big(n \log \frac{n}{Q}\big)$ comparisons, respectively, to achieve success probability $1 - Q$ [9]. Considering the increasing complexity of modern computing, it is of great interest to develop approximation algorithms that enable a trade-off between the solution quality and the number of comparisons. In particular, approximation algorithms would even be able to attain a sublinear number of comparisons. Very recently, Leucci and Liu [23] proved that the approximate minimum selection problem, which covers the case that $k \leq n\varepsilon$, requires expected $\Theta(\varepsilon^{-1} \log \frac{1}{Q})$ comparisons, but the general case, i.e., for $n\varepsilon < k \leq \frac{n}{2}$, is still open.

We develop a randomized algorithm that performs *expected* $O(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$ comparisons to achieve success probability at least $1 - Q$. For $k = n\varepsilon$, the number of comparisons is $O(\varepsilon^{-1} \log \frac{1}{Q})$, matching Leucci and Liu's result [23], whereas for $k = n/2$ (i.e., approximating the median), the number of comparisons is $O(\varepsilon^{-2} \log \frac{1}{Q})$. We also prove that even in the absence of comparison faults, any randomized algorithm with success probability at least $1 - Q$ performs *expected* $\Omega\big(\min\{n, \frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q}\}\big)$ comparisons. As long as n is large enough, i.e., when $n = \Omega(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$, our lower bound demonstrates the optimality of our algorithm, which covers the possible range of attaining a sublinear number of comparisons. Surprisingly, for constant Q, our algorithm performs expected $O(\frac{k}{n}\varepsilon^{-2})$ comparisons, matching the best possible approximation algorithm *in the absence of computation faults*. In contrast, for the exact selection problem, the expected number of comparisons is $\Theta(n \log k)$ with faults versus $\Theta(n)$ without faults. Our results also indicate a clear distinction between approximating the minimum and approximating the k-th smallest element, which holds even for the high probability guarantee, e.g., if $k = \frac{n}{2}$, $Q = \frac{1}{n}$ and $\varepsilon = n^{-\alpha}$ for $\alpha \in (0, \frac{1}{2})$, the asymptotic difference is almost quadratic, i.e., $\tilde{\Theta}(n^{\alpha})$ versus $\tilde{\Theta}(n^{2\alpha})$.

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

Keywords and phrases Approximate Selection, Unreliable Comparisons, Independent Faults

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.37

Related Version *Full Version*: `https://arxiv.org/abs/2205.01448` [17]

Funding *Chih-Hung Liu*: Yushan Young Fellow Program by Ministry of Education, Taiwan and Research Project 111-2222-E-002-017-MY2 by National Science and Technology Council, Taiwan.

Acknowledgements The three authors began to investigate this topic when all of them were in ETH Zürich, Switzerland.

© Shengyu Huang, Chih-Hung Liu, and Daniel Rutschmann;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

We study a generalization of the *k-th smallest element selection* problem in terms of *approximation* and *fault tolerance*. Given a set S of n elements, an integer k and a parameter ε, the *fault-tolerant ε-approximate k-selection* problem, FT-APX(k, ε) for short, is to return an element with rank in $(k - n\varepsilon, k + n\varepsilon]$ only using *unreliable* comparisons whose outcome can be *incorrect*. Due to the comparison faults, it is impossible to guarantee a correct solution, so the number of comparisons performed by an algorithm should depend on the *failure probability* Q of the algorithm where $Q < \frac{1}{2}$. We assume that $k \leq \frac{n}{2}$ and $\varepsilon \geq \frac{1}{n}$; if $k > \frac{n}{2}$, the problem becomes to approximate the $(n-k)$-th largest element, which is symmetric, and if $\varepsilon < \frac{1}{n}$, the problem becomes the exact selection problem. The elements with rank in $(0, k - n\varepsilon]$, $(k - n\varepsilon, k + n\varepsilon]$ and $(k + n\varepsilon, n]$ of S are called *small*, *relevant* and *large*, respectively.

We consider *independent random comparison faults*: There is a strict ordering relation among S, but algorithms can only gather information via unreliable comparisons between two elements. The outcome of each comparison is wrong with a known constant probability $\boldsymbol{p} < \frac{1}{2}$. When comparing the same pair of elements multiple times, each outcome is *independent* of the previous outcomes; comparisons involving different pairs of elements are also independent.

The above fault model has been widely studied for various fundamental problems such as finding the minimum, selecting the k-th smallest (resp. largest) element and sorting a sequence [9, 29, 30]. Feige et al [9] proved that to achieve success probability $1 - Q$, the aforementioned three problems require $\Theta\big(n \log \frac{1}{Q}\big)$, $\Theta\big(n \log \frac{k}{Q}\big)$ and $\Theta\big(n \log \frac{n}{Q}\big)$ comparisons, respectively, both in *expectation* and in the *worst case*. In the sequel, their selection algorithm is denoted by Select(k, Q), and its performance is summarized as follows.

▶ **Theorem 1** ([9]). Select(k, Q) *performs* $O\big(n \log \frac{k}{Q}\big)$ *comparisons to select the k-th smallest (resp. largest) element among n elements with success probability at least* $1 - Q$.

Due to the increasing complexity of modern computing, error detection and error correction require enormous computing resources. Emerging technologies enable the tolerance of computation errors for saving computing resources [28, 16, 7, 19, 32]. Meanwhile, many practical applications do not require an optimal answer, but just a good enough one. These circumstances motivate the study of fault-tolerant approximation algorithms, especially for the possibility of attaining a sublinear number of comparisons.

Recently, Leucci and Liu [23] studied the approximate minimum selection problem, which asks for one element with rank in $[1, n\varepsilon]$ and thus is equivalent to FT-APX(k, ε) with $k = 0$ (since FT-APX(k, ε) seeks one element with rank in $(k - n\varepsilon, k + n\varepsilon]$ under our formulation). They developed an algorithm using *expected* $O(\varepsilon^{-1} \log \frac{1}{Q})$ comparisons and also proved a matching lower bound. Moreover, if $k \leq n\epsilon$, a correct answer to FT-APX$(0, \varepsilon)$ is also correct to FT-APX(k, ε), indicating that this case is essentially the approximate minimum selection. As a result, the challenge is to tackle the case that $n\epsilon < k \leq \frac{n}{2}$.

A straightforward approach to the FT-APX(k, ε) problem is to first *randomly* pick $m = \Theta(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$ elements so that the underlying $\lceil m \cdot \frac{k}{n} \rceil$-th smallest element is *relevant* with probability at least $1 - \frac{Q}{2}$, and then apply Select$(\lceil m \cdot \frac{k}{n} \rceil, \frac{Q}{2})$ on the m elements. By Theorem 1, this approach requires $\Theta\big(\frac{k}{n}\varepsilon^{-2}\big((\log \frac{1}{Q})(\log \frac{k}{n}\varepsilon^{-2}) + \log^2 \frac{1}{Q}\big)\big)$ comparisons.

▶ Remark 2. Using standard sampling techniques, e.g., similar to Leucci and Liu's ideas [23], the number of comparisons in the above straightforward approach can easily be improved to $O\big(\frac{k}{n}\varepsilon^{-2}(\log \frac{1}{Q})(\log \frac{k}{n}\varepsilon^{-2}) + \log^2 \frac{1}{Q}\big)$. However, for constant Q, this number remains $O\big(\frac{k}{n}\varepsilon^{-2}(\log \frac{k}{n}\varepsilon^{-2})\big)$, and there is no obvious way of improving it to $O(\frac{k}{n}\varepsilon^{-2})$. Remark 8 in Section 4 will discuss how variants of Quickselect run into a similar issue.

Table 1 Summary for the known results and our new results.

	Minimum	k-th Element
Exact	$\Theta(n \log \frac{1}{Q})$ [9]	$\Theta(n \log \frac{k}{Q})$ [9]
Approximate	$\Theta(\varepsilon^{-1} \log \frac{1}{Q})$ [23]	$O(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$ [ours] $\Omega\big(\min\{n, \frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q}\}\big)$ [ours]
Exact without faults	$\Theta(n)$	$\Theta(n)$
Approximate without faults	$\Theta\big(\min\{n, \varepsilon^{-1} \log \frac{1}{Q}\}\big)$	$\Theta\big(\min\{n, \frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q}\}\big)$ [ours]

To sum up, it is of great interest to study if the FT-APX(k, ε) problem can be solved with probability $1 - Q$ using $O(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$ comparisons. Moreover, since comparison faults increase the number of comparisons required for the exact selection problem, i.e., from $\Theta(n)$ without faults to $\Theta(n \log \frac{k}{Q})$ with faults, which shows a clear gap even for constant Q, one may wonder if the same phenomenon occurs for the approximate selection problem. Furthermore, although finding the minimum and finding the k-th smallest element require different numbers of comparisons, namely $\Theta(n \log \frac{1}{Q})$ versus $\Theta(n \log \frac{k}{Q})$, to attain a high success probability, i.e., $Q = \frac{1}{n}$, both problems require $\Theta(n \log n)$ comparisons. Hence, it is also desirable to investigate if there is a stronger distinction between these two problems in the approximation scenario, especially for the high success probability.

1.1 Our Contributions

We develop a randomized algorithm that performs *expected* $O(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$ comparisons to solve the FT-APX(k, ε) problem with probability at least $1 - Q$. Also, we prove that even without considering comparison faults, any randomized algorithm with success probability $1 - Q$ requires *expected* $\Omega\big(\min\{n, \frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q}\}\big)$ comparisons. As long as n is large enough, i.e., when $n = \Omega(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$, our lower bound demonstrates the optimality of our algorithm, which covers the possible range of attaining a sublinear number of comparisons. Table 1 summarize the known results and our new results.

Furthermore, for any constant Q, e.g., $Q = 1/4$, our algorithm performs expected $O(\frac{k}{n}\varepsilon^{-2})$ comparisons, which is optimal even in the absence of comparison faults. This surprising outcome distinguishes the approximate selection problem from the exact selection problem, where the exact selection problem requires expected $\Theta(n)$ comparisons without faults, but expected $\Theta(n \log k)$ comparisons with faults.

Moreover, our results also indicate that there is a distinction between the approximate *minimum* selection problem and the general approximate *k-th element* selection problem in terms of the expected number of comparisons, i.e., $\Theta(\varepsilon^{-1} \log \frac{1}{Q})$ [24] versus $\Theta(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$. This distinction even holds for the high probability guarantee ($Q = \frac{1}{n}$) in contradiction to the fact that the two problems have the same complexity $\Theta(n \log n)$ in the *exact* selection (Theorem 1). For example, if $k = \frac{n}{2}$ and $Q = \frac{1}{n}$, the two approximate selection problems require expected $\Theta(\varepsilon^{-1} \log n)$ and $\Theta(\varepsilon^{-2} \log n)$ comparisons, respectively. If $\varepsilon = n^{-\alpha}$ for a constant $\alpha \in (0, \frac{1}{2})$, the asymptotic difference is almost quadratic, i.e., $\tilde{\Theta}(n^{\alpha})$ versus $\tilde{\Theta}(n^{2\alpha})$.

▶ Remark 3. The $\frac{k}{n}\varepsilon^{-2}$ term in the above complexities is actually $\max\{\varepsilon^{-1}, \frac{k}{n}\varepsilon^{-2}\}$. As discussed before, the case that $k \le n\epsilon$, by which $\varepsilon^{-1} \ge \frac{k}{n}\varepsilon^{-2}$, belongs to the approximate minimum selection problem. Therefore, to simplifying the description, we assume that $k > n\epsilon$ throughout the paper if no further specification.

As noted in Remark 2, our technical advance is to improve the $\frac{k}{n}\varepsilon^{-2}(\log\frac{1}{Q})(\log\frac{k}{n}\varepsilon^{-2}) + \log^2\frac{1}{Q}$ term to $\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q}$. To some extent, compared with Leucci and Liu's algorithms, our algorithm covers the entire range of k instead of the case when k is trivially small. In addition, our algorithm owns an elegant feature that it only exploits simple sampling techniques, e.g., selecting the median of three samples and selecting the minimum of two samples.

The top-level of our algorithms, inspired by Leucci and Liu [23], reduces the FT-APX(k, ε) problem on n elements to the FT-APX$(\frac{m}{2}, \frac{3}{8})$ problem on $m = \Theta(\log\frac{1}{Q})$ elements. More precisely, if a relevant element can be selected with probability $\frac{8}{9}$, we can generate a sequence of $\Theta(\log\frac{1}{Q})$ elements in which $\frac{3}{4}$ of elements around the middle, with probability $1 - \frac{Q}{2}$, are all relevant. For such a "dense" sequence, we design a delicate trial-and-error method to select a relevant element with probability $1 - \frac{Q}{2}$ using expected $\Theta(\log\frac{1}{Q})$ comparisons.

The main challenge is to obtain a relevant element with probability $\frac{8}{9}$ using only $O(\frac{k}{n}\varepsilon^{-2})$ comparisons. For the approximate minimum ($k = 0$), Leucci and Liu [23] applied Select$(1, \frac{1}{10})$ on $\Theta(\varepsilon^{-1})$ randomly picked elements and attained $O(\varepsilon^{-1})$ comparisons. However, for general k, this method requires $\Theta(\frac{k}{n}\varepsilon^{-2}\log\frac{k}{n}\varepsilon^{-2})$ comparisons with an extra logarithmic factor.

We first work on a special case that $k = \frac{n}{2}$, i.e., the approximate median selection. Based on the symmetry property of the median, we observe that the median of three randomly picked elements is more likely to be relevant than a randomly picked element. We exploit this observation to iteratively increase the ratio of relevant elements while keeping the underlying median being relevant. Once the ratio becomes a constant fraction, we will apply a straightforward method.

For general k, we design a "purifying" process that iteratively increases the ratio of relevant elements while keeping elements around a "controlled" position being relevant. Despite no symmetry property, we still observe that under certain conditions, the minimum of two randomly picked elements is more likely to be relevant than a randomly picked one. Then, we derive feasible parameters to control the relative position of k, i.e., the middle of the remaining relevant elements, during the purifying process. Once the relative position becomes a constant fraction of the remaining elements, we add dummy smallest elements and apply our approximate median selection.

As a by-product, we also give a randomized algorithm using *deterministic* $O\big(\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q} + (\log\frac{1}{Q})(\log\log\frac{1}{Q})^2\big)$ comparisons, and it is still open how to attain deterministic $O\big(\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q}\big)$ comparisons. Besides, when $\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q} = \omega(n)$, we derive another lower bound of $\Omega(\max\{n, \varepsilon^{-1}\log\frac{(k+n\varepsilon)/(2n\varepsilon)}{Q}\})$ (Theorem 21). In this situation, a trivial upper bound of $O(n\log\frac{k}{Q})$ follows from Theorem 1. It is also not clear how to fill this gap between the lower and the upper bounds, i.e., $\Omega(\max\{n, \varepsilon^{-1}\log\frac{(k+n\varepsilon)/(2n\varepsilon)}{Q}\})$ versus $O(n\log\frac{k}{Q})$.

The rest of the paper is organized as follows. Section 1.2 gives a brief literature review. Section 2 provides a few preliminary remarks. Section 3 presents the top-level algorithm. Section 4 and Section 5 describe sub-algorithms to approximate the median and the k-th element with constant probability, respectively. Section 6 sketches the lower bound analysis. Appendices A–D include several technical details omitted from the main text. For other technical details not included in this manuscript, interested readers are referred to the current full version [17].

1.2 Brief Literature

Dating back to the 1987, Ravikumar et al. [31] already studied a variant of the problem of finding the *exact* minimum using unreliable comparisons when at most f comparison faults are allowed. They proved that $\Theta(fn)$ comparisons are necessary in the worst case. Later, Aigner [1] considered a *prefix-bounded* error model: for a fraction parameter $\gamma < \frac{1}{2}$, at most an γ-fraction of the past comparisons failed at any point during the execution of an algorithm. He proved that $\Theta(\frac{1}{1-p})^n$ comparisons are necessary to find the minimum in the worst case. Furthermore, he proved that if $p > \frac{1}{n-1}$, no algorithm can succeed with certainty [1].

When errors occur independently, as already discussed, Feige et al. [9] showed that the number of comparisons required for selecting the exact k-th smallest element with success probability at least $1 - Q$ is $\Theta(n \log \frac{k}{Q})$. Recently, Braverman et al. [4] investigated the *round complexity* and the number of comparisons by partition and selection algorithms. They proved that for any constant error probability, $\Theta(n \log n)$ comparisons are necessary for any algorithm that selects the minimum with high probability. Also, Chen et al. [6] studied the problem of computing the smallest k elements using r given independent noisy comparisons between each pair of elements. In a very general error model called *strong stochastic model*, they gave a linear-time algorithm with competitive ratio of $\tilde{O}(\sqrt{n})$, and also proved that this competitive ratio is tight.

The related problem of sorting with faults has also received considerable attention. When there are at most f comparison faults, $\Theta(n \log n + fn)$ comparisons are necessary and sufficient to correctly sort n elements [21, 25, 2]. For the prefix-bounded model, although Aigner's result on the minimum selection [1] implies that $(\frac{1}{1-p})^{O(n \log n)}$ are sufficient to sort n elements, Borgstrom and Kosaraju [3] showed that checking whether the input elements are sorted already requires $\Omega\big((\frac{1}{1-p})^n\big)$ comparisons. When comparison faults are permanent, or equivalently, when a pair of elements can only be compared once, the underlying sorting problem has also been extensively studied especially because it can be connected to both the *minimum feedback arc set* problem and the *rank aggregation* problem [26, 18, 4, 5, 20, 22, 14, 11, 13, 12]. There are also sorting algorithms for memory faults [10, 24]. For more knowledge about fault-tolerant search algorithms, we refer the interested readers to a survey by Pelc [30] and a monograph by Cicalese [8].

2 Preliminary

As explained in the beginning of Section 1 and in Remark 3, we assume that $n\epsilon < k \leq \frac{n}{2}$ and $\epsilon \geq \frac{1}{n}$ throughout the paper if no further specification. For ease of exposition, we use $\boldsymbol{\beta}$ to denote $\frac{\boldsymbol{k}}{\boldsymbol{n}}$ in some analyses, and we sometimes abuse the name x of an element to denote its rank, e.g., we might write "$x \in [l, r]$" to denote that the rank of x lies in the range $[l, r]$. Comparing two elements, x and y, yields an outcome of either $x < y$ or $y > x$. A typical subroutine in our algorithms is to draw elements using sampling with replacement, so multiple copies of an element may appear in a set. When two copies of the same element are compared, the tie is broken using any arbitrary (but consistent) ordering among the copies.

In our fault model, there is a standard strategy called *majority vote* for reducing the "error probability" of comparing two elements. We state this strategy as follows.

▶ **Lemma 4** (Majority Vote). *For any error probability $p \in [0, \frac{1}{2})$, there exists a postive integer c_p such that a strategy that compares two elements $2c_p \cdot t + 1$ times and returns the majority result succeeds with probability at least $1 - e^{-t}$, where $c_p = \lceil \frac{4(1-p)}{(1-2p)^2} \rceil$. The exact failure probability of this strategy is*

$$\sum_{i=0}^{c_p \cdot t} \binom{2c_p \cdot t + 1}{i} (1-p)^i p^{2c_p \cdot t + 1 - i}.$$

Many analyses in this paper will make use of the following Chernoff bound.

▶ **Lemma 5** (Chernoff Bound). *Let X be the sum of independent Bernoulli random variables. If $A \le E[X] \le B$, then for any $\delta \in (0,1)$,*

$$\Pr[X \ge (1+\delta)\cdot B] \le e^{-\frac{\delta^2}{3}B} \qquad \text{and} \qquad \Pr[X \le (1-\delta)\cdot A] \le e^{-\frac{\delta^2}{2}A}.$$

3 Top Level of Algorithm

The high-level idea is to reduce solving FT-APX(k,ε) on n elements with probability at least $1-Q$ to solving FT-APX$(\frac{m}{2},\frac{3}{8})$ on $m=\Theta(\log\frac{1}{Q})$ elements with probability at least $1-\frac{Q}{2}$. Specifically, if a *relevant* element can be selected with probability at least $\frac{8}{9}$, then m selected elements, for some $m=\Theta(\log\frac{1}{Q})$, contain at least $\frac{7}{8}m$ relevant elements with probability at least $1-\frac{Q}{2}$; see Lemma 23 in Appendix B. In this situation, at least $2\cdot(\frac{7}{8}-\frac{1}{2})\cdot m = 2\cdot\frac{3}{8}m$ elements around the median, i.e., the range $(\frac{1}{8}m,\frac{7}{8}m]$, are relevant. Therefore, solving the FT-APX$(\frac{m}{2},\frac{3}{8})$ problem on these m elements with probability at least $1-\frac{Q}{2}$ yields a relevant element with probability at least $1-2\cdot\frac{Q}{2}=1-Q$.

Section 5 will present an approach that uses $O(\frac{k}{n}\varepsilon^{-2})$ comparisons to select a relevant element with probability at least $\frac{8}{9}$, by which the above reduction takes $O(\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q})$ comparisons. In the remaining of this section, we will explain how to solve FT-APX$(\frac{m}{2},\frac{3}{8})$ with probability $1-\frac{Q}{2}$ efficiently in both expectation and determination.

We first design a simple trial-and-error method that uses *expected* $O(\log\frac{1}{Q})$ comparisons to select an element from $(\frac{1}{8}m,\frac{7}{8}m]$ with probability at least $1-\frac{Q}{2}$:

> Repeatedly pick an element randomly and verify if its rank lies in $(\frac{1}{8}m,\frac{7}{8}m]$ until one element passes the verification.

Since $(\frac{1}{8}m,\frac{7}{8}m]$ contains $\frac{3}{4}m$ elements, the expected number of repetitions before encountering a correct element is only $O(1)$. Therefore, the key is to implement the verification step such that the method returns a correct element with probability at least $1-\frac{Q}{2}$ and the expected number of comparisons is $O(\log\frac{1}{Q})$.

We implement the *verification step* for an element x based on a simple experiment that randomly picks three elements, and checks if x is neither the smallest nor the largest among the four elements. To simply the follow-up analysis, we assume that the three elements are sampled with replacement and picking x again is allowed. Under the above assumptions, the probability that the if-condition holds is $1-(\frac{r_x}{m})^3-(1-\frac{r_x}{m})^3=\frac{3}{4}-3(\frac{r_x}{m}-\frac{1}{2})^2$ where r_x is the rank of x among the m elements, $(\frac{r_x}{m})^3$ is the probability that none of the three picked element is larger than x and $(1-\frac{r_x}{m})^3$ is the probability that none of the three picked element is smaller than x. Also, the check can be conducted with success probability at least $\frac{17}{18}$ using $O(1)$ comparisons (by plugging in $n=4$, $k=1$ and $Q=\frac{1}{36}$ into Theorem 1 twice for the smallest and largest versions, respectively). Therefore, if $x\in(\frac{2}{8}m,\frac{6}{8}m]$, the experiment succeeds with probability *at least* $\left(\frac{3}{4}-3(\frac{2}{8}-\frac{1}{2})^2\right)\cdot\frac{17}{18}=\frac{17}{32}$, where $\left(\frac{3}{4}-3(\frac{2}{8}-\frac{1}{2})^2\right)$ is the minimum probability that the if-condition holds and $\frac{17}{18}$ is the success probability of the check, while if $x\in[1,\frac{1}{8}m]$ or $x\in(\frac{7}{8}m,m]$, the experiment succeeds with probability *at most* $\left(\frac{3}{4}-3(\frac{1}{8}-\frac{1}{2})^2\right)+\frac{1}{18}=\frac{221}{576}\le\frac{15}{32}$, where $\left(\frac{3}{4}-3(\frac{1}{8}-\frac{1}{2})^2\right)$ is the maximum probability that the if-condition holds and $\frac{1}{18}$ is the failure probability of the check.

In the above derivation, we ignore two ranges $(\frac{1}{8}m,\frac{2}{8}m]$ and $(\frac{6}{8}m,\frac{7}{8}m]$ since all elements in these two ranges are relevant, by which returning any element in these two ranges will not decrease the success probability, and since the considered range $(\frac{2}{8}m,\frac{6}{8}m]$ contains enough elements. Based on the above calculated probabilities, we can conceptually treat the above

simple experiment as an unreliable comparison with error probability $\frac{15}{32}$. By Lemma 4, if the verification step conducts this simple experiment $2 \cdot c_{15/32} \ln \frac{2}{Q} + 1$ times and takes the majority result, its success probability is at least $1 - \frac{Q}{2}$.

Now, we are ready to analyze the expected number of comparisons and the success probability of our trial-and-error method. Roughly speaking, the process of this trial-and-error method is similar to flipping a coin until a head appears where the probability of getting a head is at least $\frac{1}{4}$, which corresponds to a geometric distribution, and each flip requires $O(\log \frac{1}{Q})$ comparisons. More precisely, a single round returns an element in $(\frac{2}{8}m, \frac{6}{8}m]$ with probability *at least* $\frac{1}{2} \cdot (1 - \frac{Q}{2}) \geq \frac{1}{4}$, and thus the probability to conduct the i-th round is at most $(\frac{3}{4})^{i-1}$. Therefore, the expected number of comparisons is at most $\sum_{i \geq 1} (\frac{3}{4})^{i-1} \cdot (2 \cdot c_{15/32} \ln \frac{2}{Q} + 1) = O(\log \frac{1}{Q})$. Besides, this method fails only if it returns an element in $[1, \frac{1}{8}m]$ or $(\frac{7}{8}m, m]$, and such probability of a single round is *at most* $\frac{1}{4} \cdot \frac{Q}{2} = \frac{Q}{8}$. Again, since the probability to conduct the i-th round is at most $(\frac{3}{4})^{i-1}$, the failure probability is at most $\sum_{i \geq 1} (\frac{3}{4})^{i-1} \cdot \frac{Q}{8} = \frac{Q}{2}$, concluding the following theorem:

▶ **Theorem 6.** *It takes* ***expected*** $O(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$ *comparisons to solve the FT-APX*(k, ε) *problem with probability at least* $1 - Q$.

Finally, to derive a deterministic bound, we note that the simple experiment in the verification step may be viewed as a *biased* coin toss. From this viewpoint, we are able to turn the FT-APX$(\frac{m}{2}, \frac{3}{8})$ problem into finding a coin with bias bigger than $\frac{15}{32}$, given that at least half of the coins have bias at least $\frac{17}{32}$. Grossman and Moshkovitz [15] provided an algorithm that solves the new problem with probability $1 - \frac{Q}{2}$ using $O(\log \frac{1}{Q} \cdot (\log \log \frac{1}{Q})^2)$ coin tosses, leading to the following theorem.

▶ **Theorem 7.** *It takes* $O(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q} + \log \frac{1}{Q} (\log \log \frac{1}{Q})^2)$ *comparisons to solve the FT-APX*(k, ε) *problem with probability at least* $1 - Q$.

4 Approximate Median Selection

We attempt to select an element in $(\frac{n}{2} - n\varepsilon, \frac{n}{2} + n\varepsilon]$, i.e., $\boldsymbol{k = \frac{n}{2}}$, with probability at least $1 - \frac{1}{18}$ using only $O(\varepsilon^{-2})$ comparisons. This algorithm will then be applied in Section 5 as a subroutine. A straightforward method, denoted by ST-Median(ε), picks $m = \Theta(\varepsilon^{-2})$ elements randomly to make their median *relevant* with probability at least $1 - \frac{1}{72}$ and applies the Select$(\frac{m}{2}, \frac{1}{72})$ algorithm (Theorem 1), resulting in a failure probability of at most $\frac{1}{36}$. However, the Select$(\frac{m}{2}, \frac{1}{72})$ algorithm takes $O(m \log m) = O(\varepsilon^{-2} \log \varepsilon^{-1})$ comparisons with an *extra* logarithmic factor. To achieve $O(\varepsilon^{-2})$ comparisons, we will "purify" the input elements in a way that the ratio of relevant elements is increasing while the underlying median is still relevant. Once the ratio of relevant elements becomes a constant fraction, i.e., from 2ε to $\Omega(1)$, we can afford to apply the ST-Median algorithm. We assume that $\varepsilon < \frac{1}{6}$ since if $\varepsilon \geq \frac{1}{6}$, the ST-Median$(\varepsilon)$ algorithm takes only $O(\varepsilon^{-2} \log \varepsilon^{-1}) = O(1)$ comparisons.

▶ Remark 8. A major difficulty to overcome in the purifying process is the following: if we consider three elements that are each relevant with probability ρ, then their median, even in the absence of comparison faults, is relevant with probability at most $\frac{3}{2}\rho + O(\rho^2)$, which is a lot less than 3ρ. Thus, we risk running out of elements long before the ratio of relevant elements becomes a constant. This issue remains if we replace three by a larger constant, and it applies to any algorithm that works in a non-constant number of phases, including algorithms that more closely resemble Quickselect. Those algorithms would need to start with $\Omega(\varepsilon^{-(2+\delta)})$ elements for some $\delta > 0$ and hence cannot achieve the $O(\varepsilon^{-2})$ bound.

To settle the issue in Remark 8, we maintain a multiset of elements and re-sample from this multiset at every phase. Our re-sampling method allows us to decrease the number of elements by less than a factor of $\frac{3}{2}$, so we can avoid running out of elements.

The algorithm is sketched as follows:

1. For $1 \le i \le L$, generate a multiset M_i of n_i elements by repeatedly picking three elements from M_{i-1} *randomly* and selecting the median of the three using a symmetric median selection algorithm (Lemma 9 below).
2. Apply the ST-Median($\boldsymbol{\varepsilon_L}$) algorithm on M_L.

Initially, $M_0 = S$, $n_0 = n$, $\varepsilon_0 = \varepsilon$. M_i is called ***good*** if all elements in the range $(\frac{n_i}{2} - n_i\varepsilon_i, \frac{n_i}{2} + n_i\varepsilon_i]$ are *relevant*. Moreover, n_i is decreasing with i while ε_i is increasing with i, and $L = \min\{i \mid \varepsilon_i \ge \frac{1}{6}\}$, i.e., the minimum of number of rounds such that at least $2 \cdot \frac{1}{6} = \frac{1}{3}$ of the elements around the middle is relevant. The rest of this section illustrates the idea behind this process and implements these parameters n_i and ε_i.

▶ **Lemma 9.** *For **three** elements, consider the following median selection algorithm:*

1. *For each pair of elements, apply the majority vote strategy with $2c_p \cdot 4 + 1$ comparisons (Lemma 4), and assign a point to the element that attains the majority result.*
2. *Return the element with exactly one point. If all three elements get exactly one point, return one of them uniformly at random.*

The above algorithm returns the median with probability at least $1 - \frac{1}{13}$, and returns the minimum and the maximum with the same probability, i.e., at most $\frac{1}{26}$.

The purifying process is inspired by a simple observation: a randomly picked element is relevant with probability 2ε, while the *median* of three randomly picked elements is relevant with probability much greater than 2ε. Let E_S denote the event that the median of three randomly picked elements is small. Then,

$$\Pr[E_S] = 3\left(\frac{1}{2} - \varepsilon\right)^2\left(\frac{1}{2} + \varepsilon\right) + \left(\frac{1}{2} - \varepsilon\right)^3 = \frac{1}{2} - \frac{3}{2}\varepsilon + 2\varepsilon^3.$$

If $\varepsilon < \frac{1}{6}$, then $\Pr[E_S] \le \frac{1}{2} - \frac{3}{2}\varepsilon + 2(\frac{1}{6})^2\varepsilon = 1 - \frac{13}{9}\varepsilon$. By Lemma 9, the median selection returns the median with probability at least $1 - \frac{1}{13}$, and returns the minimum (resp. the maximum) with probability at most $\frac{1}{26}$. A simple calculation, together with the above arguments, gives the following lemma:

▶ **Lemma 10.** *If M_{i-1} is good, then each element in M_i is small (resp. large) with probability at most $\frac{1}{2} - \frac{4}{3}\varepsilon_{i-1}$.*

Proof. We only prove the small case, and it is symmetric to the large case. Let p_s denote the probability that an element randomly picked from M_{i-1} is small. Since M_{i-1} is good, all elements in its range $(\frac{1}{2}n_{i-1} - n_{i-1}\varepsilon_{i-1}, \frac{1}{2}n_{i-1} + n_{i-1}\varepsilon_{i-1}]$ are relevant, and $p_s \le \frac{1}{2} - \varepsilon_{i-1}$. Let p_1, p_2 and p_3 denote the probabilities that the median selection algorithm in Lemma 9 returns the minimum, the median and the maximum of three elements, respectively. By Lemma 9, $p_2 \ge \frac{12}{13}$, and $p_1 = p_3 \le \frac{1}{26}$. Also recall that $\varepsilon_{i-1} \le \frac{1}{6}$. Then, the probability that an element in M_i is small is

$$
\begin{aligned}
&\underbrace{p_s^3}_{\text{three small}} + \underbrace{3p_s^2(1-p_s)}_{\text{two small \& one non-small}} \cdot(1-p_3) + \underbrace{3p_s(1-p_s)^2}_{\text{one small \& two non-small}} \cdot p_1 \\
&= p_s^3 + 3p_s^2(1-p_s)(1-p_1) + 3p_s(1-p_s)^2 \cdot p_1 \\
&= \left(3p_s^2 - 2p_s^3\right) + p_1 \cdot \underbrace{\left(3p_s - 9p_s^2 + 6p_s^3\right)}_{\geq 0 \text{ since } 0 \leq p_s \leq \frac{1}{2}} \\
&\overset{p_1 \leq \frac{1}{26}}{\leq} \frac{1}{26} \cdot \underbrace{\left(3p_s + 69p_s^2 - 46p_s^3\right)}_{f(x) := -46x^3+69x^2+3x \ \&\ f'(x)>0 \text{ for } 0 \leq x \leq 1} \\
&\overset{p_s \leq \frac{1}{2} - \varepsilon_{i-1}}{\leq} \frac{1}{26} \cdot \left(3\left(\frac{1}{2} - \varepsilon_{i-1}\right) + 69\left(\frac{1}{2} - \varepsilon_{i-1}\right)^2 - 46\left(\frac{1}{2} - \varepsilon_{i-1}\right)^3\right) \\
&= \frac{1}{2} - \frac{75}{52}\varepsilon_{i-1} + \frac{23}{13}\varepsilon_{i-1}^3 \\
&\overset{\varepsilon_{i-1} < \frac{1}{6}}{\leq} \frac{1}{2} - \frac{75}{52}\varepsilon_{i-1} + \frac{23}{468}\varepsilon_{i-1} \\
&= \frac{1}{2} - \frac{652}{468}\varepsilon_{i-1} \\
&\leq \frac{1}{2} - \frac{4}{3}\varepsilon_{i-1}
\end{aligned}
$$

◀

By Lemma 10, it is feasible to set $\varepsilon_i = (\frac{5}{4})^i \cdot \varepsilon$, i.e., growing slightly slower than $\frac{4}{3}$. Then, the size n_i is set as $\lceil 2000 \cdot i \cdot (\frac{4}{5})^{\mathbf{2i}} \cdot \varepsilon^{-2} \rceil$ to limit the number of comparisons and the failure probability. First, n_i is linear in ε^{-2} since the minimum number of elements to be looked at is $\Omega(\varepsilon^{-2})$ (Section 6). Second, to bound the total number of comparisons, n_i should shrink exponentially with i. Third, to bound the failure probability of the algorithm, the failure probability of the i-th round should also shrink exponentially with i. From the above three aspects, since the Chernoff bound (Lemma 5) will be applied for the probabilistic analysis, n_i should be linear in i, and the shrink factor of n_i should be at least $(\frac{4}{5})^2$ to cancel out the square of the growth factor $\frac{5}{4}$ of ε_i.

Because the ST-Median(ε_L) algorithm (stated at the beginning of this section) fails with probability at most $\frac{1}{36}$, it is sufficient to prove that $\Pr[M_L \text{ is good }] \geq 1 - \frac{1}{36}$. Let E_i denote the event that M_i is good. By definition, $\Pr[E_0] = 1$. With the Chernoff bound, we can prove the following lemma:

▶ **Lemma 11.** *For $1 \leq i \leq L$,*

$$\Pr[M_i \textit{ is NOT good } \mid M_{i-1} \textit{ is good }] \leq 2 \cdot e^{-5i}.$$

Proof. Assume that M_{i-1} is good. Let X_i be the number of small elements in M_i and let Y_i be the number of large elements in M_i. For the statement, it is sufficient to prove that $\Pr[X_i \geq \frac{n_i}{2} - n_i\varepsilon_i] \leq e^{-5i}$ and $\Pr[Y_i \geq \frac{n_i}{2} - n_i\varepsilon_i] \leq e^{-5i}$. We will prove the first claim, and it is symmetric to the second claim. By Lemma 10,

$$E[X_i] \leq (\frac{1}{2} - \frac{4}{3}\varepsilon_{i-1})n_i = (\frac{1}{2} - \frac{4}{3} \cdot \frac{4}{5}\varepsilon_i)n_i = (\frac{1}{2} - \frac{16}{15}\varepsilon_i)n_i = \frac{15 - 32\varepsilon_i}{30}n_i.$$

By Lemma 5 (Chernoff bound), we can get

$$\begin{aligned}
\Pr\left[X_i \geq \left(\frac{1}{2}-\varepsilon_i\right)n_i\right] &= \Pr\left[\left(1+\frac{\frac{1}{15}\varepsilon_i}{\frac{1}{2}-\frac{16}{15}\varepsilon_i}\right)\left(\frac{1}{2}-\frac{16}{15}\varepsilon_i\right)n_i\right] \\
&= \Pr\left[\left(1+\underbrace{\frac{2\varepsilon_i}{15-32\varepsilon_i}}_{:=\delta}\right)\underbrace{\left(\frac{15-32\varepsilon_i}{30}n_i\right)}_{\geq E[X_i]}\right] \\
&\underbrace{\leq \exp\left(-\frac{1}{3}\left(\frac{2\varepsilon_i}{15-32\varepsilon_i}\right)^2\cdot\left(\frac{15-32\varepsilon_i}{30}n_i\right)\right)}_{\text{Lemma 5}} \\
&= \exp\left(-\frac{4}{90}\cdot\frac{\varepsilon_i^2}{15-32\varepsilon_i}\cdot n_i\right) \leq \exp\left(-\frac{4}{90}\cdot\frac{\varepsilon_i^2}{15}\cdot n_i\right) \\
&= \exp\left(-\frac{2}{675}\cdot\left(\left(\frac{5}{4}\right)^{2i}\varepsilon^2\right)\cdot\left(2000\cdot i\cdot\left(\frac{4}{5}\right)^{2i}\cdot\varepsilon^{-2}\right)\right) \\
&\leq e^{-5i}.
\end{aligned}$$

◀

By Lemma 11, we can lower bound $\Pr[E_L]$ as

$$\Pr[E_L] = 1-\Pr[\bigcup_{i=1}^{L}\overline{E_i}\mid E_{i-1}] \geq 1-\sum_{i=1}^{L}2\cdot e^{-5i} \geq 1-4\cdot e^{-5} \geq 1-\frac{1}{36}.$$

By Lemma 9, each median selection takes $O(1)$ comparisons, so the purifying process takes $O(\sum_{i=1}^{L}n_i) = O\big(\varepsilon^{-2}\sum_{i=1}^{L}i\cdot(\frac{4}{5})^{2i}\big) = O(\varepsilon^{-2})$ comparisons. Since $\varepsilon_L \geq \frac{1}{6}$, the ST-Median($\varepsilon_L$) algorithm takes $O(1)$ comparisons, concluding the following theorem:

▶ **Theorem 12.** *It takes $O(\varepsilon^{-2})$ comparisons to select an element in $(\frac{n}{2}-n\varepsilon, \frac{n}{2}+n\varepsilon]$ with probability at least $1-\frac{1}{18}$.*

5 Approximate k-th Element Selection

We attempt to select an element in $(k-n\varepsilon, k+n\varepsilon]$ with probability at least $1-\frac{1}{9}$ using only $O(\frac{k}{n}\varepsilon^{-2})$ comparisons. Recall that $k > n\varepsilon$ as assumed in Remark 3. If $n\varepsilon < k \leq 2n\varepsilon$, we halve the value of ε so that $k > 2n\varepsilon$, which does not increase the asymptotic complexity. Therefore, we can safely assume $k > 2n\varepsilon$ afterwards. In this scenario, the straightforward approach mentioned in Section 1 requires $O\big(\frac{k}{n}\varepsilon^{-2}\log(k\varepsilon^{-1})\big)$ comparisons with an extra $\log(k\varepsilon^{-1})$ factor. Another approach is to add $n-2k$ dummy smallest elements (so that the relevant elements lie in the middle) and to apply the algorithm in Section 4 with $\frac{\varepsilon}{2}$, leading to $O(\varepsilon^{-2})$ comparisons. As a result, both approaches are more expensive than $O(\frac{k}{n}\varepsilon^{-2})$.

At a high level, our breakthrough is an iterative "purifying" process that increases both the ratio of relevant elements and the relative position of k, i.e., the middle position of relevant elements, while "controlling" the relative position. Once the relative position becomes a constant fraction of the remaining elements, e.g., $\frac{1}{8}$, we add dummy smallest elements and apply the approximate median selection algorithm in Section 4. As the ratio of relevant elements increases at the same time, the resulting number of comparisons will be $O(\frac{k}{n}\varepsilon^{-2})$ instead of $O(\varepsilon^{-2})$.

The algorithm is sketched as follows:

1. For $1 \leq i \leq L$, generate a set S_i of n_i elements by repeatedly picking two elements from S_{i-1} *randomly* and selecting the minimum of the two using $2\mathrm{c}_p\cdot 3+1$ comparisons (Lemma 4).

2. Add $n_L - 2k_L$ dummy smallest elements to M_L and apply the approximate median selection algorithm in Section 4 on M_L with respect to ε_L.

Initially, $S_0 = S$, $n_0 = n$, $k_0 = k$, $\varepsilon_0 = \varepsilon$. S_i is called ***good*** if all elements in the range $(k_i - n_i\varepsilon_i,\, k_i + n_i\varepsilon_i]$ are *relevant*. For ease of exposition, let β_i denote $\frac{k_i}{n_i}$. Both β_i and ε_i increase with i while n_i decreases with i. We set $L = \min\{i \mid \beta_i \geq \frac{1}{8}\}$. Recall that $\beta = \frac{k}{n}$. We assume that $\beta < \frac{1}{8}$; otherwise, we conduct the second step directly, i.e., $L = 0$.

The purifying process is based on a simple observation that the minimum of two randomly picked element is *small* with probability

$$\underbrace{(\beta-\varepsilon)^2}_{\text{two small}} + \underbrace{2\,(\beta-\varepsilon)\,(1-(\beta-\varepsilon))}_{\text{one small \& one non-small}} = 2\,(\beta-\varepsilon) - (\beta-\varepsilon)^2\,,$$

while a randomly picked element is small with probability merely $\beta - \varepsilon$. By a similar calculation, the minimum of two randomly picked elements is *relevant* with $4\varepsilon - \beta \cdot 4\varepsilon$. Since k is exactly the number of small elements plus half the number of relevant elements, the above derivation suggests the following formulation of β_i:

$$\beta_i := \underbrace{2\,(\beta_{i-1}-\varepsilon_{i-1}) - (\beta_{i-1}-\varepsilon_{i-1})^2}_{\Pr[\text{ small }]} + \underbrace{(2\varepsilon_{i-1} - \beta_{i-1}\cdot 2\varepsilon)}_{\Pr[\text{ relevant }]\div 2}\,.$$

These derivations need to adapt to the failure probability q of selecting the minimum using $2c_p \cdot 3 + 1$ comparisons. By Lemma 4, $q \leq e^{-3} < \frac{1}{20}$ and $q = \sum_{i=1}^{3c_p}(1-p)^i p^{6c_p+1-i}$. Then, a selected element in the first round is *relevant* with probability

$$\underbrace{4\varepsilon^2}_{\text{two relevant}} + q \cdot \underbrace{2\cdot(\beta-\varepsilon)\,2\varepsilon}_{\text{one small \& one relevant}} + (1-q)\cdot \underbrace{2\cdot(1-(\beta+\varepsilon))\,2\varepsilon}_{\text{one large \& one relevant}}\,,$$

which is equal to $4\varepsilon \cdot \big((1-q) - (1-2q)\cdot\beta\big)$. Since $\beta < \frac{1}{8}$ and $q < \frac{1}{20}$, the above probability is larger than $\frac{67}{40}\cdot 2\varepsilon$. Therefore, it is feasible to set $\boldsymbol{\varepsilon_i = (\frac{3}{2})^i \cdot \varepsilon}$, i.e., growing slower than $\frac{67}{20}$.

To fit the formulation of β_i to the above failure probability q, a similar calculation yields that each selected element in the first round is *small* with probability

$$(\beta-\varepsilon)^2 + (1-q)\cdot 2\,(\beta-\varepsilon)\,(1-(\beta-\varepsilon))\,.$$

Since the relative position is the number of small elements plus half the number of relevant elements, it is feasible to set the value of β_i as follows (after arrangement):

$$\beta_i := \left(2\beta_{i-1} - \beta_{i-1}^2 - \varepsilon_{i-1}^2\right) - 2q\left(\beta_{i-1} - \beta_{i-1}^2 - \varepsilon_{i-1}^2\right).$$

Moreover, we can prove by induction important properties of β_i as stated below:

▶ **Lemma 13.** *For* $0 \leq i \leq L$,

$$\beta_i > 2\varepsilon_i \quad \textit{and} \quad \beta_i \leq 2^i\cdot\beta. \quad \textit{Thus,} \quad \frac{k_i}{n_i} \leq 2^i \cdot \frac{k}{n} \quad \textit{for} \quad 0 \leq i \leq L.$$

Proof. We prove by induction. For $i = 0$, by assumption in the first paragraph of Section 5, $\beta > 2\varepsilon$, i.e., $\beta_0 = \beta > 2\varepsilon = 2\varepsilon_0$. Also, $\beta_0 = \beta \leq 2^0 \cdot \beta$. Assume that for $i = k \geq 0$, $\beta_k > 2\varepsilon_k$ and $\beta_k \leq 2^k \cdot \beta$. Note that $k < L$; otherwise, the $(k+1)$-th round does not exist. By Section 5,

$$\beta_{k+1} = \left(2\beta_k - \beta_k^2 - \varepsilon_k^2\right) - 2q\left(\beta_k - \beta_k^2 - \varepsilon_k^2\right).$$

We first prove that $\beta_{k+1} > 2\varepsilon_{k+1}$ as follows:

$$\begin{aligned}
\beta_{k+1} &= \left(2\beta_k - \beta_k^2 - \varepsilon_k^2\right) - 2q\left(\beta_k - \beta_k^2 - \varepsilon_k^2\right) \overset{q<\frac{1}{20}}{>} \frac{19}{10}\beta_k - \frac{9}{10}(\beta_k^2 + \varepsilon_k^2) \\
&= \frac{9}{10}\left(-\left(\beta_k - \frac{19}{18}\right)^2 + \left(\frac{19}{18}\right)^2 - \varepsilon_k^2\right) \\
&\overset{\beta_k<\frac{1}{8}\ \&\ 2\varepsilon_k<\beta_k}{>} \frac{9}{10}\left(-\left(2\varepsilon_k - \frac{19}{18}\right)^2 + \left(\frac{19}{18}\right)^2 - \varepsilon_k^2\right) \\
&= \frac{19}{5}\varepsilon_k - \frac{9}{2}\varepsilon_k^2 \overset{\varepsilon_k<\frac{1}{2}\beta_k<\frac{1}{16}}{>} \frac{563}{160}\varepsilon_k > 3\varepsilon_k = 2\varepsilon_{k+1}.
\end{aligned}$$

Then, we prove that $\beta_{k+1} \le 2^{k+1} \cdot \beta$ as follows:

$$\begin{aligned}
\beta_{k+1} &= \left(2\beta_k - \beta_k^2 - \varepsilon_k^2\right) - 2q\left(\beta_k - \beta_k^2 - \varepsilon_k^2\right) \overset{q\ge 0}{\le} 2\beta_k - \beta_k^2 - \varepsilon_k^2 \\
&\le 2\beta_k \le 2 \cdot 2^k \cdot \beta = 2^{k+1} \cdot \beta.
\end{aligned}$$

◀

The size n_i of S_i is set as $\lceil 960 \cdot i \cdot (\frac{8}{9})^i \cdot \frac{k}{n}\varepsilon^{-2} \rceil$ to control the number of comparisons and the failure probability. Similar to Section 4, n_i should shrink exponentially with i and should also be linear in both $\frac{k}{n}\varepsilon^{-2}$ and i. The major difference lies in that the existence of k_i changes the shrink factor of n_i. Since $\frac{k_i}{n_i} \le 2^i \cdot \frac{k}{n}$ (by Lemma 13) and $\varepsilon_i = (\frac{3}{2})^i \cdot \varepsilon$, the shrink factor of n_i should be at least $\frac{8}{9}$. This is based on the fact that $2^{-i} \cdot (\frac{3}{2})^{2i} \cdot (\frac{8}{9})^i = 1$, which will be much clearer in the probability analysis.

To sum up, $\boldsymbol{n_i} = \lceil 960 \cdot i \cdot (\frac{8}{9})^i \cdot \frac{k}{n}\varepsilon^{-2} \rceil$, $\boldsymbol{\varepsilon_i} = (\frac{3}{2})^i \cdot \varepsilon$, $\boldsymbol{\beta_i} = (2\beta_{i-1} - \beta_{i-1}^2 - \varepsilon_{i-1}^2) - 2q \cdot (\beta_{i-1} - \beta_{i-1}^2 - \varepsilon_{i-1}^2)$ with $\boldsymbol{q} = \sum_{i=1}^{3\mathrm{c}_p} (1-p)^i p^{6\cdot \mathrm{c}_p + 1 - i}$, and $\boldsymbol{L} = \min\{i \mid \beta_i \ge \frac{1}{8}\}$.

To attain the success probability $1 - \frac{1}{9}$, since the approximate median selection in Section 4 fails with probability at most $\frac{1}{18}$, it is sufficient to prove that $\Pr[S_L \text{ is good}] \ge 1 - \frac{1}{18}$ (Theorem 12). Let E_i denote the event that S_i is good. By definition, $\Pr[E_0] = 1$. Applying the Chernoff bound with the above parameters gives the following lemma: (The proof is rather technical, and interested readers are referred to the current full version [17].)

▶ **Lemma 14.** *For* $1 \le i \le L$

$$\Pr[S_i \textit{ is NOT good} \mid S_{i-1} \textit{ is good}] \le 2 \cdot e^{-4.3 \cdot i}.$$

By Lemma 14, we can lower bound $\Pr[E_L]$ as

$$\Pr[E_L] = 1 - \Pr[\bigcup_{i=1}^{L} \overline{E_i} \mid E_{i-1}] \ge 1 - \sum_{i=1}^{L} 2 \cdot e^{-4.3 \cdot i} \ge 1 - 4 \cdot e^{-4.3} \ge 1 - \frac{1}{18}.$$

For the number of comparisons, since each selection takes $2\mathrm{c}_P \cdot 3 + 1 = O(1)$ comparisons, the purifying process takes $\sum_{i=1}^{L} O(n_i) = \frac{k}{n}\varepsilon^{-2} \cdot \sum_{i=1}^{L} O\big(i \cdot (\frac{8}{9})^i\big) = O(\frac{k}{n}\varepsilon^{-2})$ comparisons. By Theorem 12, the approximate median selection takes $O(\varepsilon_L^{-2}) = O\big((\frac{2}{3})^{2L} \cdot \varepsilon^{-2}\big) = O(2^{-L} \cdot \varepsilon^{-2})$ comparisons. Since $\frac{k_L}{n_L} \le 2^L \cdot \frac{k}{n}$ (Lemma 13) and $\frac{k_L}{n_L} = \beta_L \ge \frac{1}{8}$, we have $2^{-L} = O(\frac{k}{n})$ and $O(2^{-L} \cdot \varepsilon^{-2}) = O(\frac{k}{n}\varepsilon^{-2})$, implying the following main theorem:

▶ **Theorem 15.** *It takes* $O(\frac{k}{n}\varepsilon^{-2})$ *comparisons to select an element in* $(k - n\varepsilon, k + n\varepsilon]$ *with probability at least* $1 - \frac{1}{9}$.

6 Lower Bound

We sketch the derivation of an $\Omega(\min\{n, \frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q}\})$ lower bound for the *expected* number of comparisons. Our derivation contains two key ingredients. First, we design an auxiliary decision tree that simulates any corresponding randomized algorithm with success probability at least $1-Q$, but owns nice properties for the analysis. Second, we derive a sampling lemma (Corollary 18) that lower bounds the probability of a returned element being relevant.

We assume that $\boldsymbol{4n\varepsilon \le k}$. If $k \le n\varepsilon$, the $\Omega(\varepsilon^{-1}\log\frac{1}{Q})$ lower bound for the approximate minimum selection problem [23] applies, and if $n\varepsilon < k < 4n\varepsilon$, we multiply the value of ε by 4 so that $k \le n\varepsilon$ and the former argument still works, which does not change the lower bound asymptotically. We assume that there are **no comparison faults**, which does not increase the lower bound and is easier for analysis.

Let T be the decision tree of any *randomized* algorithm that solves FT-APX(k,ε) with probability at least $1-Q$. T is said to *look at* an element x if T performs at least one comparison involving x. Let $\mathfrak{D}$ be the *expected* number of elements that T looks at. Since $\mathfrak{D}$ is not larger than twice the expected number of comparisons, it is sufficient to lower bound $\mathfrak{D}$. If $\mathfrak{D} \ge \frac{n}{10}$, then $\mathfrak{D} = \Omega(n)$. Below, we deal with the case that $\mathfrak{D} < \frac{n}{10}$.

We construct an auxiliary decision tree $\tilde{T}$ based on T: $\tilde{T}$ first simulates T until reaching a leaf u of T that returns an element x, and then conducts three additional steps *sequentially*:

(a) If T does not look at x, then $\tilde{T}$ compares x with another element.

(b) If $\tilde{T}$ has looked at fewer than $2\mathfrak{D} + \lceil\frac{8n}{k}\rceil$ elements so far, then $\tilde{T}$ performs more comparisons such that $\tilde{T}$ has looked at *exactly* $2\mathfrak{D} + \lceil\frac{8n}{k}\rceil$ elements after this step.

(c) $\tilde{T}$ compares all pairs of elements that it has looked at, and then returns x.

Intuitively, $\tilde{T}$ represents the same algorithm as T, but these additional steps will give $\tilde{T}$ nice properties for analysis. Roughly speaking, Step (a) ensures that $\tilde{T}$ must look at the returned element. The term $2\mathfrak{D} + \lceil\frac{8n}{k}\rceil$ in Step (b) comes from that by Markov's inequality, T looks at more than $2\mathfrak{D}$ elements with probability at most $\frac{1}{2}$, and that the $\lceil\frac{8n}{k}\rceil$ term will cancel out an $\frac{k}{n}$ term later. Step (c) enables $\tilde{T}$ to know the sorted order of the elements that $\tilde{T}$ looked at. These three steps lead to three nice properties in the following lemma.

▶ **Lemma 16.** *$\tilde{T}$ has the following properties:*

(1) *$\tilde{T}$ knows the sorted order of the elements that $\tilde{T}$ has looked at.*

(2) *$\tilde{T}$ has success probability at least $1-Q$.*

(3) *$\tilde{T}$ looks at exactly $2\mathfrak{D} + \lceil\frac{8n}{k}\rceil$ elements with probability at least $1/2$. Note that this includes the elements that $\tilde{T}$ looks at during its simulation of T.*

Proof. Property (1) comes from step (c) in which $\tilde{T}$ compares all pairs of elements that $\tilde{T}$ has looked at. Remember that we assume no comparison faults for the lower bound analysis.

For property (2), note that $\tilde{T}$ first simulates T, then does some additional comparison and then returns the element that T would have returned (independent of the outcome of the additional comparisons). Hence $\tilde{T}$ has the same success probability as T, which is at least $1-Q$ by assumption.

For property (3), according to the three steps, if T looks at no more than $2\mathfrak{D}$ elements, then $\tilde{T}$ will look exactly $2\mathfrak{D} + \lceil\frac{8n}{k}\rceil$ elements. Since the probability that T looks at more than $2\mathfrak{D}$ elements is at most $\frac{1}{2}$ (by the definition of $\mathfrak{D}$ and by Markov's inequality), property (3) follows. ◀

Let us consider the execution of $\tilde{T}$ on a uniformly shuffled input. Recall that we assume there are no comparison faults. According to Lemma 16(1), the element returned by a fixed leaf of $\tilde{T}$ will always have the same rank among the elements that $\tilde{T}$ has looked at,

independent of the order of the input. Under this circumstance, it is desirable to analyze the relevance of elements with a certain rank among all sampled elements. Toward this end, we derive a sampling lemma (Lemma 17) that lower bounds the probability of an element with a small sample rank being small and the probability of an element with a large sample rank being large, which further induces a key sampling lemma (Corollary 18) that lower bounds the probability of an element being relevant.

Corollary 18 roughly states that for a set A of randomly sampled elements (without replacement), the probability that an element of a certain rank in A is NOT relevant decreases as $e^{-\Omega(\frac{n}{k}\cdot\varepsilon^2\cdot|A|)}$. Remark 22 in the end of Section 6 will sketch the ideas of deriving Lemma 17. For ease of exposition, we also use β to denote $\frac{k}{n}$ in Lemma 17 and Corollary 18.

▶ **Lemma 17.** *Let A consist of $m \le \frac{n}{4}$ elements sampled from S without replacement. Suppose that $m\beta \ge 8$ and that $\frac{1}{2} \ge \beta \ge 4\varepsilon$. Then, there is an absolute constant $\eta = \sqrt{\frac{\pi}{320}} \cdot e^{-24}$, coming from Theorem 26, with the following properties:*

1. *Let u be the r-th smallest element of A. If $r \le \lceil \beta m \rceil$, then u is small with probability at least*

$$\eta \cdot e^{-12\frac{\varepsilon^2}{\beta(1-\beta)}m}.$$

2. *Let v be the r-th largest element of A. If $r \le \lceil (1-\beta)m \rceil$, then v is large with probability at least*

$$\eta \cdot e^{-12\frac{\varepsilon^2}{\beta(1-\beta)}m}.$$

Since $1-\beta \ge 1/2$ and every element of the m elements is either among the $\lceil \beta m \rceil$ smallest ones or among the $\lceil (1-\beta)m \rceil$ largest ones, Lemma 17 directly implies Corollary 18.

▶ **Corollary 18.** *Let A consist of $m \le \frac{n}{4}$ elements sampled from S* without *replacement. Suppose that $m\beta \ge 8$ and that $\frac{1}{2} \ge \beta \ge 4\varepsilon$. Then, an arbitrary element u in A is NOT relevant with probability at least*

$$\eta \cdot e^{-24\cdot\frac{n}{k}\cdot\varepsilon^2\cdot m}$$

for an absolute constant $\eta = \sqrt{\frac{\pi}{320}} \cdot e^{-24}$ coming from Theorem 26.

By Lemma 16(3), with probability at least 1/2, the execution of $\tilde{T}$ reaches a leaf after looking at exactly $2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil$ elements. Together with Corollary 18 on each such leaf, we can lower bound the failure probability of $\tilde{T}$ as shown in the following lemma.

▶ **Lemma 19.** *If $k \ge 200$ and $4n\varepsilon \le k$, then the failure probability of $\tilde{T}$ on a uniformly shuffled input is at least*

$$\frac{1}{2} \cdot \eta \cdot e^{-24\varepsilon^2\frac{n}{k}(2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil)} \quad \text{for an absolute constant } \eta = \sqrt{\frac{\pi}{320}} \cdot e^{-24} \text{ from Theorem 26.}$$

Proof. Recall that we build $\tilde{T}$ only when $\mathfrak{D} < \frac{n}{10}$. Fix a leaf w of $\tilde{T}$. Suppose that the execution of $\tilde{T}$ reaches w. Let x be the element that $\tilde{T}$ returns and let A be the set of elements that $\tilde{T}$ has looked at when the execution reaches w.

As $\tilde{T}$ is run on a uniformly shuffled input, the distribution of the set A as a random variable is the same as the distribution of a set of $|A|$ elements sampled from S without replacement. Note that since $\tilde{T}$ has only compared elements in A, these comparisons do not affect the distribution of A as a random variable. By Lemma 16.(1), x always has the same rank in A. If $|A| = 2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil$, then $|A| \le \frac{n}{4}$ and $\frac{k}{n} \cdot |A| \ge 8$. Moreover, we have $4n\varepsilon \le k$ by assumption. Therefore, if $|A| = 2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil$, Corollary 18 implies that $\tilde{T}$ fails with probability at least

$$\eta \cdot e^{-24 \cdot \frac{n}{k} \cdot \varepsilon^2 \cdot |A|} = \eta \cdot e^{-24 \cdot \frac{n}{k} \cdot \varepsilon^2 \cdot (2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil)}.$$

In summary, if $\tilde{T}$ reaches a leaf after looking at exactly $2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil$ elements, then $\tilde{T}$ fails with probability at least $\eta \cdot e^{-24 \cdot \frac{n}{k} \cdot \varepsilon^2 \cdot (2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil)}$. By Lemma 16.(3), the if-condition holds with probability at least $\frac{1}{2}$, leading to the statement. ◀

Since $\tilde{T}$ succeeds with probability at least $1 - Q$, we have $Q \geq \frac{1}{2}\,\eta \cdot e^{-24\varepsilon^2 \frac{n}{k}(2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil)}$, implying that $\mathfrak{D} = \Omega(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$. We can conclude the following main theorem.

▶ **Theorem 20.** *If $Q < \frac{1}{2}$, then the expected number of comparisons performed by any randomized algorithm that solves the FT-APX(k, ε) problem with probability at least $1 - Q$ is $\Omega\big(\min\big\{n, \frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q}\big\}\big)$.*

Proof. As discussed in the beginning of Section 6, if $k < 4n\varepsilon$, the lower bound $\Omega(\varepsilon^{-1} \log \frac{1}{Q})$ for approximate minimum selection [23] applies. Similarly, if $k \leq 200$, we can increase ε by $\frac{200}{n}$ so that $n\varepsilon > n \cdot \frac{200}{n} = 200 \geq k$, which changes ε by at most a constant factor[1], and apply the lower bound for the approximate minimum selection [23]. Therefore, it is sufficient to consider the case that $4\varepsilon \leq \frac{k}{n} \leq \frac{1}{2}$ and $k \geq 200$. Recall that T is the decision tree of any randomized algorithm that solves FT-APX(k, ε) with probability at least $1 - Q$ and $\mathfrak{D}$ is the expected number of elements that T looks at. If $\mathfrak{D} \geq \frac{n}{10}$, a lower bound $\Omega(n)$ follows. Otherwise, we build the auxiliary decision tree $\tilde{T}$.

By Lemma 16.(2), the success probability of $\tilde{T}$ is at least $1 - Q$, and by Lemma 19, the failure probability of $\tilde{T}$ is at least $\frac{1}{2} \cdot \eta \cdot e^{-24\varepsilon^2 \frac{n}{k}(2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil)}$ for a constant η, implying that

$$Q \geq \frac{1}{2} \cdot \eta \cdot e^{-24\varepsilon^2 \frac{n}{k}(2\,\mathfrak{D} + \lceil \frac{8n}{k} \rceil)},$$

or equivalently

$$\mathfrak{D} \geq \frac{1}{48} \cdot \frac{k}{n}\varepsilon^{-2} \ln \frac{\eta}{2Q} - \frac{1}{2}\left\lceil \frac{8n}{k} \right\rceil.$$

If $Q \leq \frac{\eta}{1000}$, since $\varepsilon^{-1} \geq \frac{4n}{k}$ (from $k \geq 4n\epsilon$), the first term $\frac{1}{48} \cdot \frac{k}{n}\varepsilon^{-2} \ln \frac{\eta}{2Q}$ dominates the second term $\frac{1}{2}\lceil \frac{8n}{k} \rceil$, and thus $\mathfrak{D} = \Omega(\frac{k}{n}\varepsilon^{-2} \ln \frac{1}{Q})$. ($\eta = \sqrt{\frac{\pi}{320}} \cdot e^{-24}$ as stated in Theorem 26.)

It remains to analyze the case that $Q > \frac{\eta}{1000}$, for which we construct an auxiliary algorithm that solves the FT-APX(k, ε) problem with probability at least $1 - \frac{\eta}{1000}$. We will use $\mathcal{A}$ and $\tilde{\mathcal{A}}$ to denote the original algorithm and the auxiliary algorithm, respectively. Recall that $\mathcal{A}$ solves the FT-APX(k, ε) problem with probability at least $1 - Q$. Select k' such that A outputs a small element with probability at most $\frac{k'}{n} - \frac{1-Q}{2}$ and a large element with probability at most $1 - \frac{k'}{n} - \frac{1-Q}{2}$. Thus, by using $\mathcal{A}$ to get sampled elements instead of sampling from the input, the FT-APX(k, ε) problem is reduced to the FT-APX$\big(k', \frac{1-Q}{2}\big)$ problem (with the restriction that we may only use sampled elements). Motivated by this, let $\tilde{\mathcal{A}}$ be a modified (fault-free) version of our algorithms (Section 3–5) for the FT-APX$(k', \frac{1-Q}{2})$ problem with success probability at least $1 - \frac{\eta}{1000}$ in which each sampling from S is implemented by calling $\mathcal{A}$ on S. The correctness of $\tilde{\mathcal{A}}$ relies on the fact that our algorithms only sample elements from S uniformly at random and the corresponding analysis only cares about the probability of getting a small / relevant / large element.

[1] The value of ε should be at least $\frac{1}{n}$; otherwise, the problem becomes the *exact* selection.

As applying our algorithm to solve the FT-APX($k', \frac{1-Q}{2}$) problem with probability $1 - \frac{\eta}{1000}$ would sample $O(\frac{k'}{n}(1-Q)^{-2}\log\frac{1000}{\eta})$ times from S, $\tilde{\mathcal{A}}$ invokes $\mathcal{A}$ at most $O(\frac{k'}{n}(1-Q)^{-2}\log\frac{1000}{\eta})$ times and thus performs *expected* $O(\mathfrak{D}\frac{k'}{n}(1-Q)^{-2}\log\frac{1000}{\eta})$ comparisons. Since all terms except $\mathfrak{D}$ are bounded from above by a constant, the above bound is can be reformulated as $O(\mathfrak{D})$. On the other hand, we have already proven that the expected number of comparison to solve the FT-APX(k, ε) problem with probability at least $1 - \frac{1000}{\eta}$ is $\Omega(\frac{k}{n}\varepsilon^{-2}\log\frac{1000}{\eta}) = \Omega(\frac{k}{n}\varepsilon^{-2})$. Since the first bound $O(\mathfrak{D})$ is an upper bound for the second bound $\Omega(\frac{k}{n}\varepsilon^{-2})$, $\mathfrak{D} = \Omega(\frac{k}{n}\varepsilon^{-2}) = \Omega(\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q})$. Recall that $\log\frac{1}{Q}$ is a constant since $Q \geq \frac{\eta}{1000}$ and η is an absolute constant.

To sum up, when $4\varepsilon \leq \frac{k}{n} \leq \frac{1}{2}$ and $k \geq 200$,the expected number of comparisons required by any algorithm that solve FT-APX(k, ε) with probability $1 - Q$ is

$$\Omega\left(\min\{n, \frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q}\}\right). \qquad \blacktriangleleft$$

If $\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q} = w(n)$, the lower bound in Theorem 20 becomes just $\Omega(n)$. By reducing the approximate selection problem to the exact selection problem, we can show a stronger lower bound in this case as the following theorem.

▶ **Theorem 21.** *If $Q < \frac{1}{2}$ and $\frac{k}{n}\varepsilon^{-2}\log\frac{1}{Q} = w(n)$, then the expected number of comparisons performed by any randomized algorithm that solves FT-APX(k, ε) with probability at least $1 - Q$ is*

$$\Omega\left(\max\left\{n, \varepsilon^{-1}\log\frac{\frac{k+n\varepsilon}{2n\varepsilon}}{Q}\right\}\right).$$

Proof. The first term n directly comes from the first term n of Theorem 20. Recall that we assume $k \leq \frac{n}{2}$. The second term $\varepsilon^{-1}\log\frac{(k+n\varepsilon)/(2n\varepsilon)}{Q}$ can be reduced from the lower bound $\Omega(n\log\frac{k}{Q})$ for the exact k-th smallest element selection problem [9] as follows. Note that as remarked in [9, Section 1], their bound holds both in expectation and in the worst case.

Assume we attempt to select the ℓ-th smallest element among m elements. We can duplicate each element $2 \cdot n\varepsilon$ times and solve the FT-APX$\big(k, \varepsilon\big)$ problem where $n = m \cdot n\varepsilon$ and $k = (2n\varepsilon)\cdot\ell - n\varepsilon$. This setting implies that $m = \varepsilon^{-1}$ and $\ell = (k + n\varepsilon)/(2n\varepsilon)$. Since selecting the ℓ-th smallest element among m elements with probability at least $1 - Q$ requires $\Omega(m\log\frac{\ell}{Q})$ comparisons, a lower bound of $\Omega(\varepsilon^{-1}\log\frac{(k+n\varepsilon)/(2n\varepsilon)}{Q})$ follows. $\blacktriangleleft$

▶ Remark 22. For the proof of Lemma 17, the main observation is that the number of small (or large) elements in A has a hypergeometric distribution. The probability density function of the hypergeometric distribution can be expressed explicitly with binomial coefficients. By the entropy bound for binomial coefficients and a second order tangent bound based on the second derivative, a useful tool (Theorem 32 in Appendix E.4 of the full version [17]) follows, and induces a first-tail bound for the hypergeometric distribution (Theorem 26 in Appendix D.2), from which Lemma 17 follows.

References

1 Martin Aigner. Finding the maximum and minimum. *Discrete Applied Mathematics*, 74(1):1–12, 1997.

2 Amitava Bagchi. On sorting in the presence of erroneous information. *Information Processing Letters*, 43(4):213–215, 1992.

3 Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *Proceedings of the Twenty-fifth Symposium on Theory of Computing (STOC93)*, pages 130–136, 1993.

4 Mark Braverman, Jieming Mao, and S. Matthew Weinberg. Parallel algorithms for select and partition with noisy comparisons. In *Proceedings of the Forty-eighth48th Symposium on Theory of Computing (STOC16)*, pages 851–862, 2016.

5 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *Proceedings of the Nineteenth Symposium on Discrete Algorithms (SODA08)*, pages 268–276, 2008.

6 Xi Chen, Sivakanth Gopi, Jieming Mao, and Jon Schneider. Competitive analysis of the top-k ranking problem. In *Proceedings of the Twenty-Eighth Symposium on Discrete Algorithms (SODA17)*, pages 1245–1264, 2017.

7 Hyungmin Cho, Larkhoon Leem, and Subhasish Mitra. ERSA: error resilient system architecture for probabilistic applications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 31(4):546–558, 2012.

8 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms - Reliable Computation with Unreliable Information*. Monographs in Theoretical Computer Science. Springer, 2013.

9 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.

10 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. *Theoretical Computer Science*, 410(44):4457–4470, 2009.

11 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Sorting with recurrent comparison errors. In *Proceedings of the Twenty-Eighth International Symposium on Algorithms and Computation (ISAAC17)*, pages 38:1–38:12, 2017.

12 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal sorting with persistent comparison errors. In *Proceedings of the Twenty-seventh European Symposium on Algorithms (ESA19)*, pages 49:1–49:14, 2019.

13 Barbara Geissmann, Stefano Leucci, Chih-Hung Liu, and Paolo Penna. Optimal dislocation with persistent errors in subquadratic time. *Theory Comput. Syst.*, 64(3):508–521, 2020.

14 Barbara Geissmann, Matús Mihalák, and Peter Widmayer. Recurring comparison faults: Sorting and finding the minimum. In *Proceedings of the Twentieth International Symposium on Fundamentals of Computation Theory (FCT15)*, pages 227–239, 2015.

15 Ofer Grossman and Dana Moshkovitz. Amplification and derandomization without slowdown. *SIAM Journal on Computing*, 49(5):959–998, 2020.

16 Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *18th IEEE European Test Symposium (ETS)*, pages 1–6, 2013.

17 Shengyu Huang, Chih-Hung Liu, and Daniel Rutschman. Approximate selection with unreliable comparisons in optimal expected time. *CoRR*, abs/2205.01448, 2022. `doi:10.48550/arXiv.2205.01448`.

18 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the Thirty-nineth Symposium on Theory of Computing (STOC07)*, pages 95–103, 2007.

19 Christoph M. Kirsch and Hannes Payer. Incorrect systems: it's not the problem, it's the solution. In *Proceedings of the 49th Design Automation Conference 2012 (DAC)*, pages 913–917, 2012.

20 Rolf Klein, Rainer Penninger, Christian Sohler, and David P. Woodruff. Tolerant algorithms. In *Proceedings of the Nineteenth European Symposium on Algorithms (ESA11)*, pages 736—-747, 2011.

21 K. B. Lakshmanan, Bala Ravikumar, and K. Ganesan. Coping with erroneous information while sorting. *IEEE Transactions on Computers*, 40(9):1081–1084, 1991.

22 Tom Leighton and Yuan Ma. Tight bounds on the size of fault-tolerant merging and sorting networks with destructive faults. *SIAM Journal on Computing*, 29(1):258–273, 1999.

23 Stefano Leucci and Chih-Hung Liu. Approximate minimum selection with unreliable comparisons in optimal expected time. *Algorithmica*, 84(1):60–84, 2022.

24 Stefano Leucci, Chih-Hung Liu, and Simon Meierhans. Resilient dictionaries for randomly unreliable memory. In *Proceedings of the 27th Annual European Symposium on Algorithms, (ESA19)*, pages 70:1–70:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

25 Philip M. Long. Sorting and searching with a faulty comparison oracle. Technical report, University of California at Santa Cruz, 1992.

26 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Sorting noisy data with partial information. In *Proceedings of the Fourth Conference on Innovations in Theoretical Computer Science (ITCS13)*, pages 515–528, 2013.

27 M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2 edition, 2017.

28 Krishna Palem and Avinash Lingamneni. Ten years of building broken chips: The physics and engineering of inexact computing. *ACM Transactions on Embedded Computing Systems*, 12(2s):87:1–87:23, 2013.

29 Andrzej Pelc. Searching with known error probability. *Theoretical Computer Science*, 63(2):185–202, 1989.

30 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theoretical Computer Science*, 270(1-2):71–109, 2002.

31 Bala Ravikumar, K. Ganesan, and K. B. Lakshmanan. On selecting the largest element in spite of erroneous information. In *Proceedings of the fourth Symposium on Theoretical Aspects of Computer Science (STACs87)*, pages 88–99, 1987.

32 Joseph Sloan, John Sartori, and Rakesh Kumar. On software design for stochastic processors. In *Proceedings of the 49th Annual Design Automation Conference 2012 (DAC)*, pages 918–923, 2012.

A Supplementary material for Section 2

▶ **Lemma 4** (Majority Vote). *For any error probability $p \in [0, \frac{1}{2})$, there exists a postive integer c_p such that a strategy that compares two elements $2c_p \cdot t + 1$ times and returns the majority result succeeds with probability at least $1 - e^{-t}$, where $c_p = \lceil \frac{4(1-p)}{(1-2p)^2} \rceil$. The exact failure probability of this strategy is*

$$\sum_{i=0}^{c_p \cdot t} \binom{2c_p \cdot t + 1}{i} (1-p)^i p^{2c_p \cdot t + 1 - i}.$$

Proof. Let $\{X_i \mid 1 \le i \le 2c_p \cdot t + 1\}$ be $2c_p \cdot t + 1$ *independent* Bernoulli random variables such that $X_i = 1$ if the i-th comparison succeeds, i.e., $\Pr[X_i = 1] = 1 - p$ and $\Pr[X_i = 0] = p$. Let $X = \sum_{i=1}^{2c_p \cdot t + 1} X_i$. Then, $E[X] = (2c_p \cdot t + 1)(1-p)$. Since $p < \frac{1}{2}$, we know $2(1-p) > 1$ and we can apply Lemma 5 to prove the first statement as follows:

$$\begin{aligned}
\Pr[X \le \frac{2c_p \cdot t + 1}{2}] &= \Pr[X \le \frac{1}{2(1-p)} E[X]] = \Pr[X \le \left(1 - \frac{1-2p}{2-2p}\right) E[X]] \\
&\underbrace{\le \exp\left(-\frac{1}{2} \cdot (\frac{1-2p}{2-2p})^2 \cdot E[X]\right)}_{\text{Lemma 5}} \\
&= \exp\left(-\frac{1}{2} \cdot (\frac{1-2p}{2-2p})^2 \cdot (2c_p \cdot t + 1)(1-p)\right) \\
&= \exp\left((2c_p \cdot t + 1)\frac{(1-2p)^2}{8(1-p)}\right) < \exp\left(-c_p t \frac{(1-2p)^2}{4(1-p)}\right).
\end{aligned}$$

which satisfies the statement if we choose $c_p = \lceil \frac{4(1-p)}{(1-2p)^2} \rceil$. Since X is a binomial random variable and c_p is an integer, the second statement comes as follows:

$$\Pr[X \leq \frac{2c_p \cdot t + 1}{2}] = \Pr[X \leq c_p \cdot t] = \sum_{i=0}^{c_p \cdot t} \binom{2c_p \cdot t + 1}{i} (1-p)^i p^{2c_p \cdot t + 1 - i}. \quad \blacktriangleleft$$

▶ **Lemma 5** (Chernoff Bound). *Let X be the sum of independent Bernoulli random variables. If $A \leq E[X] \leq B$, then for any $\delta \in (0, 1)$,*

$$\Pr[X \geq (1+\delta) \cdot B] \leq e^{-\frac{\delta^2}{3} B} \qquad \text{and} \qquad \Pr[X \leq (1-\delta) \cdot A] \leq e^{-\frac{\delta^2}{2} A}.$$

Proof. The two statements can be extended from the proofs of [27, Theorem 4.4(2)] and [27, Theorem 4.5(2)], respectively. Here, we only state the difference. Since X is the sum of *independent* Bernoulli random variables, by [27, Section 4.2.1]

$$E[e^{tX}] \leq e^{(e^t - 1)E[X]}.$$

For the first claim, using any $\boldsymbol{t > 0}$,

$$\Pr[X \geq (1+\delta) \cdot B] = \Pr[e^{tX} \geq e^{t(1+\delta) \cdot B}] \leq \frac{E[e^{tX}]}{e^{t(1+\delta)B}} \leq \frac{e^{(e^t-1)E[X]}}{e^{t(1+\delta)B}} \overset{E[X] \leq B}{\leq} \frac{e^{(e^t-1)B}}{e^{t(1+\delta)B}}.$$

The remaining steps are identical to the proof of [27, Theorem 4.4(2)].

For the second claim, using any $\boldsymbol{t < 0}$,

$$\Pr[X \leq (1-\delta) \cdot A] = \Pr[e^{tX} \geq e^{t(1-\delta) \cdot A}] \leq \frac{E[e^{tX}]}{e^{t(1-\delta)A}} \leq \frac{e^{(e^t-1)E[X]}}{e^{t(1-\delta)A}} \overset{A \leq E[X]}{\leq} \frac{e^{(e^t-1)A}}{e^{t(1+\delta)A}}.$$

The remaining steps are identical to the proof of [27, Theorem 4.5(2)]. ◀

B Supplementary material for Section 3

▶ **Lemma 23.** *Let $m = 2^{10} \cdot 3^2 \cdot \ln \frac{2}{Q}$, let $X_1, X_2 \ldots, X_m$ be m identically and independently distributed Bernoulli random variables with probability $p \geq \frac{8}{9}$, and let $X = \sum_{i=1}^{m} X_i$.*

$$\Pr[X \geq \frac{7}{8}m] \geq 1 - \frac{Q}{2}.$$

Proof. It is sufficient to prove that $\Pr[X \leq \frac{7}{8}m] \leq \frac{Q}{2}$. Since $p \geq \frac{8}{9}$, $E[X] \geq \frac{8}{9}m$. By Lemma 5,

$$\begin{aligned} \Pr[X \leq \frac{7}{8}m] &= \Pr[X \leq (1 - \frac{1}{64}) \cdot \frac{8}{9}m] \overset{\text{Lemma 5}}{\leq} \exp\left(-\frac{1}{2} \cdot (\frac{1}{64})^2 \cdot \frac{8}{9}m\right) \\ &= \exp\left(-\frac{1}{2^{10} \cdot 3^2}m\right) \leq \exp\left(-\frac{2^{10} \cdot 3^2 \cdot \ln \frac{2}{Q}}{2^{10} \cdot 3^2}\right) = e^{-\ln \frac{2}{Q}} = \frac{Q}{2}. \end{aligned} \quad \blacktriangleleft$$

▶ **Theorem 6.** *It takes **expected** $O(\frac{k}{n}\varepsilon^{-2} \log \frac{1}{Q})$ comparisons to solve the FT-APX(k, ε) problem with probability at least $1 - Q$.*

Proof. Let $m = 2^{10} \cdot 3^2 \cdot \ln \frac{2}{Q}$ as in Lemma 23. The algorithm consists of two stages. The first stage aims to select m elements in which all elements in the range $(\frac{1}{8}m, \frac{7}{8}m]$ are relevant, and the second stage aims to select an element from $(\frac{1}{8}m, \frac{7}{8}m]$.

For the number of comparisons, by Theorem 15, it takes $O(\frac{k}{n}\varepsilon^{-2})$ comparisons to select a relevant element with probability at least $1 - \frac{1}{9}$, so the first stage takes $O(\frac{k}{n}\varepsilon^{-2} \cdot m) = O(\frac{k}{n}\varepsilon^{-2} \cdot \log \frac{1}{Q})$ comparisons. For the second stage, by Section 3, one verification step performs $O(\log \frac{1}{Q})$ comparisons. To derive the expected total number of comparisons, we need to calculate the probability of conducting the i-th round. Since the probability of picking an element in $(\frac{2}{8}m, \frac{6}{8}m]$ is $\frac{1}{2}$ at any round and such an element is verified in $(\frac{1}{8}m, \frac{7}{8}m]$ with probability at least $1 - \frac{Q}{2} \geq \frac{1}{2}$ at any round, any round returns an element with probability at least $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Similar to geometric distribution, the probability that the i-th round is conducted is at most $(1 - \frac{1}{4})^{i-1} = (\frac{3}{4})^{i-1}$, so the second stage takes expected $\sum_{i=1}^{\infty}(\frac{3}{4})^{i-1}O(\log \frac{1}{Q}) = O\big(\log \frac{1}{Q} \cdot \sum_{i=1}^{\infty}(\frac{3}{4})^{i-1}\big) = O(\log \frac{1}{Q})$ comparisons. To sum up, the algorithm takes expected $O(\frac{k}{n}\varepsilon^{-2} \cdot \log \frac{1}{Q})$ comparisons.

For the success probability, by Theorem 15 and Lemma 23, the first stage fails with probability at most $\frac{Q}{2}$. The second stage fails only when returning an element in $[1, \frac{1}{8}m]$ or $(\frac{7}{8}m, m]$. Since a single round picks an element in $[1, \frac{1}{8}m] \cup (\frac{7}{8}m, m]$ with probability $\frac{1}{4}$ and the verification fails with probability at most $\frac{Q}{2}$. a single round returns an element in $[1, \frac{1}{8}m] \cup (\frac{7}{8}m, m]$ with probability *at most* $\frac{1}{4} \cdot \frac{Q}{2} = \frac{Q}{8}$, Therefore, the failure probability of the second stage is at most $\sum_{i \geq 1}(\frac{3}{4})^{i-1} \cdot \frac{Q}{8} = \frac{Q}{2}$, concluding the statement. ◀

C Supplementary material for Section 4

▶ **Lemma 9.** *For **three** elements, consider the following median selection algorithm:*

1. *For each pair of elements, apply the majority vote strategy with $2c_p \cdot 4 + 1$ comparisons (Lemma 4), and assign a point to the element that attains the majority result.*
2. *Return the element with exactly one point. If all three elements get exactly one point, return one of them uniformly at random.*

The above algorithm returns the median with probability at least $1 - \frac{1}{13}$, and returns the minimum and the maximum with the same probability, i.e., at most $\frac{1}{26}$.

Proof. Let q be the failure probability of one majority vote. Since one majority vote consists of $2c_p \cdot 4 + 1$ comparisons, by Lemma 4, $q \leq e^{-4}$. If all three majority votes succeed, then the algorithm will return the median, implying that the algorithm will return the median with probability at least $(1-q)^3 \geq 1 - 3q \geq 1 - 3 \cdot e^{-4} \geq 1 - \frac{1}{13}$.

Now, we will prove that the algorithm returns the minimum and the maximum with the same probability. Since there are three majority votes, there are 8 possibilities, and these 8 possibilities lead to four different situations: exactly the minimum or exactly the median or exactly the maximum gets one point, or all the three elements get one point. A tree diagram for these 8 possibilities can easily calculate the probabilities of the four situations. In detail, exactly the minimum (resp. exactly the maximum) gets one point with probability $q(1-q)$, exactly the median gets one point with probability $(1-q)^3 + q^3$, and all three elements get one point with probability $q(1-q)$. Since the algorithm returns an element uniformly at random when all the three elements get one point, the algorithm returns the minimum and the maximum with the same probability $\frac{4}{3}q(1-q)$.

Since the algorithm returns the median with probability at least $1 - \frac{1}{13}$ and returns the minimum and the maximum with the same probability, the probability that the algorithm returns the minimum (resp. the maximum) is at most $\frac{1}{26}$. ◀

D Supplementary material for Section 6

D.1 Sampling Lemma

This subsection aims to build up a sampling bound (Corollary 18) that is the key ingredient to prove Lemma 19. Corollary 18 roughly states that for a set A of randomly sampled elements (without replacement), the probability that an element of a certain rank in A is NOT relevant decreases as $e^{-\Omega(\frac{\varepsilon^2}{\beta}|A|)}$. To prove Corollary 18, we first derive Lemma 17 that deals with different positions in A. For ease of exposition, we also use β to denote $\frac{k}{n}$ in the proofs. As assumed in the whole paper, $\beta \leq \frac{1}{2}$, and as stated in Section 6, it is also sufficient to consider $\beta \geq 4\varepsilon$ since if $\beta < 4\varepsilon$, we then can apply the lower bound for the approximate minimum selection [23].

▶ **Lemma 17.** *Let A consist of $m \leq \frac{n}{4}$ elements sampled from S without replacement. Suppose that $m\beta \geq 8$ and that $\frac{1}{2} \geq \beta \geq 4\varepsilon$. Then, there is an absolute constant $\eta = \sqrt{\frac{\pi}{320}} \cdot e^{-24}$, coming from Theorem 26, with the following properties:*

1. *Let u be the r-th smallest element of A. If $r \leq \lceil \beta m \rceil$, then u is small with probability at least*

$$\eta \cdot e^{-12\frac{\varepsilon^2}{\beta(1-\beta)}m}.$$

2. *Let v be the r-th largest element of A. If $r \leq \lceil (1-\beta)m \rceil$, then v is large with probability at least*

$$\eta \cdot e^{-12\frac{\varepsilon^2}{\beta(1-\beta)}m}.$$

Proof. We first prove (1). Let X denote the number of small elements in A. Then $X \sim \text{Hypergeom}(n, (\beta-\varepsilon)k, m)$ has a hypergeometric distribution (Definition 24 in Appendix D.2). Since $r \leq \lceil \beta m \rceil$, u is small if and only if A contains at least r small elements, i.e., if and only if $X \geq r$. Put $a = \beta$ and b $= \beta - \varepsilon$. Then we have $a \leq \frac{8}{5}b$ and $(1-a) \leq \frac{8}{5}(1-b)$ as $\beta \geq 4\varepsilon$. As $m\beta \geq 8$ and $\beta \leq \frac{1}{2}$, we also have $ma(1-a) \geq 4$. Hence by Theorem 26

$$\Pr[X \geq r] \geq \Pr[X \geq \lceil \beta m \rceil] = \Pr[X \geq \beta m] \geq \eta \cdot e^{-6\frac{\varepsilon^2}{b(1-b)}}$$

for some absolute constant $\eta = \sqrt{\frac{\pi}{320}} \cdot e^{-24}$. Since $\beta \geq 2\varepsilon$, we have $b \geq \frac{\beta}{2}$, and since we also have $(1-b) \geq (1-\beta)$, we have

$$\frac{\varepsilon^2}{b(1-b)} \leq \frac{2\varepsilon^2}{\beta(1-\beta)},$$

implying that

$$\Pr[X \geq r] \geq \eta \cdot e^{-12\frac{\varepsilon^2}{\beta(1-\beta)}m}$$

The proof of (2) is symmetric with large elements instead of small ones and with $(1-\beta)$ instead of β. ◀

As every element is either among the $\lceil \beta m \rceil$ smallest or among the $\lceil (1-\beta)m \rceil$ largest ones, the lemma directly implies the following.

▶ **Corollary 18.** *Let A consist of $m \leq \frac{n}{4}$ elements sampled from S* without *replacement. Suppose that $m\beta \geq 8$ and that $\frac{1}{2} \geq \beta \geq 4\varepsilon$. Then, an arbitrary element u in A is NOT relevant with probability at least*

$$\eta \cdot e^{-24 \cdot \frac{n}{k} \cdot \varepsilon^2 \cdot m}$$

for an absolute constant $\eta = \sqrt{\frac{\pi}{320}} \cdot e^{-24}$ coming from Theorem 26.

Proof. Let r be the rank of u in A. If $r \leq \lceil \beta m \rceil$, then by part (1) of Lemma 17, u is small with probability at least

$$\eta \cdot e^{-12 \frac{\varepsilon^2}{\beta(1-\beta)} m}.$$

Otherwise, $r \geq \lceil \beta m \rceil + 1 \geq \beta m + 1$, so $m + 1 - r \leq (1-\beta)m \leq \lceil (1-\beta)m \rceil$. Since u is the $(m+1-r)$-th largest element of A, by part (2) of Lemma 17, u is large with probability at least

$$\eta \cdot e^{-12 \frac{\varepsilon^2}{\beta(1-\beta)} m}.$$

Since $1 - \beta \geq \frac{1}{2}$, we have

$$\eta \cdot e^{-12 \frac{\varepsilon^2}{\beta(1-\beta)} m} \geq \eta \cdot e^{-24 \frac{\varepsilon^2}{\beta} m} = \eta \cdot e^{-24 \cdot \frac{k}{n} \cdot \varepsilon^2 \cdot m}.$$ ◀

D.2 A lower tail for hypergeometric distribution

▶ **Definition 24.** *Consider M balls, out of which K balls are black and $M - K$ balls are white.* $\mathrm{Hypergeom}(M, K, m)$ *is the probability distribution for the number of black balls in m draws from the M balls using sampling without replacement, which is the so-called* hypergeometric distribution. *$X \sim \mathrm{Hypergeom}(M, K, m)$ means that X is a random variable with* $\mathrm{Hypergeom}(M, K, m)$ *distribution.*

Due to the page limit, we omit the proof of Corollary 25; please see the full version [17].

▶ **Corollary 25.** *Let $X \sim \mathrm{Hypergeom}(M, K, m)$. Let $0 < \ell < m$ be an integer. Put $a = \frac{\ell}{m}$, $b = \frac{K}{M}$ and $x = \frac{m}{M}$. If $a \leq 2b$, $(1-a) \leq 2(1-b)$ and $x \leq \frac{1}{4}$, then we have*

$$Pr[X = \ell] \geq \sqrt{\frac{\pi}{64ma(1-a)}} \cdot e^{-3\frac{(a-b)^2}{b(1-b)}m}.$$

▶ **Theorem 26.** *Let $X \sim \mathrm{Hypergeom}(M, K, m)$. Let $0 \leq \ell \leq m$ be a real number with $\ell < K$ and $m - \ell < M - K$. Put $a = \frac{\ell}{m}$, $b = \frac{K}{M}$ and $x = \frac{m}{M}$. If $a \leq \frac{8}{5}b$, $(1-a) \leq 2(1-b)$, $x \leq \frac{1}{4}$ and $ma(1-a) \geq 4$, then we have*

$$Pr[X \geq \ell] \geq \sqrt{\frac{\pi}{320}} \cdot e^{-24} \cdot e^{-\frac{6(a-b)^2}{b(1-b)}m}.$$

Proof. Let $0 \leq t \leq \sqrt{ma(1-a)}$ be a real number such that $\ell + t$ is an integer and put $a' = \frac{\ell+t}{m}$. As $ma(1-a) \geq 4$, we have $t \leq \sqrt{ma(1-a)} \leq \frac{ma(1-a)}{4}$, so that

$$a' = a + \frac{t}{m} \leq a + \frac{a(1-a)}{4} \leq \frac{5}{4}a \leq 2b,$$

and $1 \leq a' \leq 1-a \leq 2(1-b)$. We may hence apply Corollary 25 and get

$$\begin{aligned}\Pr[X = \ell + t] &\geq \sqrt{\frac{\pi}{64\big(ma'(1-a')\big)}} \cdot e^{-3\frac{(a+\frac{t}{m}-b)^2}{b(1-b)}m} \\ &\geq \sqrt{\frac{\pi}{80ma(1-a)}} \cdot e^{\frac{-3(a+\frac{t}{m}-b)^2}{b(1-b)}m}\end{aligned}$$

where we used that

$$a'(1-a') \leq \frac{5}{4}a(1-a).$$

Since $(a+\frac{t}{m}-b)^2 \leq 2(a-b)^2 + 2(\frac{t}{m})^2$, we have

$$\frac{3(a+\frac{t}{m}-b)^2}{b(1-b)}m \leq \frac{6(a-b)^2}{b(1-b)}m + \frac{6t^2}{mb(1-b)}$$

where

$$\frac{6t^2}{mb(1-b)} \leq \frac{6ma(1-a)}{mb(1-b)} = 6\frac{a(1-a)}{b(1-b)} \leq 24.$$

Hence we have

$$\Pr[X = \ell + t] \geq \sqrt{\frac{\pi}{80\big(ma(1-a)-t\big)}} \cdot e^{-\frac{6(a-b)^2}{b(1-b)}m} \cdot e^{-24}.$$

There are at least $\sqrt{ma(1-a)} - 1$ possible values of t. As $ma(1-a) \geq 4$, we have

$$\sqrt{ma(1-a)} - 1 \geq \frac{\sqrt{ma(1-a)}}{2}.$$

Thus summing over all possible possible values of t yields the statement. ◀

Relating Description Complexity to Entropy

Reijo Jaakkola
Tampere University, Finland

Antti Kuusisto
Tampere University, Finland
University of Helsinki, Finland

Miikka Vilander
Tampere University, Finland

Abstract

We demonstrate some novel links between entropy and description complexity, a notion referring to the minimal formula length for specifying given properties. Let MLU be the logic obtained by extending propositional logic with the universal modality, and let GMLU be the corresponding extension with the ability to count. In the finite, MLU is expressively complete for specifying sets of variable assignments, while GMLU is expressively complete for multisets. We show that for MLU, the model classes with maximal Boltzmann entropy are the ones with maximal description complexity. Concerning GMLU, we show that expected Boltzmann entropy is asymptotically equivalent to expected description complexity multiplied by the number of proposition symbols considered. To contrast these results, we prove that this link breaks when we move to considering first-order logic FO over vocabularies with higher-arity relations. To establish the aforementioned result, we show that almost all finite models require relatively large FO-formulas to define them. Our results relate to links between Kolmogorov complexity and entropy, demonstrating a way to conceive such results in the logic-based scenario where relational structures are classified by formulas of different sizes.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Finite Model Theory

Keywords and phrases finite model theory, entropy, formula size, randomness, formula size game

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.38

Funding Antti Kuusisto was supported by the Academy of Finland project *Theory of computational logics*, grant numbers 352419, 352420, 353027, 324435, 328987. Furthermore, Antti Kuusisto and Miikka Vilander were supported by the Academy of Finland consortium project *Explaining AI via Logic* (XAILOG), grant number 345612.

1 Introduction

In this article we investigate links between description complexity and entropy. By description complexity of a model, we mean the minimal length of a formula that specifies the model up to a maximal possible extent. With a strong enough logic, this amounts to investigating the length of formulas specifying models up to isomorphism, but this is by no means the only interesting scenario. By the description complexity of a class of models, we mean the minimal length of a formula defining that class. In this paper we are particularly interested in the description complexity of completely specified model classes, i.e., equivalence classes of logics. The main objective of the paper is to point out links between description complexity and entropy. By entropy, we refer essentially to Shannon's entropy and the earlier notion of Boltzmann entropy from statistical mechanics.

We first consider models with unary relational vocabularies. We study two related logics, MLU and GMLU. The logic MLU is the extension of propositional logic with the universal modality $\blacklozenge$, also known as global modality. The truth definition states that $\mathfrak{M}, w \vDash \blacklozenge\varphi$ if

© Reijo Jaakkola, Antti Kuusisto, and Miikka Vilander;
licensed under Creative Commons License CC-BY 4.0

$\mathfrak{M}, u \vDash \varphi$ for some u in the domain of $\mathfrak{M}$. Thus, in the finite, this logic is tuned to specify precisely which variable assignments are present in the model considered. The system GMLU is the extension of MLU with the ability to count: we have $\mathfrak{M}, w \vDash \blacklozenge^{\geq d}\varphi$ if $\mathfrak{M}, u \vDash \varphi$ for at least d points u in the domain of $\mathfrak{M}$. We note that when limiting to models with a finite unary vocabulary and a fixed finite bound on domain size, GMLU is expressively complete, being able to define all classes of models closed under isomorphism. While MLU can fully specify which *set* of assignments is present in a model, GMLU can lift this specification to the level of *multisets*.

Let τ be a finite unary relational vocabulary, and let $\mathrm{Mod}_n(\tau)$ denote the class of τ-models over the fixed domain $W = \{1, \ldots, n\}$. Let $\equiv_{\mathrm{MLU}}$ and $\equiv_{\mathrm{GMLU}}$ denote the logical equivalence relations of MLU and GMLU over $\mathrm{Mod}_n(\tau)$. We first prove that among the classes of $\equiv_{\mathrm{MLU}}$, the class with the largest description complexity is the class with the largest Boltzmann entropy. This means that the models with the largest description complexity belong to the class that has the largest Boltzmann entropy. We then move on to investigating GMLU. Let $\langle H_B \rangle$ denote the expected Boltzmann entropy over the equivalence classes of $\equiv_{\mathrm{GMLU}}$, with the probability of an individual class being its size divided by the size of $\mathrm{Mod}_n(\tau)$. Let $\langle C \rangle$ denote the expected description complexity of a model chosen randomly from $\mathrm{Mod}_n(\tau)$, and let $|\tau|$ denote the size of the vocabulary τ. We will prove that

$$\langle H_B \rangle \sim |\tau| \langle C \rangle \tag{1}$$

that is, $\langle H_B \rangle$ is asymptotically equivalent to $|\tau|\langle C \rangle$. This gives an intimate relationship between $\langle C \rangle$ and Boltzmann entropy. To obtain a link to Shannon entropy, we simply note that the Shannon entropy of the distribution of models based on $\equiv_{\mathrm{GMLU}}$ is equal to $\langle H_B \rangle - \log(|\mathrm{Mod}_n(\tau)|)$.

We then move on to investigating general (finite) relational vocabularies. Our main result there is that the expected description complexity of classes of FO grows asymptotically faster with domain size than the corresponding expected Boltzmann entropy. To establish this result, we show that almost all models require relatively large FO-formulas to define them.

Concerning related work, there exist well known relationships between Kolmogorov complexity and entropy. Notably, for any computable distribution, the expected Kolmogorov complexity can be linked, within a constant, to Shannon entropy. See for example [6, 9, 10, 14] for discussions of the issue. The article [14] discusses some generalizations and shows, e.g., that the relationship fails in the general case for Rényi and Tsallis entropies. Links between description lengths and entropy are fundamentally interesting, linking syntactic issues to semantic randomness. Most notable results in the field concern variants of Kolmogorov complexity. The aim of the current article is to provide one way of demonstrating how these results extend beyond the realm of binary strings and descriptions via programs. The link given in Equation (1) elucidates nicely the relationship between the syntax of GMLU and models with unary vocabularies. The result on FO provides contrast to this and warns against overselling the analogy between description complexities and entropy. However, we conjecture that even for FO, a monotone Galois connection can be demonstrated between description complexities of a relevant collection of classes and related Boltzmann entropies, but this is left for future work for lack of space.

Links between description complexity and the properties of described model classes relate also to the relationship between classifier size and classified data. This topic is relevant, for example, from the point of view of current research on explainability in AI. For work on this topic, see, e.g., [8, 2].

Concerning other related work, we turn attention to the proof techniques used in the paper. One of the main tools we use is the framework of logic-related games. We note that standard Ehrenfeucht-Fraïssé games, and their variants such as bisimulation games, do not

suffice for the purposes of this article. Thus we utilize *formula size games* for MLU and GMLU instead. Generally, the first formula size game was defined for propositional logic by Razborov in [13]. A better known version is the game of Adler and Immerman for CTL in [1]. The game for MLU resembles the similar game developed in [7] which was there also used to demonstrate a nonelementary succinctness gap between modal logic and FO. For GMLU, we develop a suitable game by extending the game for MLU. The hard part is using the games in a suitable way. We accomplish this by using the fact that models in classes with large Boltzmann entropy realize a rich number of types. In addition to games, we also use various techniques for estimating Boltzmann entropy and description complexity, e.g., Stirling's approximation, the weak law of large numbers and counting arguments.

2 Preliminaries

Let $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. We use $f = \mathcal{O}(g)$ to denote that $f \leq Cg(n)$, for some constant $C > 0$ and large enough n. If we want to emphasize that the implied constant C depends on some parameter p (which is independent of n), we will write $f = \mathcal{O}_p(g)$. We use $f = \Omega(g)$ to denote that $f(n) \geq Cg(n)$, for some constant $C > 0$ and large enough n. Finally, we use $f = \Theta(g)$ to denote that $f = \mathcal{O}(g)$ and $f = \Omega(g)$. We say that f is **asymptotically** g, if $\lim_{n\to\infty} f/g = 1$ and we denote this by $f \sim g$. By log we mean logarithm to base two.

The following variants of classical results will be useful for our purposes.

▶ **Proposition 1** (Stirling's approximation [5]). $\log(n!) = n\log(n) - n\log(e) + \Theta(\log(n))$

▶ **Proposition 2** (Weak law of large numbers [11]). *Let $(X_n)_{n\in\mathbb{N}}$ be a sequence of Bernoulli random variables with success probability $p := \Pr[X_n = 1]$. Then for every $\delta > 0$ we have that*

$$\lim_{n\to\infty} \Pr\left[\left|p - \tfrac{1}{n}\sum_{i=1}^{n} X_n\right| < \delta\right] = 1.$$

We next define the logics studied in this work. Let $\tau = \{p_1, \ldots, p_k\}$ be a set of proposition symbols. The syntax of **graded universal modal logic** GMLU[τ] is generated as follows.

$$\varphi := \blacklozenge^{\geq d}\psi \mid \blacksquare^{<d}\psi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \blacklozenge^{\geq d}\varphi \mid \blacksquare^{<d}\varphi$$
$$\psi := p \mid \neg p \mid \psi \vee \psi \mid \psi \wedge \psi$$

Here $p \in \tau$ and $d \in \mathbb{N}$. Notice that by design, the formulas of GMLU[τ] only contain proposition symbols that occur in the scope of a global modal operator $\blacklozenge^{\geq d}$ or $\blacksquare^{<d}$. Additionally, all formulas are in negation normal form. (In the sequel, the notation $\neg\varphi$ will always mean the negation normal form formula, where the negation has been pushed to the level of literals.) Now, let $\mathfrak{M}$ be a Kripke model with universe W. The semantics of the global graded modalities are defined as follows: $(\mathfrak{M}, w) \models \blacklozenge^{\geq d}\varphi \Leftrightarrow$ there are at least d points $v \in W$ such that $(\mathfrak{M}, v) \models \varphi$. Additionally, $(\mathfrak{M}, w) \models \blacksquare^{<d}\varphi \Leftrightarrow (\mathfrak{M}, w) \models \neg\blacklozenge^{\geq d}\neg\varphi$. Intuitively this means that all points in $\mathfrak{M}$ satisfy φ, except for less than d exceptions. The rest of the semantics is defined as usual in propositional logic. Note that $\blacklozenge^{\geq d}$ and $\blacksquare^{<d}$ are dual to each other. (We note that in this article, modal logics will always have a strictly unary vocabulary, so Kripke models will not have an accessibility relation as part of the relational structure involved.)

Given a Kripke model $\mathfrak{M}$ over τ and $\varphi \in$ GMLU[τ], we define the point-free truth relation such that $\mathfrak{M} \models \varphi \Leftrightarrow$ for every $w \in W$, we have $\mathfrak{M}, w \models \varphi$. Since no propositional symbol occurs outside the scope of a global modality, $\mathfrak{M} \models \varphi$ iff there is some $w \in W$ for which $\mathfrak{M}, w \models \varphi$. Hence the truth of any formula of GMLU is independent of the evaluation point w. The property that truth is always independent of the evaluation point is the reason we defined

GMLU so that proposition symbols must occur in the scope of modalities. The fragment of GMLU[τ] where $d = 1$ for all modalities is called **universal modal logic** MLU[τ]. This logic has only the modalities $\blacklozenge^{\geq 1}$ and $\blacksquare^{<1}$, and we denote these with $\blacklozenge$ and $\blacksquare$ for simplicity.

A 1-**type** π over τ is a maximally consistent set of literals (propositional symbols and their negations). This means that π has exactly one of p or $\neg p$ for each $p \in \tau$. The set of all 1-types over τ is denoted by $\boldsymbol{\alpha}_\tau$. Given a Kripke model $\mathfrak{M}$ over τ and $w \in W$, we let $\mathrm{tp}_{\mathfrak{M}}[w]$ denote the unique 1-type that w **realizes**.

The **size** of a formula $\varphi \in \mathrm{GMLU}[\tau]$, denoted $size(\varphi)$, is defined as follows:

- $size(\alpha) = 1$ for a literal α,
- $size(\varphi \vee \psi) = size(\varphi \wedge \psi) = size(\varphi) + size(\psi) + 1$,
- $size(\blacklozenge^{\geq d}\varphi) = size(\blacksquare^{<d}\varphi) = size(\varphi) + d$.

We emphasize that according to our definition *all literals have the same size*. The motivation for this is to consider negative (i.e., negated) information and positive (i.e., non-negated) information as equal in relation to formula size. This also explains the convention of defining GMLU such that formulas are in negation normal form.

We will also consider standard first-order logic FO. Let $\tau = \{R_1, \ldots, R_k\}$ be a set of relation symbols. The syntax of FO[τ] is generated by the following grammar:

$$\varphi := x = y \mid \neg x = y \mid R(\overline{x}) \mid \neg R(\overline{x}) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x\varphi \mid \forall x\varphi,$$

where $\overline{x}$ is a tuple of variables. We use the standard semantics of FO[τ]. The **size** of a formula $\varphi \in \mathrm{FO}[\tau]$, denoted $size(\varphi)$, is defined as follows:

- $size(\alpha) = 1$ for a literal α,
- $size(\varphi \vee \psi) = size(\varphi \wedge \psi) = size(\varphi) + size(\psi) + 1$,
- $size(\exists x\varphi) = size(\forall x\varphi) = size(\varphi) + 1$

Again we emphasize that according to our definition, all literals have the same size.

Let $\mathcal{L} = (L, \models)$ be a logic and $\mathcal{M}$ a *finite class of models*. The class $\mathcal{M}$ is here considered fixed and known from the context. We say that a formula $\varphi \in L$ of $\mathcal{L}$ **defines** a set $M \subseteq \mathcal{M}$ if for all $\mathfrak{M} \in \mathcal{M}$, we have $\mathfrak{M} \models \varphi$ iff $\mathfrak{M} \in M$. Such a set M is called $\mathcal{L}$**-definable** (with respect to $\mathcal{M}$). Given an $\mathcal{L}$-definable set M, its $\mathcal{L}$**-description complexity** $C_{\mathcal{L}}(M)$ is the size of a minimum size formula $\varphi \in \mathcal{L}$ which defines M. Now, if $\mathcal{L}$ is closed under negation (as all the logics in this paper are), then the relation "$\mathfrak{M}$ and $\mathfrak{N}$ satisfy the same $\mathcal{L}$-formulas" induces a partition of $\mathcal{M}$ denoted by $\equiv_{\mathcal{L}}$. The $\mathcal{L}$-description complexity *of a model* $\mathfrak{M}$ with respect to $\equiv_{\mathcal{L}}$ is $C_{\equiv_{\mathcal{L}}}(\mathfrak{M}) := C_{\mathcal{L}}(M)$, where M is the equivalence class of $\mathfrak{M}$. For brevity, we formulate the results below only for description complexities of classes rather than models.

For an example of description complexity, consider MLU[τ] for the singleton alphabet $\tau = \{p\}$. Models, where p is true in every point is a class of the partition $\equiv_{\mathrm{MLU}[\tau]}$. The description complexity of this class is 2 as the minimum size formula that defines the class is $\blacksquare p$.

Let $\mathcal{M}$ be a finite class of models and let $\equiv$ be an *arbitrary* equivalence relation over $\mathcal{M}$. Given an equivalence class $M \subseteq \mathcal{M}$, we define its **Boltzmann entropy** as $H_B(M) := \log(|M|)$. This terminology is borrowed from statistical mechanics, where the Boltzmann entropy of a macrostate is the quantity $k_B \ln(\Omega)$. Here k_B is the Boltzmann constant, ln the natural logarithm and Ω the number of microstates associated with the macrostate. Note that in our definition, we use the binary logarithm. As a measure of randomness, it is natural to define the Boltzmann entropy *of a model* $\mathfrak{M}$ as $H^{\equiv}_B(\mathfrak{M}) := H_B(M)$, where M is the equivalence class of $\mathfrak{M}$. This reflects the *informal intuition* that often the randomness of an object x is in fact more related to the size (or richness) of a similarity class of objects that x belongs to rather than to x itself. Consider, for example, the equivalence classes that a sufficiently weak logic defines over the universe of binary strings of a fixed finite length. As a general intuition, it is natural to associate a formula φ (or the class it defines) with a macrostate, while the models of φ are then the corresponding microstates.

Let $\{M_i \mid i \in I\}$ enumerate the equivalence classes of $\equiv$. As they form a partition of $\mathcal{M}$, we have the following natural probability distribution over the equivalence classes: $p_{\equiv}(M_i) := |M_i|/|\mathcal{M}|$. Given a random variable $X : \{M_i \mid i \in I\} \to \mathbb{R}_{\geq 0}$, we use $\langle X \rangle$ to denote its expected value with respect to $p_{\equiv}$. Now, suppose we are in a context where we have fixed a finite universe $\mathcal{M}$ of models. Let $\equiv_{\mathrm{GMLU}} \subseteq \mathcal{M} \times \mathcal{M}$ be the corresponding equivalence relation of GMLU. Suppose $\{M_i \mid i \in I\}$ enumerates the equivalence classes of $\equiv_{\mathrm{GMLU}}$. Recall that $C_{\mathrm{GMLU}}(M_i)$ denotes the GMLU description complexity of the class M_i. Let $p_{\equiv_{\mathrm{GMLU}}}(M_i)$ be the corresponding probability $|M_i|/|\mathcal{M}|$. In this paper, we denote by $\langle C \rangle$ the expected description complexity of GMLU, that is, $\langle C \rangle = \sum_{i \in I} p_{\equiv_{\mathrm{GMLU}}}(M_i) C_{\mathrm{GMLU}}(M_i)$. The class $\mathcal{M}$ will be clear from the context. Note that trivially the same expected value is obtained for the description complexity of *models* over $\mathcal{M}$ if we give every model $\mathfrak{M} \in \mathcal{M}$ the probability $1/|\mathcal{M}|$ (the uniform distribution).

We note that it would be natural to define the Boltzmann entropy of an equivalence relation $\equiv$ as the expected value $\langle H_B \rangle$ of H_B with respect to the above natural probability distribution $p_{\equiv}$. The value $\langle H_B \rangle$ is closely related to the **Shannon entropy** $H_S(\equiv)$ of $\equiv$, which we define as the expected value of the random variable $M_i \mapsto -\log(p_{\equiv}(M_i))$. More explicitly, we define that $H_S(\equiv) := -\sum_{i \in I} p_{\equiv}(M_i) \log(p_{\equiv}(M_i))$. Note that this expression is always well-defined, since $M_i \neq \varnothing$, for every $i \in I$. The following result is established in Appendix A.1. Note that the expected value of $H_B^{\equiv}$ over the uniform distribution on $\mathcal{M}$ is equal to $\langle H_B \rangle$, so the result could also be formulated for single models.

▶ **Proposition 3.** *Let $\mathcal{M}$ be a finite class of models and $\equiv \subseteq \mathcal{M} \times \mathcal{M}$ an equivalence relation over $\mathcal{M}$. Then $H_S(\equiv) + \langle H_B \rangle = \log(|\mathcal{M}|)$.*

There exist results in the literature on entropy similar to the above, see, e.g., [3] and [15]. By the proposition, both the Shannon entropy of $\equiv$ and the expected Boltzmann entropy of $\equiv$ cannot be simultaneously large (meaning close to their maximum value $\log(|\mathcal{M}|)$). Indeed, suppose we do not alter $\mathcal{M}$, so $\log(|\mathcal{M}|)$ is constant. Now suppose we alter $\equiv$ so that $H_S(\equiv)$ is increased. This lowers $\langle H_B \rangle$. Vice versa, increasing $\langle H_B \rangle$ lowers $H_S(\equiv)$. Shannon entropy and expected Bolzmann entropy are complementary quantities, summing to a constant.

3 MLU: The largest class has maximal description complexity

Fix $\tau = \{p_1, \ldots, p_k\}$ and let $\mathrm{Mod}_n(\tau)$ be the set of Kripke models over τ and the *fixed universe* $W = \{1, \ldots, n\}$. In this section we consider the equivalence $\equiv_{\mathrm{MLU}[\tau]}$ as defined above and denote it by $\equiv$. We show that in this canonical partition, the largest class, which is the one with the largest Boltzmann entropy, has maximal MLU[τ]-description complexity.

The equivalence classes of $\equiv$ can be described easily. For Kripke models $\mathfrak{M}_1, \mathfrak{M}_2 \in \mathrm{Mod}_n(\tau)$, we have $\mathfrak{M}_1 \equiv \mathfrak{M}_2 \Leftrightarrow \{\mathrm{tp}_{\mathfrak{M}_1}[w] \mid w \in W\} = \{\mathrm{tp}_{\mathfrak{M}_2}[w] \mid w \in W\}$. That is, each equivalence class is uniquely determined by the 1-types realized in it. As the number of 1-types over τ is 2^k, the number of equivalence classes of $\equiv$ is $2^{2^k} - 1$. Given a set $\Pi \subseteq \boldsymbol{\alpha}_\tau$, we let M_Π be the equivalence class that has the models that realize exactly the 1-types in Π.

Note the equivalence class M_Π of any set $\Pi \subseteq \boldsymbol{\alpha}_\tau$ can be defined by the following formula:

$$\varphi(\Pi) := \bigwedge_{\pi \in \Pi} \Diamond \psi(\pi) \wedge \Box \Big(\bigwedge_{\pi \in \boldsymbol{\alpha}_\tau \setminus \Pi} \neg \psi(\pi) \Big),$$

where $\psi(\pi)$ is the conjunction of the literals in the 1-type π. For $\Pi \neq \boldsymbol{\alpha}_\tau$, the size of the formula $\varphi(\Pi)$ is $k2^{k+1} + |\Pi|$. For $\Pi = \boldsymbol{\alpha}_\tau$, the size is $k2^{k+1} + 2^k - 1$. We see that the classes with at most one type missing are tied for the largest formula size.

Using, e.g., standard probabilistic arguments, one can show that for Kripke models of size n, where n is much larger than 2^k, the largest equivalence class is the one realizing all the 1-types. In fact, the largest class will contain "almost all" of the models of size n.

▶ **Proposition 4.** *If n is large with respect to k, then $|M_\Pi| < |M_{\boldsymbol{\alpha}_\tau}|$, for every $\Pi \subset \boldsymbol{\alpha}_\tau$.*

On the other hand, we can show that the equivalence class containing models which realize all the 1-types is one of the most difficult ones to define. To prove this, we will start by introducing a formula size game for MLU[τ].

The formula size game for MLU[τ], denoted $\mathrm{FS}^\tau_{r_0}(\mathcal{A}_0, \mathcal{B}_0)$ has two players: Samson and Delilah. We refer to them as S and D, or he and she, respectively. The game has three parameters: a natural number $r_0 \geq 1$ and two sets of Kripke-models $\mathcal{A}_0$ and $\mathcal{B}_0$. Positions of the game are of the form $(r, \mathcal{A}, \mathcal{B})$ and the starting position is $(r_0, \mathcal{A}_0, \mathcal{B}_0)$.

In each position, S makes a move. The moves available for S in position $(r, \mathcal{A}, \mathcal{B})$ are:

- p-move: S chooses a τ-literal α. The game ends. If $\mathcal{A} \models \alpha$ and $\mathcal{B} \models \neg\alpha$, then S wins. Otherwise D wins. S cannot make this move if he has not made a $\blacklozenge$-move so far.
- $\vee$-move: S chooses $\mathcal{A}_1, \mathcal{A}_2 \subseteq \mathcal{A}$ such that $\mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$ and $r_1, r_2 \geq 1$ such that $r_1 + r_2 + 1 = r$. D chooses whether the next position is $(r_1, \mathcal{A}_1, \mathcal{B})$ or $(r_2, \mathcal{A}_2, \mathcal{B})$.
- $\wedge$-move: The same as a $\vee$-move with the roles of $\mathcal{A}$ and $\mathcal{B}$ switched.
- $\blacklozenge$-move: For every $(\mathfrak{M}, w) \in \mathcal{A}$, S chooses $v \in W$. Let $\mathcal{A}'$ be the set of models $(\mathfrak{M}, v)$ chosen this way. Let $\mathcal{B}' := \{(\mathfrak{M}, v) \mid (\mathfrak{M}, w) \in \mathcal{B} \text{ for some } w \in W, v \in W\}$. The next position of the game is $(r-1, \mathcal{A}', \mathcal{B}')$. S cannot make this move if $r = 1$.
- $\blacksquare$-move: The same as a $\blacklozenge$-move with the roles of $\mathcal{A}$ and $\mathcal{B}$ switched.

▶ **Theorem 5.** *The following statements are equivalent:*

1. *S has a winning strategy in the game $\mathrm{FS}^\tau_r(\mathcal{A}, \mathcal{B})$.*
2. *There is $\varphi \in \mathrm{MLU}[\tau]$ with size at most r such that $\mathcal{A} \models \varphi$ and $\mathcal{B} \models \neg\varphi$.*

Proof. Simple proof by induction. A version for basic modal logic can be found in [7]. ◀

Suppose that $\pi_1, \ldots, \pi_n$, where $n = 2^{|\tau|}$, enumerates all the 1-types over τ. Let $\mathfrak{M}_0$ denote a Kripke model with domain $\{1, \ldots, n\}$ and with the property that for every $1 \leq i \leq n$ the 1-type realized by i is π_i. For every $i \neq j$, we let $\mathfrak{M}_{i,j}$ denote the Kripke model obtained from $\mathfrak{M}_0$ by specifying that the 1-type of i is π_j. We further denote $\mathfrak{M}_i := \mathfrak{M}_{i,1}$ for $2 \leq i \leq n$ and $\mathfrak{M}_1 := \mathfrak{M}_{1,2}$. Each model $\mathfrak{M}_i$ is now missing the type π_i and is otherwise identical to $\mathfrak{M}_0$. We let $\mathcal{A}_0 = \{(\mathfrak{M}_0, 1)\}$ and $\mathcal{B}_0 = \{(\mathfrak{M}_i, 1) \mid 1 \leq i \leq n\}$. We will next show that separating these two sets requires a large MLU[τ] formula.

▶ **Lemma 6.** *D has a winning strategy in the game $\mathrm{FS}^\tau_{k2^{k+1}+2^k-2}(\mathcal{A}_0, \mathcal{B}_0)$.*

Proof. We use the following notation for the set of different underlying models that occur in a set X of pointed models: $\mathrm{Md}(X) = \{\mathfrak{M} \mid (\mathfrak{M}, i) \in \mathcal{A} \text{ for some } i\}$.

We define a measure for a position of the game called hardness. We use hardness as an invariant to show that a position is too hard for S to handle with the available resource r. Let π_i be a type and let $P = (r, \mathcal{A}, \mathcal{B})$ be a position of the game. We define four different kinds of types and the hardness of those types as follows:

1. If no $\blacklozenge$-moves have been made in the game so far, $\mathcal{A} \neq \emptyset$ and $\mathfrak{M}_i \in \mathrm{Md}(\mathcal{B})$, then π_i is of kind 1 and $h_i(P) = 2k$.
2. Otherwise, if there are propositionally equivalent $(\mathfrak{M}_0, j) \in \mathcal{A}$ and $(\mathfrak{M}_i, l) \in \mathcal{B}$, then π_i is of kind 2 and $h_i(P) = 2k$.

3. Otherwise, if $(\mathfrak{M}_0, i) \in \mathcal{A}$ and $\mathfrak{M}_i \in \mathrm{Md}(\mathcal{B})$, then π_i is of kind 3 and

$$h_i(P) = 2 \cdot |\{(\mathfrak{M}_i, j) \in \mathcal{B} \mid j \neq i, \pi_i \text{ and } \pi_j \text{ differ by exactly one proposition}\}| - 1.$$

4. Otherwise, π_i is of kind 4 and $h_i(P) = 0$.

We further denote the number of types with positive hardness by $\#h^+(P)$ and define the hardness $h(P)$ of the position P as $h(P) = \sum_{1 \leq i \leq n} h_i(P) + \#h^+(P) - 1$.

We will describe the winning strategy for D in terms of maintaining the following two conditions in each position P of the game:

(a) $r < h(P)$,

(b) there is at most one type of kind 3 in position P.

We will show that while these conditions hold, S cannot win. Since the resource r of S will run out eventually, this is a winning strategy for D.

In the starting position P_0 no ♦-moves have been made and $\mathfrak{M}_i \in \mathrm{Md}(\mathcal{B}_0)$ for each $1 \leq i \leq n$ so all types π_i are of kind 1 and have $h_i(P_0) = 2k$. Thus condition (b) holds and

$$r = k2^{k+1} + 2^k - 2 < k2^{k+1} + 2^k - 1 = 2k2^k + 2^k - 1 = h(P_0)$$

p-move: In each position P of the game, we have $r \geq 1$ so while $r < h(P)$ holds, we have $h(P) \geq 2$. Using this, we show that any p-move made by S while $r < h(P)$ leads to a win for D. If no ♦-moves have been made, then S cannot make a p-move. If there is a type π_i of kind 2, then there are propositionally equivalent $(\mathfrak{M}_0, j) \in \mathcal{A}$ and $(\mathfrak{M}_i, l) \in \mathcal{B}$ so no literal separates them. If neither of the above hold, then by condition (b), there is a type π_i of kind 3 with $(\mathfrak{M}_0, i) \in \mathcal{A}$ and $(\mathfrak{M}_i, j), (\mathfrak{M}_i, l) \in \mathcal{B}$, where π_j and π_l differ from π_i by exactly one proposition. Again no literal separates $\mathcal{A}$ and $\mathcal{B}$.

$\vee$-move: Similar to the $\wedge$-move case below. Full details in the Appendix.

$\wedge$-move: We show that one of the positions P_1, P_2 satisfies the conditions (a) and (b).

Let $\mathcal{B}_1, \mathcal{B}_2 \subseteq \mathcal{B}$ and $r_1, r_2 \geq 1$ be the choices of S. Let π_i be a type. If π_i is of kind 1, then $\mathfrak{M}_i \in \mathrm{Md}(\mathcal{B}_1)$ or $\mathfrak{M}_i \in \mathrm{Md}(\mathcal{B}_2)$ so π_i is still of kind 1 and $h_i(P_1) = 2k$ or $h_i(P_2) = 2k$. If π_i is of kind 2 with propositionally equivalent models $(\mathfrak{M}_0, j) \in \mathcal{A}$ and $(\mathfrak{M}_i, l) \in \mathcal{B}$, then $(\mathfrak{M}_i, l) \in \mathcal{B}_1$ or $(\mathfrak{M}_i, l) \in \mathcal{B}_2$ so π_i is still of kind 2 and $h_i(P_1) = 2k$ or $h_i(P_2) = 2k$.

Finally if π_i is a type of kind 3, then S can split the models $(\mathfrak{M}_i, j) \in \mathcal{B}$, where π_i and π_j differ by one proposition, between the sets $\mathcal{B}_1$ and $\mathcal{B}_2$.

Assume that S puts all these models on the same side. Then $h_i(P_1) = h_i(P)$ or $h_i(P_2) = h_i(P)$. Thus $h_i(P_1) + h_i(P_2) \geq h_i(P)$. Additionally $\#h^+(P_1) + \#h^+(P_2) \geq \#h^+(P)$ so

$$\begin{aligned} h(P_1) + h(P_2) &= \sum_{1 \leq i \leq n} h_i(P_1) + \sum_{1 \leq i \leq n} h_i(P_2) + \#h^+(P_1) + \#h^+(P_2) - 2 \\ &\geq \sum_{1 \leq i \leq n} h_i(P) + \#h^+(P) - 1 - 1 = h(P) - 1. \end{aligned}$$

Now $r_1 + r_2 = r - 1 < h(P) - 1 \leq h(P_1) + h(P_2)$ so we have $r_1 < h(P_1)$ or $r_2 < h(P_2)$. Now assume that S splits some models $(\mathfrak{M}_i, j)$ to both sides. Now $h_i(P_1) + h_i(P_2) \geq h_i(P) - 1$. In addition, the type π_i has positive hardness in both positions P_1 and P_2 so $\#h^+(P_1) + \#h^+(P_2) \geq \#h^+(P) + 1$. These two deviations from the above case, that only concern the single type π_i of kind 3, cancel each other out so again $h(P_1) + h(P_2) \geq h(P) - 1$ and therefore $r_1 < h(P_1)$ or $r_2 < h(P_2)$. Finally, all types are of the same kind as in position P so condition (b) holds.

♦-move: Let $(\mathfrak{M}_0, i)$ be a choice of S. For each $j \neq i$ with $\mathfrak{M}_j \in \mathrm{Md}(\mathcal{B})$, we have $(\mathfrak{M}_j, i) \in \mathcal{B}'$ so π_j is a type of kind 2 and $H_j(P') = 2k$. If there are multiple versions of $\mathfrak{M}_0$ in $\mathcal{A}$ and S makes another choice $(\mathfrak{M}_0, l)$, then all types π_j with $\mathfrak{M}_j \in \mathrm{Md}(\mathcal{B})$ work the same

way. If S only chooses $(\mathfrak{M}_0, i)$ and we have $\mathfrak{M}_i \in \mathrm{Md}(\mathcal{B})$, then π_i becomes a type of kind 3 with $(\mathfrak{M}_i, j) \in \mathcal{B}'$ for all $j \neq i$ so $h_i(P') = 2k - 1$. We additionally note that by the definition of hardness, $h(P) \leq 2k \cdot |\mathrm{Md}(\mathcal{B})| + |\mathrm{Md}(\mathcal{B})| - 1$. Thus

$$h(P') \geq 2k \cdot |\mathrm{Md}(\mathcal{B})| - 1 + |\mathrm{Md}(\mathcal{B})| - 1 \geq h(P) - 1 > r - 1 = r'$$

and condition (b) is maintained.

■**-move:** For each $\mathfrak{M}_i \in \mathrm{Md}(\mathcal{B})$, S chooses at least one $(\mathfrak{M}_i, l) \in \mathcal{B}'$. Let π_j be the type this model realizes. Now $(\mathfrak{M}_0, j) \in \mathcal{A}'$ realizes the same type, π_i is of kind 2 and $h_i(P') = 2k$. Thus $h(P') = 2k \cdot |\mathrm{Md}(\mathcal{B})| + |\mathrm{Md}(\mathcal{B})| - 1 \geq h(P) > r - 1 = r'$ and condition (b) holds. ◀

We have shown that the largest class $M_{\boldsymbol{\alpha}_\tau}$ requires a formula of size at least $k2^{k+1} + 2^k - 1$ to define. Since any of the classes can be defined via a formula of *precisely* this size, we see that in the case of MLU[τ] the largest class is maximally difficult to define.

▶ **Proposition 7.** *The largest equivalence class $M_{\boldsymbol{\alpha}_\tau}$ of $\equiv_{\mathrm{MLU}[\tau]}$ has maximal* MLU[τ]*-description complexity.*

4 GMLU: Relating entropy and description complexity asymptotically

Fix $\tau = \{p_1, \ldots, p_k\}$ and let $\ell = 2^k$. A Kripke model $\mathfrak{M}$ with universe $W = \{1, \ldots, n\}$ can be described in GMLU[τ] up to isomorphism. Hence the equivalence classes of $\equiv_{\mathrm{GMLU}[\tau]}$, hereafter denoted $\equiv$, over $\mathrm{Mod}_n(\tau)$ are the isomorphism classes. Since $\mathfrak{M}$ can be described up to isomorphism by listing how many times each 1-type is realized, there is a one-to-one correspondence between isomorphism classes and tuples $(n_1, \ldots, n_\ell)$, where $n_1 + \cdots + n_\ell = n$. We will use $[n_1, \ldots, n_\ell]$ to denote the isomorphism class consisting of those Kripke models of size n in which the ith type is realized precisely n_i-times. Note that $|[n_1, \ldots, n_\ell]| = \binom{n}{n_1, \ldots, n_\ell}$.

In this section we show that the expected Boltzmann entropy $\langle H_B \rangle$ is asymptotically $|\tau|$ times the expected GMLU[τ]-description complexity with respect to the distribution $p_\equiv$.

4.1 Expected Boltzmann entropy

In this subsection we will establish that $\langle H_B \rangle \sim |\tau| n$. Using Proposition 1 we get the following alternative asymptotic formula for $\langle H_B \rangle$

$$\begin{aligned}
&\sum_{n_1+\cdots+n_\ell=n} p_\equiv([n_1, \ldots, n_\ell]) \log \binom{n}{n_1, \ldots, n_\ell} \\
&= \sum_{n_1+\cdots+n_\ell=n} p_\equiv([n_1, \ldots, n_\ell]) \left(n \left(\log(n) - \sum_{i=1}^{\ell} \frac{n_i}{n} \log(n_i) \right) + \Theta\big(\log(n)\big) + \sum_{i=1}^{\ell} \Theta(\log(n_i)) \right) \\
&= \left(\sum_{n_1+\cdots+n_\ell=n} p_\equiv([n_1, \ldots, n_\ell]) \left(\sum_{i=1}^{\ell} \frac{n_i}{n} \log\left(\frac{n}{n_i}\right) \right) \right) n \\
&\quad + \Theta\big(\log(n)\big) + \sum_{n_1+\cdots+n_\ell=n} p_\equiv([n_1, \ldots, n_\ell]) \sum_{i=1}^{\ell} \Theta(\log(n_i))
\end{aligned}$$

We will show that

$$\left(\sum_{n_1+\cdots+n_\ell=n} p_\equiv([n_1, \ldots, n_\ell]) \left(\sum_{i=1}^{\ell} \frac{n_i}{n} \log\left(\frac{n}{n_i}\right) \right) \right) n \tag{2}$$

is asymptotically $|\tau|n$, which will of course entail that $\langle H_B \rangle \sim |\tau|n$. Note that

$$\sum_{i=1}^{\ell} \frac{n_i}{n} \log\left(\frac{n}{n_i}\right) \tag{3}$$

is the Shannon entropy of the distribution on $\{1, \ldots, \ell\}$ which assigns to each $1 \leq i \leq \ell$ the weight $\frac{n_i}{n}$. Thus we can use $\log(\ell) = |\tau|$ to bound the formula

$$\sum_{n_1+\cdots+n_\ell=n} p_{\equiv}([n_1, \ldots, n_\ell]) \left(\sum_{i=1}^{\ell} \frac{n_i}{n} \log\left(\frac{n}{n_i}\right) \right) \tag{4}$$

from above. Hence $|\tau|n$ is an upper bound on (2).

We will next bound (4) from below by using Proposition 2. For every $1 \leq i \leq \ell$ and $j \in \mathbb{Z}_+$ we let X_j^i denote a random Bernoulli variable with success probability $2^{-|\tau|}$. Intuitively speaking, X_j^i is an indicator function for the event "the jth element received the ith 1-type". Now, for every $1 \leq i \leq \ell$ and for all $\delta > 0$ the law of large numbers implies that

$$\lim_{n\to\infty} \Pr\left[\left|\tfrac{1}{n}\textstyle\sum_{j=1}^{n} X_j^i - 2^{-|\tau|}\right| < \delta\right] = 1.$$

Thus it follows from the union bound that the following probability

$$\Pr\left[\forall\ 1 \leq i \leq \ell : \left|\frac{1}{n}\sum_{j=1}^{n} X_j^i - 2^{-|\tau|}\right| < \delta\right] \tag{5}$$

approaches 1 as $n \to \infty$. Fix $\delta > 0$. For every n we let I_n^δ denote the following set:

$$\left\{(n_1, \ldots, n_\ell) \mid n_1 + \cdots + n_\ell = n \text{ and } \forall\ 1 \leq i \leq \ell : \left|\frac{n_i}{n} - 2^{-|\tau|}\right| < \delta\right\}.$$

The set I_n^δ includes the tuples $(n_1, \ldots, n_\ell)$, where the numbers add up to n and are very close to each other. The probability result above intuitively means that a randomly chosen tuple is almost always in I_n^δ. Thus, roughly speaking, we only need to consider models, where the points are split between all of the types very evenly.

Let n be large enough so that (5) is larger than $(1-\delta)$. For every $(n_1, \ldots, n_\ell) \in I_n^\delta$ we want to estimate the formula (3) from below. Fix a tuple $(n_1, \ldots, n_\ell) \in I_n^\delta$. Now, for every $1 \leq i \leq \ell$ we have $2^{-|\tau|} - \delta < n_i/n < 2^{-|\tau|} + \delta$, which also entails that $n/n_i > 2^{|\tau|}/(1 + \delta 2^{|\tau|})$. Thus for every $1 \leq i \leq \ell$ we have that

$$2^{|\tau|}(2^{-|\tau|} - \delta) \log\left(\frac{2^{|\tau|}}{(1+\delta 2^{|\tau|})}\right) < \sum_{i=1}^{\ell} \frac{n_i}{n} \log\left(\frac{n}{n_i}\right).$$

Now we can bound the formula (4) from below by

$$2^{|\tau|}(2^{-|\tau|} - \delta) \log\left(\frac{2^{|\tau|}}{(1+\delta 2^{|\tau|})}\right) \cdot \sum_{(n_1,\ldots,n_\ell)\in I_n^\delta} p_{\equiv}([n_1, \ldots, n_\ell]).$$

Notice that the right-hand side expresses the probability that a random τ-model $\mathfrak{A}$ of size n belongs to $[n_1, \ldots, n_\ell]$, for some $(n_1, \ldots, n_\ell) \in I_n^\delta$, which we know is at least $(1-\delta)$, since we chose n to be large enough. Thus we have, for every $\delta > 0$ and n sufficiently large, the following lower bound for the formula (4):

$$f(\delta) := 2^{|\tau|}(2^{-|\tau|} - \delta) \log\left(\frac{2^{|\tau|}}{(1+\delta 2^{|\tau|})}\right) \cdot (1-\delta).$$

Observe that $f(\delta) \to |\tau|$ as $\delta \to 0$. Hence, for every $\varepsilon > 0$ we have that $(1-\varepsilon)|\tau| < f(\delta)$, for sufficiently small δ. Combining this with our upper bound of $|\tau|n$ for (2) one can easily show that (2) is asymptotically $|\tau|n$. This concludes our proof of the following theorem.

▶ **Theorem 8.** $\langle H_B \rangle \sim |\tau|n$.

4.2 Expected description complexity

In this subsection we show that $\langle C \rangle \sim n$. Let M be an equivalence class of $\equiv$. For a 1-type π we denote $|\pi|_M := |\{w \in W \mid (\mathfrak{M}, w) \vDash \pi\}|$, where $\mathfrak{M} \in M$. The number $|\pi|_M$ is the number of points that satisfy the type π in the models of the class M. Since we will focus on a single class M we will omit the subscript in the sequel. Let π_m be the 1-type with the largest number of points in the models of the class M. Let $I := \{1 \leq i \leq 2^{|\tau|} \mid |\pi| \geq 1\}$. The set I consists of the indices of types that are realized in the class M. In this subsection we show that the formula size required to define such a class M is in the order of $\min(n, 2(n - |\pi_m|))$.

For upper bounds, we define a class M via two different formulas, one of them using the largest type π_m defined above:

$$\varphi_1 := \bigwedge_{i \in I} \blacklozenge^{\geq |\pi_i|} \psi(\pi_i)$$

$$\varphi_2 := \blacksquare^{<1}\Big(\bigvee_{i \in I} \psi(\pi_i)\Big) \wedge \bigwedge_{i \in I \setminus \{m\}} \blacklozenge^{\geq |\pi_i|} \psi(\pi_i) \wedge \bigwedge_{i \in I \setminus \{m\}} \blacksquare^{<|\pi_i|+1} \neg\psi(\pi_i)$$

It is easy to verify that $size(\varphi_1) = n + \mathcal{O}_{|\tau|}(1)$ and $size(\varphi_2) = 2(n - |\pi_m|) + \mathcal{O}_{|\tau|}(1)$.

For the lower bounds, we utilize a formula size game $\mathrm{FSC}_r^\tau(\mathcal{A}, \mathcal{B})$ for GMLU$[\tau]$. The rules of the game are the same as in the MLU$[\tau]$-game except the $\blacklozenge$-moves and $\blacksquare$-moves are replaced with the following new moves:

- $\blacklozenge^{\geq d}$-move: S chooses a number $d \in \mathbb{N}$. If $r \leq d$, the game ends and D wins. Otherwise, for every $(\mathfrak{M}, w) \in \mathcal{A}$, S chooses d different points $v \in W$. Let $\mathcal{A}'$ be the set of models $(\mathfrak{M}, v)$ chosen this way. For every $(\mathfrak{M}, w) \in \mathcal{B}$, S chooses $n - d + 1$ different points $v \in W$. Let $\mathcal{B}'$ again be the set of models chosen. The next position of the game is $(r - d, \mathcal{A}', \mathcal{B}')$.
- $\blacksquare^{<d}$-move: The same as a $\blacklozenge^{\geq d}$-move with the roles of $\mathcal{A}$ and $\mathcal{B}$ switched.

The equivalent of Theorem 5 can be proved for this new game in a very similar manner.

For an example of using these new moves, consider a model $\mathfrak{M} \in \mathcal{A}$ with three points, where p is true and another model $\mathfrak{M}' \in \mathcal{B}$, with only two points, where p is true. To separate these models S might make a $\blacklozenge^{\geq d}$-move choosing $d = 3$. S would choose the three points in $\mathfrak{M}$ with p and the $n - 2$ points in $\mathfrak{M}'$ with $\neg p$. Now all three versions of $\mathfrak{M} \in \mathcal{A}'$ have p while all $n - 2$ versions of $\mathfrak{M}' \in \mathcal{B}'$ have $\neg p$ so S wins by making a p-move.

We now define the starting model sets of our formula size game. As before, we assume the domain of the models is $W = \{1, \ldots, n\}$. Let $\mathcal{A}_0 := \{(\mathfrak{M}, 1)\}$, where $\mathfrak{M} \in M$. We additionally assume that the points 1 and 2 of the model are propositionally equivalent. We do not need to fix the model $\mathfrak{M}$ any more precisely but note that there is only one model in the set $\mathcal{A}_0$. Now let $(i, j) \in I \times I$ with $i \neq j$ and let $w \in W$ be the largest number with $(\mathfrak{M}, w) \vDash \pi_i$. The model $\mathfrak{M}_{i \to j}$ has $(\mathfrak{M}_{i \to j}, w) \vDash \pi_j$ and is otherwise identical to $\mathfrak{M}$. In other words, $\mathfrak{M}_{i \to j}$ has one less point of the type π_i and one more of the type π_j compared to $\mathfrak{M}$. We let $\mathcal{B}_0 := \{(\mathfrak{M}_{i \to j}, 1) \mid i, j \in I, i \neq j\}$. There are $|I|^2$ models in the set $\mathcal{B}_0$. Note that all models in $\mathcal{A}_0$ and $\mathcal{B}_0$ have propositionally equivalent starting points.

Let us now consider the formula size game $\mathrm{FSC}^{\tau}_{r_0}(\mathcal{A}_0, \mathcal{B}_0)$. For any position $(r, \mathcal{A}, \mathcal{B})$ of this game, we define a directed graph $\mathcal{G}(\mathcal{A}, \mathcal{B}) := (V, E)$ by setting $V := I$ and $(i, j) \in E$ iff there are propositionally equivalent $(\mathfrak{M}, w) \in \mathcal{A}$ and $(\mathfrak{M}_{i\to j}, v) \in \mathcal{B}$. We call a set $C \subseteq \{i^+, i^- \mid i \in I\}$ a **cover** of $\mathcal{G}(\mathcal{A}, \mathcal{B})$ if for every $(i, j) \in E$ we have $i^+ \in C$ or $j^- \in C$. The cost of a cover C is

$$r(C) := \sum_{i^+ \in C} |\pi_i|_M + \sum_{i^- \in C} |\pi_i|_M.$$

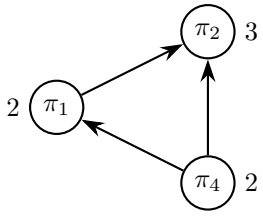


Figure 1 The graph $\mathcal{G}(\mathcal{A}, \mathcal{B})$ and the number of points for each type.

We give an example of a graph $\mathcal{G}(\mathcal{A}, \mathcal{B})$ and a cover for this graph. Consider the alphabet $\tau = \{p, q\}$ and the class M of models of size 7, where the type $\pi_1 = \{p, q\}$ is realized in two points, the type $\pi_2 = \{\neg p, q\}$ in three points and the type $\pi_4 = \{\neg p, \neg q\}$ in two points. The type $\pi_3 = \{p, \neg q\}$ is not realized in models of M. We assume that $(\mathfrak{M}, w) \in \mathcal{A}$ for some $\mathfrak{M} \in M$ and we have propositionally equivalent $(\mathfrak{M}_{i\to j}, v) \in \mathcal{B}$ for the pairs $(1, 2)$, $(4, 2)$ and $(4, 1)$. The graph $\mathcal{G}(\mathcal{A}, \mathcal{B})$ is pictured below. The set $C = \{2^-, 4^+\}$ is a cover of $\mathcal{G}(\mathcal{A}, \mathcal{B})$. To see this, note that the inclusion of 4^+ covers all edges from π_4 to other nodes and 2^- covers edges from other nodes to π_2. This covers all edges so C is a cover. The cost of C is $|\pi_2| + |\pi_4| = 3 + 2 = 5$.

We are now ready for the crucial Lemma of this subsection.

▶ **Lemma 9.** *Let $P := (r, \mathcal{A}, \mathcal{B})$ be a position of the game $\mathrm{FSC}^{\tau}_{r_0}(\mathcal{A}_0, \mathcal{B}_0)$ and let $R(P) := \min\{r(C) \mid C \text{ is a cover of } \mathcal{G}(\mathcal{A}, \mathcal{B})\}$. If $r < R(P)$, then D has a winning strategy in the game from the position P.*

Proof. We show that any move S makes either leads to D winning the game immediately or maintains the conditions of the claim given the correct choice by D.

***p*-move:** Since $R(P) > 0$, there are propositionally equivalent pointed models on both sides of the game so clearly D wins if S makes any p-move.

$\vee$-move: Let $\mathcal{A}_1, \mathcal{A}_2 \subseteq \mathcal{A}$ and $r_1, r_2 \geq 1$ be the choices of S and let $P_1 = (r_1, \mathcal{A}_1, \mathcal{B})$ and $P_2 = (r_2, \mathcal{A}_2, \mathcal{B})$. For each edge $e = (i, j) \in E$, there are propositionally equivalent models $(\mathfrak{M}, w) \in \mathcal{A}$ and $(\mathfrak{M}_{i\to j}, v) \in \mathcal{B}$. Since $\mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$, every model $(\mathfrak{M}, w) \in \mathcal{A}$ is in $\mathcal{A}_1$ or $\mathcal{A}_2$ so every edge of the graph $\mathcal{G}(\mathcal{A}, \mathcal{B})$ is present in at least one of the graphs $\mathcal{G}(\mathcal{A}_1, \mathcal{B})$ and $\mathcal{G}(\mathcal{A}_2, \mathcal{B})$. We claim that $r_1 < R(P_1)$ or $r_2 < R(P_2)$. Assume for contradiction that $r_1 \geq R(P_1)$ and $r_2 \geq R(P_2)$. Then there is a cover C_1 of $\mathcal{G}(\mathcal{A}_1, \mathcal{B})$ with $r(C_1) \leq r_1$ and the same for P_2. Now $C_1 \cup C_2$ is a cover of $\mathcal{G}(\mathcal{A}, \mathcal{B})$. Additionally $r(C_1 \cup C_2) \leq r(C_1) + r(C_2) \leq r_1 + r_2 \leq r$. This means that $R(P) \leq r$, which is a contradiction with the condition $r < R(P)$. Thus D can choose a position that maintains the condition of the claim.

$\wedge$-move: Very similar to the above case with the models in $\mathcal{B}$ split between $\mathcal{B}_1$ and $\mathcal{B}_2$.

$\Diamond^{\geq d}$-move: Let $d \in \mathbb{N}$ be the number chosen by S. For each $(\mathfrak{M}, w) \in \mathcal{A}$, S chooses d different points from the model $\mathfrak{M}$. Let A be the set of all points chosen this way. For each $(\mathfrak{M}_{i\to j}, w) \in \mathcal{B}$, let $B_{i\to j}$ be the set of $n - d + 1$ points chosen by S. Let $\mathrm{tp}_{\mathfrak{M}}(X)$ be the set of types realized by a set X of points in the model $\mathfrak{M}$. We consider the following two cases:

1. The model $\mathfrak{M}$ has at least $d+1$ points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Let $e=(i,j)\in E$. The model $\mathfrak{M}_{i\to j}$ only differs from $\mathfrak{M}$ by the type of one point so $\mathfrak{M}_{i\to j}$ has at least d points that realize types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Since $|B_{i\to j}|=n-d+1$, there is at least one point in $B_{i\to j}$ with a type from $\mathrm{tp}_{\mathfrak{M}}(A)$. Thus there are propositionally equivalent $(\mathfrak{M},w')\in\mathcal{A}'$ and $(\mathfrak{M}_{i\to j},v')\in\mathcal{B}'$. Thus the edge $e=(i,j)$ is still present in the graph $\mathcal{G}(\mathcal{A}',\mathcal{B}')$ of the following position. This applies for every $e\in E$ so $R(P')=R(P)$.
2. The model $\mathfrak{M}$ has exactly d points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Now A is the set of those d points. We first consider edges $e=(i,j)\in E$ with $\pi_i\notin\mathrm{tp}_{\mathfrak{M}}(A)$ or $\pi_j\in\mathrm{tp}_{\mathfrak{M}}(A)$. For any edge of this kind, the model $\mathfrak{M}_{i\to j}$ has at least d points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$ so at least one of the $n-d+1$ points in $B_{i\to j}$ has a type from $\mathrm{tp}_{\mathfrak{M}}(A)$. As in case 1, this means that all these edges are still present in the graph $\mathcal{G}(\mathcal{A}',\mathcal{B}')$.
   Let us then consider the rest of the edges $e=(i,j)\in E$ with $\pi_i\in\mathrm{tp}_{\mathfrak{M}}(A)$ and $\pi_j\notin\mathrm{tp}_{\mathfrak{M}}(A)$. For an edge of this kind, the model $\mathfrak{M}_{i\to j}$ has only $d-1$ points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Thus if S chooses the $n-d+1$ points of $B_{i\to j}$ to be exactly the points with types not in $\mathrm{tp}_{\mathfrak{M}}(A)$, then $\mathfrak{M}_{i\to j}$ has no propositionally equivalent counterpart on the other side and the edge e is not present in the graph $\mathcal{G}(\mathcal{A}',\mathcal{B}')$.
   We then consider the condition of the claim in the position $P'=(r-d,\mathcal{A}',\mathcal{B}')$. By the above arguments, the only way S could remove edges when moving from $\mathcal{G}(\mathcal{A},\mathcal{B})$ to $\mathcal{G}(\mathcal{A}',\mathcal{B}')$, was to choose in each version of the model $\mathfrak{M}$ exactly all of the points that satisfy some set $\mathrm{tp}_{\mathfrak{M}}(A)$ of types. Any edge eliminated this way originates from an index i of a type in $\mathrm{tp}_{\mathfrak{M}}(A)$. All of these edges can be covered via the cover $C_A=\{i^+\mid\pi_i\in\mathrm{tp}_{\mathfrak{M}}(A)\}$. The cost of this cover is the total number of points of the model $\mathfrak{M}$ with types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Since A contains exactly all points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$, we have $r(C_A)=|A|=d$. Let C' be a cover of $\mathcal{G}(\mathcal{A}',\mathcal{B}')$ with minimal cost so $R(P')=r(C')$. Now $C'\cup C_A$ is a cover of $\mathcal{G}(\mathcal{A},\mathcal{B})$ with cost $R(P')+d$. Thus $r<R(P)\leq R(P')+d$ so $r-d<R(P')$ and the condition of the claim is maintained.

■$^{<d}$-move: Similar to the **♦$^{\geq d}$**-move with $n-d+1$ points chosen from models in $\mathcal{A}$ and d points chosen from models in $\mathcal{B}$. Full details in the Appendix. ◀

By the above Lemma, the formula size required to define a class M of the equivalence $\equiv$ comes down to calculating the minimum cost of a cover.

▶ **Theorem 10.** *Let M be an equivalence class of the relation $\equiv$ and let π be the propositional type with most satisfying points in models in M. If the formula $\varphi\in\mathrm{GMLU}[\tau]$ defines the class M, then φ has size at least* $\min(n,2(n-|\pi|))$.

Proof. Let $s=\min(n,2(n-|\pi|))$. We use the above Lemma to show that D has a winning strategy in the game $\mathrm{FSC}^{\tau}_{s}(\mathcal{A}_0,\mathcal{B}_0)$, thus proving the claim.

It suffices to show that the minimum cost of a cover of $\mathcal{G}(\mathcal{A}_0,\mathcal{B}_0)=(V,E)$ is equal to s. First we see that $\mathcal{G}(\mathcal{A}_0,\mathcal{B}_0)$ is a complete irreflexive directed graph. We begin by noting that $C^+:=\{i^+\mid i\in I\}$ is a cover with cost n and adding any i^- or replacing i^+ with i^- does not reduce the cost. Thus if all indices are used, C^+ is a minimum cost cover. Next, we consider covers C_i, where there is an index $i\in I$ with $\{i^+,i^-\}\cap C_i=\emptyset$. Note that i is the only such index. Indeed, if there were a second such index j, then the edge (i,j) would not be covered. Now, for any $j\in I$, $j\neq i$ we have $j^+\in C_i$ since it is the only way to cover the edge (j,i). In the same way $j^-\in C_i$ since the edge (i,j) must be covered. Thus $C_i=\{j^+,j^-\mid j\in I,j\neq i\}$. The cost of C_i is

$$r(C_i)=\sum_{j^+\in C_i}|\pi_j|+\sum_{j^-\in C_i}|\pi_j|=n-|\pi_i|+n-|\pi_i|=2(n-|\pi_i|).$$

The cost minimal cover of this type is clearly the one where i is the index of the type with the most satisfying points. Thus the minimal cover size is $\min(n,2(n-|\pi|))$. ◀

▶ **Theorem 11.** $\langle C \rangle \sim n$.

Proof. Since $C(M) \leq n + \mathcal{O}_{|\tau|}(1)$ for any equivalence class M, we have $\langle C \rangle \leq n + \mathcal{O}_{|\tau|}(1)$. For the lower bound, recall from the previous section that for any $\delta > 0$ and n sufficiently large we have that

$$\sum_{(n_1,\ldots,n_\ell) \in I_n^\delta} p_{\equiv}([n_1, \ldots, n_\ell]) > (1 - \delta).$$

Observe that if $(n_1, \ldots, n_\ell) \in I_n^\delta$, for δ sufficiently small, then Theorem 10 entails that $C([n_1, \ldots, n_\ell]) \geq n$ as every 1-type is realized less than $n/2$-times. Thus, for any $\delta > 0$ and n sufficiently large, we have $\langle C \rangle \geq (1-\delta)n$. Using these bounds it is easy to show $\langle C \rangle \sim n$. ◀

The desired relation between Boltzmann entropy and description complexity now follows directly from Theorems 8 and 11.

▶ **Corollary 12.** $\langle H_B \rangle \sim |\tau| \langle C \rangle$

5 FO: Expected description complexity for polyadic vocabularies

We saw in the previous section that the ratio of expected Boltzmann entropy of an isomorphism class and its GMLU-description complexity is asymptotically the size of the underlying fixed vocabulary. Given that the main characteristic of GMLU is that it can characterize finite monadic structures up to isomorphism, one might guess that a similar behaviour would extend to FO, which can characterize arbitrary finite structures up to isomorphism. The purpose of this section is to show that surprisingly this is not the case: the expected description complexity grows faster than the expected Boltzmann entropy.

Given a relation symbol R we will use $ar(R)$ to denote its arity. Fix a finite relational vocabulary τ and let $m := \max\{ar(R) \mid R \in \tau\}$. For the rest of this section we will assume that $m \geq 2$. The following result, which fails for unary vocabularies, is established in [4].

▶ **Proposition 13.** *The number of non-isomorphic τ-models of size n is asymptotically $2^{p(n)}/n!$, where $p(n) = \sum_{R \in \tau} n^{ar(R)}$.*

In [12] the authors mention (without a proof) that with high probability, defining a single graph of size n up to isomorphism in FO requires a sentence of size $\Omega\big(\frac{n^2}{\log(n)}\big)$. Here we prove a version of this statement for an arbitrary (but finite) relational vocabulary. For the proof, recall that $\equiv_{\mathrm{FO}[\tau]}$ is over $\mathrm{Mod}_n(\tau)$.

▶ **Theorem 14.** *With high probability we have that $C_{\mathrm{FO}[\tau]}(M) = \Omega\big(\frac{n^m}{\log(n)}\big)$, when the isomorphism class M is selected uniformly at random.*

Proof. The proof is a counting argument: we will show that the ratio between "short" formulas and isomorphism classes of models of size n approaches 0 as n increases. Fix n and consider some $s \geq 2$. We will start by bounding the number of FO[τ]-sentences of size s in which w.l.o.g. only variables from the set $\{x_1, \ldots, x_n\}$, which consists of pairwise distinct variables, occur.

Each FO[τ]-sentence of size s can be viewed as a labeled tree with s nodes, the labels being literals and symbols from the set $\{\wedge, \vee, \exists, \forall\}$. Since a tree with s nodes has $s-1$ edges, the syntax tree of each FO[τ]-sentence of size s can be encoded using

$$f(s) := s + 1 + (s-1)2\log(s) + s\log(N_\tau + 4)$$

bits, where $N_\tau := \sum_{R\in\tau} n^{ar(R)}$ is the number of atomic τ-formulas over $\{x_1,\dots,x_n\}$. Thus there are at most $2^{f(s)}$ sentences of size s. Using this bound we can also bound the number of FO[τ]-sentences of size *at most* s. Indeed, the number of such sentences is at most $\sum_{i=2}^{s} 2^{f(i)} \leq s2^{f(s)} = 2^{f(s)+\log(s)}$.

Observe that $\log(N_\tau+4) \leq d\log(n)$, for some $d>0$ (and n sufficiently large). We now set

$$s = \frac{n^m}{10(m+d)\log(n)},$$

which implies (using very crude bounds such as $s \leq s\log(s)$) that

$$\begin{aligned} f(s)+\log(s) &\leq 5s\log(s) + s\log(N_\tau+4) \\ &\leq 5s(\log(s)+d\log(n)) \\ &\leq 5(m+d)s\log(n) = \frac{n^m}{2} \end{aligned}$$

Thus there are at most $2^{n^m/2}$ sentences of size at most s.

Now the number of non-isomorphic τ-models of size n is asymptotically

$$2^{p(n)}/n! \geq 2^{n^m}/n! \geq 2^{n^m - n\log(n)} = 2^{\left(1-\frac{\log(n)}{n^{m-1}}\right)n^m}$$

for sufficiently large n. Combining these two estimates we have that

$$\frac{2^{n^m/2}}{2^{p(n)}/n!} \leq \frac{2^{n^m/2}}{2^{\left(1-\frac{\log(n)}{n^{m-1}}\right)n^m}} = \left(2^{\frac{\log(n)}{n^{m-1}}-\frac{1}{2}}\right)^{n^m}$$

Since $\log(n)/n^{m-1} \to 0$, by taking n to be sufficiently large we have that $2^{\frac{\log(n)}{n^{m-1}}-\frac{1}{2}} < 1$. Thus with high probability we have that $C_{\mathrm{FO}[\tau]}(M) = \Omega\left(\frac{n^m}{\log(n)}\right)$. ◀

▶ Remark 15. Since for every isomorphism class M we have that $C_{\mathrm{FO}[\tau]}(M) = \mathcal{O}(n^m)$, there is a small gap between this upper bound and the lower bound established in Theorem 14. Even in the case of graphs it seems an open problem to determine the average case FO-description complexity of an isomorphism class, see [12] for more discussion.

Consider now the partition $\equiv_{\mathrm{FO}[\tau]}$ of $\mathrm{Mod}_n(\tau)$. In Appendix A.5 we use Theorem 14 to establish the following result.

▶ **Proposition 16.** *The expected description complexity of* $\equiv_{\mathrm{FO}[\tau]}$ *grows asymptotically faster than its expected Boltzmann entropy.*

Note that Proposition 16 does not follow immediately from Theorem 14, since there we consider the uniform distribution over the isomorphism classes, while here we need to consider $p_{\equiv_{\mathrm{FO}[\tau]}}$ which a priori could place negligible probabilities on isomorphism classes with high description complexity. However, it follows from the results of [4] that for large n the distribution $p_{\equiv_{\mathrm{FO}[\tau]}}$ is quite close to the uniform distribution.

6 Conclusion

The current paper has demonstrated links between description complexity and entropy. Concretely, we have shown that in MLU, the largest class has maximal Boltzmann entropy, while for GMLU, the expected description complexity is asymptotically equivalent to expected Boltzmann entropy. Corresponding links to Shannon entropy follow from Proposition 3. We also contrast our findings by proving that for first-order logic, description complexity grows asymptotically faster than expected Boltzmann entropy.

In general, our results relate to links between Kolmogorov complexity and entropy, developed this time in the logic-based scenario where relational structures are classified by formulas of different sizes. The results demonstrate, for example, how the size of data classifiers relates to the randomness of the classified data.

In the future we shall expand this study to further logics and more general settings. The overall aim is to investigate the interplay of description complexity and the described classes, involving, e.g., suitable Galois connections between sizes of formulas and classes. Further connections to, e.g., statistical physics should also be investigated.

References

1. Micah Adler and Neil Immerman. An *n!* lower bound on formula size. *ACM Trans. Comput. Log.*, 4(3):296–314, 2003. doi:10.1145/772062.772064.
2. Pablo Barceló, Mikaël Monet, Jorge Pérez, and Bernardo Subercaseaux. Model interpretability through the lens of computational complexity. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/b1adda14824f50ef24ff1c05bb66faf3-Abstract.html.
3. Stephen J. Blundell and Katherine M. Blundell. *Concepts in Thermal Physics*. Oxford University Press, October 2009. doi:10.1093/acprof:oso/9780199562091.001.0001.
4. Ronald Fagin. The number of finite relational structures. *Discret. Math.*, 19(1):17–21, 1977. doi:10.1016/0012-365X(77)90116-9.
5. William Feller. *An introduction to probability theory and its applications. Vol. I.* Third edition. John Wiley & Sons Inc., 1968.
6. Peter Grünwald and Paul M. B. Vitányi. Shannon information and Kolmogorov complexity. *CoRR*, cs.IT/0410002, 2004. doi:10.48550/arXiv.cs/0410002.
7. Lauri Hella and Miikka Vilander. Formula size games for modal logic and μ-calculus. *J. Log. Comput.*, 29(8):1311–1344, 2019. doi:10.1093/logcom/exz025.
8. Reijo Jaakkola, Tomi Janhunen, Antti Kuusisto, Masood Feyzbakhsh Rankooh, and Miikka Vilander. Explainability via short formulas: the case of propositional logic with implementation. In *Joint Proceedings of (HYDRA 2022) and the RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, volume 3281 of *CEUR Workshop Proceedings*, pages 64–77, 2022.
9. Sik K. Leung-Yan-Cheong and Thomas M. Cover. Some equivalences between Shannon entropy and Kolmogorov complexity. *IEEE Trans. Inf. Theory*, 24(3):331–338, 1978. doi:10.1109/TIT.1978.1055891.
10. Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Texts in Computer Science. Springer, 2019.
11. Michel Loève. *Probability Theory*. Graduate texts in mathematics. Springer, 1963.
12. Oleg Pikhurko and Oleg Verbitsky. Logical complexity of graphs: A survey. In Martin Grohe and Johann A. Makowsky, editors, *Model Theoretic Methods in Finite Combinatorics - AMS-ASL Joint Special Session, Washington, DC, USA, January 5-8, 2009*, volume 558 of *Contemporary Mathematics*, pages 129–180. American Mathematical Society, 2009.
13. Alexander A. Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Comb.*, 10(1):81–93, 1990. doi:10.1007/BF02122698.
14. Andreia Teixeira, Armando Matos, Andre Souto, and Luis Filipe Coelho Antunes. Entropy measures vs. Kolmogorov complexity. *Entropy*, 13(3):595–611, 2011. doi:10.3390/e13030595.
15. Pasko Zupanovic and Domagoj Kuic. Relation between Boltzmann and Gibbs entropy and example with multinomial distribution. *Journal of Physics Communications*, 2:045002, 2018. doi:10.1088/2399-6528/aab7e1.

A Appendix

A.1 Proof of Proposition 3

Letting $\{M_i \mid i \in I\}$ enumerate the equivalence classes of $\equiv$, we have the following chain of identities.

$$\begin{aligned}
&H_S(\equiv) + \langle H_B \rangle \\
&= -\sum_{i\in I} p_\equiv(M_i) \log p_\equiv(M_i) + \sum_{i\in I} p_\equiv(M_i) \log(|[M_i]_\equiv|) \\
&= -\sum_{i\in I} p_\equiv(M_i) \log(|[M_i]_\equiv|/|\mathcal{M}|) + \sum_{i\in I} p_\equiv(M_i) \log(|[M_i]_\equiv|) \\
&= -\sum_{i\in I} p_\equiv(M_i)(\log(|[M_i]_\equiv|) - \log(|\mathcal{M}|)) + \sum_{i\in I} p_\equiv(M_i) \log(|[M_i]_\equiv|) \\
&= -\sum_{i\in I} p_\equiv(M_i) \log(|[M_i]_\equiv|) + \sum_{i\in I} p_\equiv(M_i) \log(|\mathcal{M}|) + \sum_{i\in I} p_\equiv(M_i) \log(|[M_i]_\equiv|) \\
&= \log(|\mathcal{M}|) \sum_{i\in I} p_\equiv(M_i) \\
&= \log(|\mathcal{M}|)
\end{aligned}$$

A.2 Proof of Proposition 4

The following standard calculation shows that if n is large enough, then the probability that a random τ-model of size n does not realize all the 1-types is less than $1/2$.

$$\begin{aligned}
&\Pr[\exists \pi : \mathfrak{M} \text{ does not realize } \pi] \le \sum_{\pi} \Pr[\mathfrak{M} \text{ does not realize } \pi] \\
&= \sum_{\pi} \prod_{a\in W} \Pr[a \text{ does not realize } \pi] = \sum_{\pi} \prod_{a\in W} (1 - \Pr[a \text{ does realize } \pi]) \\
&= \sum_{\pi} \prod_{a\in W} (1 - 2^{-k}) = n(1-2^{-k})^n \to 0, \text{ as } n \to \infty
\end{aligned}$$

In the inequality we used union bound while in the first equality we used the fact that the events "a does not realize π", for $a \in W$, are independent.

A.3 Proof of Lemma 6 continued

$\vee$-move: We show that for any $\vee$-move S makes, D can choose one of the following positions P_1, P_2 that satisfies both conditions (a) and (b).

Let $\mathcal{A}_1, \mathcal{A}_2$ and r_1, r_2 be the choices of S. We assume $\mathcal{A}_1, \mathcal{A}_2 \neq \emptyset$. Let π_i be a type. If π_i is of kind 1, then π_i is still of kind 1 in both following positions and $h_i(P) = 2k = h_i(P_1) = h_i(P_2)$, since $\mathcal{B}$ remains unchanged in both positions. If π_i is of kind 2, then there are propositionally equivalent $(\mathfrak{M}_0, j) \in \mathcal{A}$ and $(\mathfrak{M}_i, l) \in \mathcal{B}$. We have $(\mathfrak{M}_0, j) \in \mathcal{A}_1$ or $(\mathfrak{M}_0, j) \in \mathcal{A}_2$ so π_i is still a type of kind 2 in one of the following positions and $h_i(P_1) = 2k$ or $h_i(P_2) = 2k$. Similarly if π_i is of kind 3, then $(\mathfrak{M}_0, i) \in \mathcal{A}_1$ or $(\mathfrak{M}_0, i) \in \mathcal{A}_2$ so π_i remains a type of kind 3 in one of the following positions and $h_i(P_1) = h_i(P)$ or $h_i(P_2) = h_i(P)$. Furthermore, each type with positive hardness in P still has positive hardness in at least one of P_1 or P_2 so $\#h^+(P_1) + \#h^+(P_2) \ge \#h^+(P)$. Thus

$$h(P_1) + h(P_2) = \sum_{1 \le i \le n} h_i(P_1) + \sum_{1 \le i \le n} h_i(P_2) + \#h^+(P_1) + \#h^+(P_2) - 2$$
$$\geq \sum_{1 \le i \le n} h_i(P) + \#h^+(P) - 1 - 1 = h(P) - 1.$$

Now $r_1 + r_2 = r - 1 < h(P) - 1 \leq h(P_1) + h(P_2)$ so we have $r_1 < h(P_1)$ or $r_2 < h(P_2)$. In addition, since all types are of the same kind as in position P, condition (b) still holds.

A.4 Proof of Lemma 9 continued

■$^{<d}$**-move:** Let $d \in \mathbb{N}$ be the number chosen by S. For each $(\mathfrak{M}, w) \in \mathcal{A}$, S chooses $n - d + 1$ different points from the model $\mathfrak{M}$. Let A be the set of all points chosen this way. For each $(\mathfrak{M}_{i\to j}, w) \in \mathcal{B}$, let $B_{i\to j}$ be the set of d points chosen by S. Let $\mathrm{tp}_{\mathfrak{M}}(X)$ be the set of types realized by the set X of points in the model $\mathfrak{M}$. We consider the following two cases:

1. The model $\mathfrak{M}$ has at least $n - d + 2$ points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Let $e = (i, j) \in E$. The model $\mathfrak{M}_{i\to j}$ only differs from $\mathfrak{M}$ by the type of one point so $\mathfrak{M}_{i\to j}$ has at least $n - d + 1$ points that satisfy types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Since $|B_{i\to j}| = d$, there is at least one point in $B_{i\to j}$ with a type from $\mathrm{tp}_{\mathfrak{M}}(A)$. Thus there are propositionally equivalent $(\mathfrak{M}, w') \in \mathcal{A}'$ and $(\mathfrak{M}_{i\to j}, v') \in \mathcal{B}'$. Thus the edge $e = (i, j)$ is still present in the graph $\mathcal{G}(\mathcal{A}', \mathcal{B}')$ of the following position. This applies for every $e \in E$ so $R(P') = R(P)$.
2. The model $\mathfrak{M}$ has exactly $n - d + 1$ points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Now A is the set of those $n - d + 1$ points. We first consider edges $e = (i, j) \in E$ with $\pi_i \notin \mathrm{tp}_{\mathfrak{M}}(A)$ or $\pi_j \in \mathrm{tp}_{\mathfrak{M}}(A)$. For any edge of this kind, the model $\mathfrak{M}_{i\to j}$ has at least $n - d + 1$ points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$ so at least one of the d points in $B_{i\to j}$ has a type from $\mathrm{tp}_{\mathfrak{M}}(A)$. As in case 1, this means that all these edges are still present in the graph $\mathcal{G}(\mathcal{A}', \mathcal{B}')$.
   Let us then consider the rest of the edges $e = (i, j) \in E$ with $\pi_i \in \mathrm{tp}_{\mathfrak{M}}(A)$ and $\pi_j \notin \mathrm{tp}_{\mathfrak{M}}(A)$. For an edge of this kind, the model $\mathfrak{M}_{i\to j}$ has only $n - d$ points with types from $\mathrm{tp}_{\mathfrak{M}}(A)$. Thus if S chooses the d points of $B_{i\to j}$ to be exactly the points with types not in $\mathrm{tp}_{\mathfrak{M}}(A)$, then $\mathfrak{M}_{i\to j}$ has no propositionally equivalent counterpart on the other side and the edge e is not present in the graph $\mathcal{G}(\mathcal{A}', \mathcal{B}')$.
   We then consider the condition of the claim in the position $P' = (r - d, \mathcal{B}', \mathcal{A}')$. We saw above that S can only eliminate an edge $e = (i, j)$ if $\pi_i \in \mathrm{tp}_{\mathfrak{M}}(A)$, $\pi_j \notin \mathrm{tp}_{\mathfrak{M}}(A)$ and $\mathrm{tp}_{\mathfrak{M}_{i\to j}}(B_{i\to j}) \subseteq \mathrm{tp}_{\mathfrak{M}}(W) \setminus \mathrm{tp}_{\mathfrak{M}}(A)$. Thus we denote $\mathrm{tp}(B) := \mathrm{tp}_{\mathfrak{M}}(W) \setminus \mathrm{tp}_{\mathfrak{M}}(A)$. All edges of this kind can be covered via the cover $C_B = \{j^- \mid \pi_j \in \mathrm{tp}(B)\}$. The cost of this cover is the total number of points with types from $\mathrm{tp}(B)$ in the model $\mathfrak{M}$. By the definition of $\mathrm{tp}(B)$ the cost is $r(C_B) = n - |A| = n - (n - d + 1) = d - 1$. Let C' be a cover of $\mathcal{G}(\mathcal{A}', \mathcal{B}')$ with minimal cost so $R(P') = r(C')$. Now $C' \cup C_B$ is a cover of $\mathcal{G}(\mathcal{A}, \mathcal{B})$ with cost $R(P') + d - 1$. Thus $r < R(P) \leq R(P') + d - 1$ so $r - d < R(P')$ and the condition of the claim is maintained.

A.5 Proof of Proposition 16

In this section we use $\equiv$ to denote $\equiv_{\mathrm{FO}[\tau]}$. Our goal is to show that the expected Boltzmann entropy of $\equiv$ grows asymptotically slower than its expected description complexity.

We start by bounding the expected Boltzmann entropy from above. For every equivalence class M of $\equiv$ we have by Proposition 1 that

$$\log(|M|) \leq \log(n!) = n\log(n) - n\log(e) + \Theta(\log(n)),$$

which in turn implies that $\langle H_B \rangle \leq n\log(n) - n\log(e) + \Theta(\log(n))$.

Next we will derive a lower bound on the expected description complexity of $\equiv$. Let c be a constant such that with high probability $C_{\mathrm{FO}[\tau]}(M) \geq c\big(\frac{n^m}{\log(n)}\big)$. (Theorem 14 guarantees that such a constant exists.) In [4] it was proved that with high probability a random τ-model is rigid, i.e., it has no non-trivial automorphism. Since the isomorphism class of a rigid τ-model is of size $n!$, we have that with high probability a random member of $\equiv$ has size $n!$. Using a union bound argument we have that

$$\lim_{n\to\infty} \Pr\left[C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right) \text{ and } |M| = n!\right] = 1 \tag{6}$$

In particular, the above probability is at least, say, $1/2$ when n is large enough. In other words, for n large enough, at least half of the isomorphism classes (of models of size n) have size $n!$ and their description complexity is at least $c\big(\frac{n^m}{\log(n)}\big)$.

Now we can bound the expected description complexity from below. First, we have that

$$\sum_M p_{\equiv}(M) C_{\mathrm{FO}[\tau]}(M) \geq \sum_{C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right)} p_{\equiv}(M) C_{\mathrm{FO}[\tau]}(M)$$

$$\geq c\left(\frac{n^m}{\log(n)}\right) \sum_{C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right)} p_{\equiv}(M) = c\left(\frac{n^m}{\log(n)}\right) \frac{1}{2^{p(n)}} \sum_{C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right)} |M|.$$

We want a constant lower bound on the expression

$$\frac{1}{2^{p(n)}} \sum_{C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right)} |M|$$

which expresses the probability that the isomorphism class of random model of size n has description complexity at least $c\big(\frac{n^m}{\log(n)}\big)$. Using Equation (6), we have for n large enough the following estimates:

$$\frac{1}{2^{p(n)}} \sum_{C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right)} |M| \geq \frac{1}{2^{p(n)}} \sum_{\substack{C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right) \\ |M| = n!}} |M| = \frac{n!}{2^{p(n)}} \sum_{\substack{C_{\mathrm{FO}[\tau]}(M) \geq c\left(\frac{n^m}{\log(n)}\right) \\ |M| = n!}} 1$$

$$\geq \frac{n!}{2^{p(n)}} (1/2) \frac{2^{p(n)}}{n!} = 1/2.$$

Thus for n large enough we have that $\langle C_{\mathrm{FO}[\tau]} \rangle \geq (1/2) c\big(\frac{n^m}{\log(n)}\big)$, which certainly grows faster than $n\log(n) - n\log(e) + \Theta(\log(n)) \geq \langle H_B \rangle$.

Induced Matching Below Guarantees: Average Paves the Way for Fixed-Parameter Tractability

Tomohiro Koana
Faculty IV, Institute of Software Engineering and Theoretical Computer Science,
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

In this work, we study the Induced Matching problem: Given an undirected graph G and an integer ℓ, is there an induced matching M of size at least ℓ? An edge subset M is an induced matching in G if M is a matching such that there is no edge between two distinct edges of M. Our work looks into the parameterized complexity of Induced Matching with respect to "below guarantee" parameterizations. We consider the parameterization $u - \ell$ for an upper bound u on the size of any induced matching. For instance, any induced matching is of size at most $n/2$ where n is the number of vertices, which gives us a parameter $n/2 - \ell$. In fact, there is a straightforward $9^{n/2-\ell} \cdot n^{O(1)}$-time algorithm for Induced Matching [Moser and Thilikos, J. Discrete Algorithms]. Motivated by this, we ask: Is Induced Matching FPT for a parameter smaller than $n/2 - \ell$? In search for such parameters, we consider $\mathsf{MM}(G) - \ell$ and $\mathsf{IS}(G) - \ell$, where $\mathsf{MM}(G)$ is the maximum matching size and $\mathsf{IS}(G)$ is the maximum independent set size of G. We find that Induced Matching is presumably not FPT when parameterized by $\mathsf{MM}(G) - \ell$ or $\mathsf{IS}(G) - \ell$. In contrast to these intractability results, we find that taking the average of the two helps – our main result is a branching algorithm that solves Induced Matching in $49^{(\mathsf{MM}(G)+\mathsf{IS}(G))/2-\ell} \cdot n^{O(1)}$ time. Our algorithm makes use of the Gallai–Edmonds decomposition to find a structure to branch on.

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

Keywords and phrases Parameterized Complexity, Below Guarantees, Induced Matching, Gallai-Edmonds Decomposition

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.39

Related Version *Full Version*: `https://arxiv.org/abs/2212.13962`

Funding *Tomohiro Koana*: Supported by the DFG Project DiPa, NI 369/21.

Acknowledgements The author would like to thank Niclas Boehmer for his feedback that improved the presentation of this paper.

1 Introduction

A matching in a graph is a set of pairwise non-incident edges. An induced matching is a matching such that no edge is incident with two edges from the matching. The notion of induced matchings was initially introduced by Stockmeyer and Vazirani [43]. Since then, the Induced Matching problem – given an undirected graph G and an integer $\ell \in \mathbb{N}$, we are to determine whether G has an induced matching of size ℓ – has been studied extensively. This problem is NP-hard, which was proven independently by Stockmeyer and Vazirani [43] and Cameron [2]. The NP-hardness persists on restricted graph classes, such as bipartite graphs of vertex degree at most three [31] and cubic planar graphs [12]. On the positive side, Induced Matching is polynomial-time solvable on trees [45], chordal graphs [2], weakly chordal graphs [4], circular-arc graphs [17], comparability graphs [18], and AT-free graphs [3].

In this work, we study the parameterized complexity of Induced Matching. The standard parameterization of Induced Matching takes the solution size ℓ as the parameter. For ℓ, Induced Matching is W[1]-hard, which can be easily seen by a parameterized

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

reduction from the W[1]-hard Independent Set problem. Indeed, the W[1]-hardness holds even on bipartite graphs [37]. Perhaps for this reason, many researchers have investigated the parameterized complexity of Induced Matching from multivariate perspectives. Moser and Sikdar [37] gave a kernel of size $O(\Delta^3\ell)$ for Induced Matching for the maximum degree Δ. Erman et al. [13] and Kanj et al. [22] independently found that Induced Matching admits a polynomial kernel of size $\ell^{O(d)}$ on d-degenerate graphs. This result was later complemented by Cygan et al. [9], who showed that the kernel size is basically tight – there is no kernel of size $\ell^{o(d)}$ under standard complexity assumptions. Recently, it was shown that Induced Matching on c-closed graphs [23] and on weakly γ-closed graphs [24] has a kernel with at most $O(c^7\ell^8)$ vertices and $\ell^{O(\gamma)}$ vertices, respectively. We remark that the parameterized complexity on planar graphs has also received considerable attention [38, 13, 22].

In this paper, we adopt an alternative approach to tackle the fixed-parameter tractability of Induced Matching – using *above guarantees* and *below guarantees* [33, 34] (see also a very recent survey [19]). This work particularly concerns below guarantees. For a graph G, let $\mathsf{IM}(G)$ denote the maximum induced matching size of G. In a nutshell, we employ $\mathsf{UB}(G) - \ell$ as a parameter where $\mathsf{UB}(G)$ is a function on G that upper-bounds $\mathsf{IM}(G)$, i.e., $\mathsf{UB}(G) \geq \mathsf{IM}(G)$ on every graph G.

In this spirit, we first consider a trivial upper bound. For the number n of vertices, $\frac{1}{2}n$ clearly constitutes an upper bound on $\mathsf{IM}(G)$. We refer to the parameterized problem arising from this upper bound $\mathsf{UB}(G) = \frac{1}{2}n$ as Induced Matching Below Trivial Guarantee:

Induced Matching Below Trivial Guarantee (IMBTG)	
Input:	An undirected graph G and an integer ℓ.
Question:	Does G have an induced matching of size ℓ?
Parameter:	$k = \frac{1}{2}n - \ell$

This problem has been studied in the literature, albeit under different names: Moser and Thilikos [38] gave an algorithm solving IMBTG in $O^\star(9^k)$ time.[1] Subsequently, Xiao and Kou [44] developed an algorithm running in $O^\star(3.1845^k)$ time. In terms of kernelization, a kernel with $O(k^2)$ vertices was given by Moser and Thilikos [38]. Later, Xiao and Kou [44] gave an improved kernel with $O(k)$ vertices.

In parameterized complexity, whenever the fixed-parameter tractability with respect to a parameter k is discovered, one asks whether the "boundary of tractability" can be taken further, that is, fixed-parameter tractability is achievable for a parameter k' smaller than k (i.e., $k' \leq g(k)$ for some function g). (An analogous question arises when W-hardness for k is discovered as well – does fixed-parameter parameter tractability hold for a parameter k' larger than k?) This question appears prominently in multivariate algorithmics [26, 40] and structural parameterizations [1, 15]. We ask ourselves this type of question for Induced Matching parameterized by below guaranteed values. More precisely, the main question we challenge in this work is the following:

($\star$) Is Induced Matching FPT for a parameterization smaller than that of IMBTG?

Let us remark we are not the first to address this kind of question in the context of above and below guarantee parameterizations: The Vertex Cover problem – given an undirected graph G and an integer ℓ, decide whether there is a set of at most ℓ vertices that is incident with every edge – is one of the problems where this kind of question was

[1] The $O^\star$ notation suppresses the polynomial factor in the input size.

considered. A simple branching algorithm solves Vertex Cover in $O^\star(2^\ell)$ time. For any graph G, it holds that $\mathsf{VC}(G) \geq \mathsf{LP}(G) \geq \mathsf{MM}(G)$, where $\mathsf{VC}(G)$, $\mathsf{MM}(G)$, and $\mathsf{LP}(G)$ denote the minimum vertex cover size, the maximum matching size, and the optimum of the linear programming relaxation of Vertex Cover, respectively. These inequalities give rise to the above-guarantee parameterizations, $\ell - \mathsf{MM}(G)$ and $\ell - \mathsf{LP}(G)$. These parameterizations have been extensively studied [10, 21, 28, 29, 39, 41, 42] – it has been shown that Vertex Cover FPT is with respect to $\ell - \mathsf{MM}(G)$ as well as $\ell - \mathsf{LP}(G)$. Notably, the border of tractability was further extended to $\ell - (2\mathsf{LP}(G) - \mathsf{MM}(G))$ [16, 27]. Let us also remark that the above guarantee parameterizations of Max Cut and related problems have been also extensively studied [6, 7, 14, 32, 35, 33, 36].

Now we come back to the question ($\star$) on Induced Matching. We wish to find an upper bound on the maximum induced matching size which is stricter than $\frac{1}{2}n$. The maximum matching size $\mathsf{MM}(G)$ serves as such a bound: Clearly, $\mathsf{MM}(G) \leq \frac{1}{2}n$. Moreover, $\mathsf{MM}(G)$ is an upper bound on $\mathsf{IM}(G)$, since every induced matching is a matching. This leads to the following parameterized problem:

Induced Matching Below Maximum Matching (IMBMM)

Input: An undirected graph G and an integer ℓ.
Question: Does G have an induced matching of size ℓ?
Parameter: $k = \mathsf{MM}(G) - \ell$

As graphs G with $\mathsf{MM}(G) = \mathsf{IM}(G)$ have been of theoretical interest, there are known results on IMBMM: Kobler and Rotics [25] gave a polynomial-time algorithm for $k = 0$. Cameron and Walker [5] extended this result by providing a structural characterization of graphs G with $\mathsf{MM}(G) = \mathsf{IM}(G)$. Later, Duarte et al. [11] developed an algorithm for IMBMM that runs in $n^{O(k)}$ time. However, it has been left open whether IMBMM is FPT. Filling the gap, we prove in Section 4 that IMBMM is W[2]-hard, i.e., presumably not FPT. This implies that taking $\mathsf{MM}(G)$ as the upper bound falls short to answer the question ($\star$).

Next, we consider another natural upper bound: the maximum independent set size $\mathsf{IS}(G)$. Since an induced matching of size ℓ contains an independent set of size ℓ, any graph satisfies $\mathsf{IS}(G) \geq \mathsf{IM}(G)$, which gives the following parameterization:

Induced Matching Below Independent Set (IMBIS)

Input: An undirected graph G and an integer ℓ.
Question: Does G have an induced matching of size ℓ?
Parameter: $k = \mathsf{IS}(G) - \ell$

Although $\frac{1}{2}n$ and $\mathsf{IS}(G)$ are incomparable in general (consider complete graphs and empty graphs), the parameter of IMBIS is essentially smaller compared to that of IMBTG: Observe that for $\ell > 0$, the graph G' on $2n$ vertices obtained from G by adding n vertices adjacent to all other vertices fulfills $\mathsf{IM}(G') \geq \ell$ if and only if $\mathsf{IM}(G) \geq \ell$. The maximum independent set size of G' is at most n – half the number of vertices in G'. Thus, if IMBIS was fixed-parameter tractability, then fixed-parameter tractability of IMBTG would follow, answering our question ($\star$). We find, however, that IMBIS is NP-hard for $k = 0$ even if an independent set of size ℓ is provided as part of the input.

Somewhat dismayed by the previous two negative results, we look into the upper bound obtained by taking the average of $\mathsf{MM}(G)$ and $\mathsf{IS}(G)$, that is, $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G))$. For any graph G, we have

$$\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \mathsf{IM}(G) = \frac{1}{2}(\mathsf{MM}(G) - \mathsf{IM}(G)) + \frac{1}{2}(\mathsf{IS}(G) - \mathsf{IM}(G)) \geq 0,$$

implying that $\mathsf{IM}(G) \leq \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G))$. (This equation also implies that the parameterization by $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$ is larger compared to the parameterization of IMBMM and IMBIS.) On the other hand, we have $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) \leq \frac{1}{2}(\mathsf{VC}(G) + \mathsf{IS}(G)) \leq \frac{1}{2}n$. Hence, the parameterization obtained from the average is indeed smaller than the trivial below guarantee. Formally, we study the following:

Induced Matching Below Average (IMBA)	
Input:	An undirected graph G and an integer ℓ.
Question:	Does G have an induced matching of size ℓ?
Parameter:	$k = \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$

The main result of this work is an FPT algorithm for IMBA that runs in time $O^\star(49^k)$.

▶ **Theorem 1.** *IMBA can be solved in $O^\star(49^k)$ time.*

In other words, we identify a novel below-guarantee parameterization for Induced Matching smaller than that of IMBTG that yields an FPT algorithm, thereby positively answering our question ($\star$). To our surprise, it turns out that an answer to our question arises from using the average of two upper bounds as an upper bound.

To prove Theorem 1, we give a branching algorithm in which the *measure* $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$ decreases by $\frac{1}{2}$ in every branching step. As we will see, branching in a naïve way (which leads to an FPT algorithm for IMBTG) does not always decrease the measure. To work around this issue, we develop branching rules based on the Gallai–Edmonds decomposition. To establish the correctness of our algorithm, we reveal a structural property of graphs G with $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$, which may be of independent interest.

2 Preliminaries

2.1 Notation

We denote the set $\{1, \ldots, t\}$ of integers by $[t]$. All graphs are simple and undirected. For a graph G, let $V(G)$ and $E(G)$ denote the set of vertices and edges, respectively. Let $v \in V(G)$ be a vertex in G and let $X \subseteq V(G)$ be a vertex set. We use $N(v)$ to denote the neighborhood of v (the set of vertices adjacent to v) and $N(X) = \bigcup_{v \in V} N(v) \setminus X$ to denote the neighborhood of X. Let $\deg(v) = |N(v)|$ denote the degree of v. Let $G[X]$ denote the subgraph induced by X. We use $G - X$ to denote $G[V(G) \setminus X]$, i.e., the graph obtained from G by deleting X. We use the shorthand $G - v$ for $G - \{v\}$. A triplet (u, v, w) of pairwise adjacent vertices is a *triangle*. A vertex v with $\deg(v) = 0$ is *isolated*. A vertex v with $\deg(v) = 1$ is a *pendant vertex*. A pair (u, v) of adjacent vertices with $\deg(u) = \deg(v) = 1$ is an *isolated edge*. A triangle is a *pendant triangle* if $\deg(u) = \deg(w) = 2$.

In our algorithm, we apply *reduction rules* and *branching rules*. Herein, a reduction rule (branching rule) is a polynomial-time procedure that given an instance I, returns an instance I' (a set of instances $I_1, \ldots, I_c$, respectively). We say that a reduction rule (branching rule) is *correct* if I is equivalent to I', i.e., I is a yes-instance if and only if I' is a yes-instance (I is a yes-instance if and only if there is some $c' \in [c]$ such that $I_{c'}$ is a yes-instance, respectively).

2.2 Parameterized complexity

In parameterized complexity, each instance of a problem is equipped with a parameter, usually denoted by the symbol k. We say that a parameterized problem is fixed-parameter tractable (FPT) if there is an algorithm that solves it in $f(k) \cdot |I|^{O(1)}$ time, where f is a computable function only depending on k and $|I|$ is the input size. We denote the running time as $O^\star(f(k))$, where the polynomial factor in $|I|$ is suppressed in the $O^\star$ notation. For more in-depth notions in parameterized complexity, we refer to the standard textbook [8].

2.3 Matching theory

We use several results from matching theory (see e.g., the book of Lovász and Plummer [30]). In particular, Kőnig's theorem and the Gallai–Edmonds structural theorem play important roles in the running time analysis for Theorem 1. Recall that a matching M in a graph G is a set of pairwise non-incident edges. If a vertex $v \in V(G)$ is incident to an edge in M, then M *covers* v. If there is no edge in M incident with v, then M *misses* v. A matching covering every vertex of G is said to be *perfect*. A matching covering all but one vertex of G is said to be *near-perfect*. A graph G is *factor-critical* if for every vertex $v \in V(G)$, $G - v$ has a perfect matching.

▶ **Theorem 2** (Kőnig's theorem)**.** *For a bipartite graph G, $\mathsf{MM}(G) = \mathsf{VC}(G)$.*

▶ **Definition 3.** *The Gallai–Edmonds decomposition of a graph G is a partition of $V(G)$ into $D(G)$, $A(G)$, and $C(G)$ where*

- $D(G) = \{v \in V(G) \mid$ *there exists a maximum matching missing* $v\}$,
- $A(G) = N(D(G))$,
- $C(G) = V(G) \setminus (A(G) \cup D(G))$.

We simply write D, A, C for $D(G), A(G), C(G)$, respectively, when G is clear from context.

▶ **Theorem 4** (The Gallai–Edmonds structure theorem (see e.g. [30]))**.** *The Gallai–Edmonds decomposition satisfies the following properties.*

- *Every connected component of $G[D]$ is factor-critical.*
- *$G[C]$ has a perfect matching.*
- *Let G' be the bipartite graph obtained from $G[D \cup A]$ by contracting every connected component of $G[D]$ into one vertex and removing edges in $G[A]$. Then, $|N_{G'}(A')| > |A'|$ for every $A' \subseteq A$.*
- *A matching M is a maximum matching if and only if the following hold:*
  - *M contains a near-perfect matching for every connected component of $G[D]$.*
  - *Every vertex in A is matched to a vertex in D.*
  - *M contains a perfect matching of $G[C]$.*
- *The Gallai–Edmonds decomposition can be computed in polynomial time.*

2.4 Cameron–Walker graphs

A graph G whose maximum matching size equals maximum induced matching size (that is, $\mathsf{MM}(G) = \mathsf{IM}(G)$) is referred to as a *Cameron–Walker graph* in the literature, as Cameron and Walker [5] gave the structural characterization of these graphs. Recall that a triangle (u, v, w) with $\deg(u) = \deg(w) = 2$ is called a pendant triangle. A triangle star is a graph obtained from a triangle by adding any number of pendant triangles to one of its vertices (see Figure 1). (When we say that we add a pendant triangle to a vertex v, it means that we add two vertices u and w and add edges such that u, v, w form a triangle.)

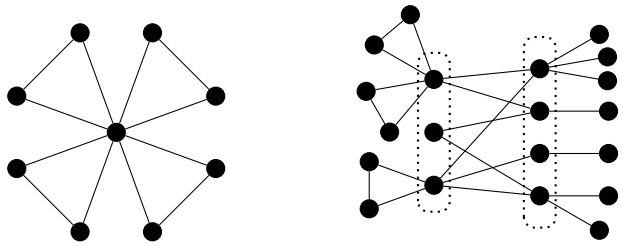


Figure 1 A triangle star with four pendant triangles (left). A Cameron–Walker graph (right). The vertex sets U and W are marked by dotted lines.

▶ **Theorem 5** ([5]). *For a connected graph G, $\mathsf{MM}(G) = \mathsf{IM}(G)$ if and only if one of the following holds:*

- *G is a star.*
- *G is a triangle star.*
- *G is obtained from a connected bipartite graph G' with a bipartition $V(G) = U \cup W$ by adding at least one pendant vertex to each vertex of U and adding any number of pendant triangles to each vertex of W.*

See Figure 1 for an illustration of a Cameron–Walker graph. The statement provided by Cameron and Walker [5] had a small mistake. The statement of Theorem 5 follows a slight modification of Hibi et al. [20]. Note that Theorem 5 yields a linear-time algorithm to recognize Cameron–Walker graphs.

2.5 FPT algorithm for IMBTG

In this subsection, we briefly discuss a simple branching algorithm that solves IMBTG in $O^\star(9^k)$ time [38]. Recall that the input of IMBTG is (G, ℓ) for a graph G and $\ell \in \mathbb{N}$, and we search for an induced matching of size ℓ in G. The parameterization of IMBTG is $k = \frac{1}{2}n - \ell$. As we will see in Section 3, the reduction rules and branching rules presented here form the basis for our algorithm for IMBA, which is parameterized by $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$.

We begin with two reduction rules:

▶ **Reduction Rule 6.** *Remove an isolated vertex.*

▶ **Reduction Rule 7.** *Remove an isolated edge (including its endpoints) and decrease ℓ by one (if $\ell > 0$).*

It is straightforward to prove the correctness of these reduction rules.

▶ **Lemma 8.** *Reduction Rules 6 and 7 are correct.*

Since there is no vertex of degree at least two in any induced matching, we may branch into three instances as follows whenever there is a vertex of degree at least two:

▶ **Branching Rule 9.** *Choose a vertex $v \in V(G)$ with at least two neighbors $u, w \in V(G)$. We branch into three instances: $(G - u, \ell)$, $(G - v, \ell)$, $(G - w, \ell)$.*

The algorithm terminates in one of the following two ways:

▶ **Termination 10.** *Return yes if $\ell = 0$.*

▶ **Termination 11.** *Return no if $\frac{1}{2}|V(G)| < \ell$.*

In the simple $O^\star(9^k)$-time algorithm for IMBTG, we repeat the following:

1. Apply Reduction Rules 6 and 7 exhaustively.
2. Check the termination conditions for Termination 10 and 11.
3. If there is a vertex of degree at least two, we branch according to Branching Rule 9.

If there is no vertex of degree at least two, then applying Reduction Rules 6 and 7 exhaustively deletes all vertices in the graph. Thus, we always reach the condition for Termination 10 ($\ell = 0$) or Termination 11 ($\ell > 0$). For the running time analysis, note that our algorithm terminates if the value of $\frac{1}{2}|V(G)| - \ell$ is negative (Termination 11). This value decreases by $\frac{1}{2}$ each time we apply Branching Rule 9: $|V(G)|$ decreases by one and ℓ remains the same. Moreover, this value does not increase by applying Reduction Rules 6 and 7. Since the value of $\frac{1}{2}|V(G)| - \ell$ equals k for the input instance, we apply Branching Rule 9 at most $2k + 1$ times. (A more careful analysis yields an upper bound of $2k$.) Hence, this algorithm runs in time $O^\star(3^{2k+1}) = O^\star(9^k)$.

We remark that Branching Rule 9 can be strengthened as follows:

▶ **Branching Rule 12.** *Choose two adjacent vertices $u, v \in V(G)$ with $N(\{u, v\}) \neq \emptyset$. We branch into three instances: $(G - u, \ell)$, $(G - v, \ell)$, $(G - N(\{u, v\}), \ell)$*

To see the correctness, observe that we have two cases: If G has an induced matching M of size ℓ containing uv, then M is an induced matching in $G - N(\{u, v\})$ as well. Otherwise, G has no induced matching of size ℓ covering both u and v, and thus u or v must be deleted.

Due to a space limitation, some proofs are in the appendix.

3 Algorithm for IMBA

In this section, we give an FPT algorithm IMBA: Given a graph G and an integer ℓ, IMBA asks whether G has an induced matching of size ℓ with the parameterization $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IM}(G)) - \ell$. We start with an overview in Section 3.1. We describe reduction rules in Section 3.2 and branching rules in Section 3.3.

3.1 Overview

Our algorithm for IMBA is an intricate adaptation of the FPT algorithm given in Section 2.5. In addition to Reduction Rules 6 and 7, we use another reduction rule on pendant triangles (Reduction Rule 14). We will also derive our branching rules from Branching Rules 9 and 12.

To ensure that our algorithm runs in $O^\star(49^k)$ time, we define the *measure* of an instance $\mathcal{I} = (G, \ell)$ of IMBA as $\mu(\mathcal{I}) = \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$. Note that the parameter k for IMBA is the measure of the input instance.[2] We design our algorithm such that (i) every branching rule generates at most seven instances whose measure is smaller by at least $\frac{1}{2}$ and (ii) there is no increase in the measure throughout. This way, since we start with the measure at k, we end up with an instance whose measure is zero (or smaller) within $2k$ branching steps.[3] As we show in Section 3.4, our algorithm correctly identifies yes-instances before the measure becomes zero or smaller. So we terminate returning no after $2k$ branching steps:

▶ **Termination 13.** *Return no if branching rules have been applied $2k$ times.*

[2] As is often the case in parameterized complexity, we assume that k is given along with input. We will not change the value of k in this section.

[3] Our algorithm does not compute the measure. In fact, it is even computationally challenging to determine whether $\mu(\mathcal{I}) \leq 0$ (this is equivalent to $\mathsf{IS}(G) \leq 2\ell - \mathsf{MM}(G)$).

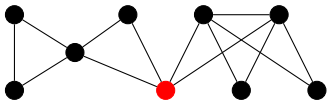


Figure 2 The maximum matching size and independent set size remain 4 even after deleting the red vertex.

Termination 13 ensures that the search tree has depth at most $2k$. Our algorithm thus runs in $O^\star(7^{2k}) = O^\star(49^k)$ time.

Our algorithm repeats the following until one of the conditions for a termination is met.

1. Apply Reduction Rules 6, 7, and 14 exhaustively.
2. Check the termination conditions for Termination 10, 11, and 13.
3. If there is a vertex of degree at least two, we apply one of the branching rules in Section 3.3.

We remark that we check the condition of Termination 10 before that of Termination 13.

As for how to branch, we want to branch in such a way that the measure drops by $\frac{1}{2}$. To this end, branching according to Branching Rule 9 in the naïve way is seemingly not appropriate because the measure may not decrease. This challenge is illustrated in Figure 2: The maximum matching size and the maximum independent set both remain unchanged (thus so does the measure) after deleting the red vertex. To find a set of vertices whose deletion guarantees a decrease in the measure by $\frac{1}{2}$, we will exploit the Gallai–Edmonds decomposition.

3.2 Reduction Rules

Our algorithm employs Reduction Rules 6 and 7. We also introduce another reduction rule on pendant triangles. Recall that a triplet (u, v, w) is a pendant triangle if u, v, w are pairwise adjacent and $\deg(u) = \deg(w) = 2$.

▶ **Reduction Rule 14.** *If there is a pendant triangle (u, v, w) with $\deg(u) = \deg(w) = 2$, then delete v.*

▶ **Lemma 15.** *Reduction Rule 14 is correct.*

As discussed in Section 3.1, we need to ensure that these reduction rules do not increase the measure. Recall that for an instance $\mathcal{I} = (G, \ell)$, its measure is $\mu(\mathcal{I}) = \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$.

▶ **Lemma 16.** *Let $\mathcal{I} = (G, \ell)$ be an instance of IMBA and let $\mathcal{I}' = (G', \ell')$ be an instance obtained by applying Reduction Rule 6, 7, or 14. Then, $\mu(\mathcal{I}') \leq \mu(\mathcal{I})$.*

3.3 Branching Rules

In this subsection, we describe our branching rules, which are based on the Gallai–Edmonds decomposition (Definition 3). Recall that the vertex set $V(G)$ is divided into three parts: A, C, and D. (Note that we have to recompute the Gallai–Edmonds decomposition after a vertex is deleted by our reduction rule or branching rule.) In our algorithm, we apply the first reduction rule for which the condition is met in the following:

- If $C \neq \emptyset$, then apply Branching Rule 17.
- If A is not independent, then apply Branching Rule 21.
- If there is a connected component S of $G[D]$ such that $G[S]$ is a triangle, then apply Branching Rule 25.

- If there is a connected component S of $G[D]$ such that $G[S]$ is a triangle star, then apply Branching Rule 28.
- If there is a connected component S of $G[D]$ with $|S| \geq 5$, then apply Branching Rule 33.
- If none of the above holds, then apply Branching Rule 36.

For each branching rule, we prove that it is correct and that it decreases the measure by at least $\frac{1}{2}$. We assume that reduction rules are exhaustively applied throughout the subsection.

In the first branching rule, we apply Branching Rule 9 on a vertex in C.

▶ **Branching Rule 17.** *Choose a vertex $v \in C$ with at least two neighbors $u, w \in N(v)$. We branch into three instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [3]$, where $G_1 = G - v$, $G_2 = G - u$, and $G_3 = G - w$.*

▶ **Lemma 18.** *Branching Rule 17 is correct.*

▶ **Lemma 19.** *Let v be a vertex in $A \cup C$ and let $G' = G - v$ be the graph obtained by deleting v. Then, $\mathsf{MM}(G') \leq \mathsf{MM}(G) - 1$.*

▶ **Lemma 20.** *In Branching Rule 17, $\mu(\mathcal{I}_i) \leq \mu(\mathcal{I}) - \frac{1}{2}$ for all $i \in [3]$.*

We claim that if we cannot branch according to Branching Rule 17 (that is, every vertex in C has at most one neighbor), then $C = \emptyset$. Assume for contradiction that there is a vertex $v \in C$. Then, v has a neighbor u in C, since $G[C]$ has a perfect matching by Theorem 4. If neither u nor v has other neighbors, then Reduction Rule 7 applies. Thus, we have $C = \emptyset$.

In the next rule, we branch on two adjacent vertices in A adapting Branching Rule 12.

▶ **Branching Rule 21.** *Choose two adjacent vertices $u, v \in A$. We branch into three instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [3]$. In the first two, we delete u or v, i.e., $G_1 = G - u$ and $G_2 = G - v$. In the third branch, we delete $N(\{u, v\})$, i.e., $G_3 = G - N(\{u, v\})$.*

▶ **Lemma 22.** *Branching Rule 21 is correct.*

Note that A is an independent set when we cannot apply Branching Rule 21. We will later describe how we branch on the vertices in D. Before doing so, we show that deleting two vertices from a nontrivial (i.e., size at least two) connected component S of $G[D]$ decreases the measure. (In fact, S is of size at least three because $G[S]$ is factor-critical by Theorem 4.)

▶ **Lemma 23.** *Let S be a connected component of $G[D]$ with at least three vertices and let $u, v \in S$. Let $G' = G - u - v$ be the graph obtained from G by deleting u and v. Then, $\mathsf{MM}(G') \leq \mathsf{MM}(G) - 1$.*

Proof. By Theorem 4, every maximum matching of G contains a near-perfect matching M_S of $G[S]$. Since $G[S]$ is factor-critical, S consists of an odd number of vertices, and we have $|M_S| = \frac{1}{2}(|S| - 1)$. Since $u, v \in S$ are deleted from G', any matching in $G'[S]$ contains at most $\lfloor \frac{1}{2}(|S| - 2) \rfloor = \frac{1}{2}(|S| - 3) = |M_S| - 1$ edges. It follows that G' has no matching of size $\mathsf{MM}(G)$. ◀

For branching on the vertices in D, we start with the connected components of $G[D]$ which form a triangle. We first prove a lemma concerning such triangles.

▶ **Lemma 24.** *Let S be a connected component of $G[D]$ such that $G[S]$ is a triangle. There exist two vertices $u, v \in S$ such that u and v have a neighbor u' and v' in A, respectively (possibly $u' = v'$).*

▶ **Branching Rule 25.** *Let $S = \{u, v, w\}$ be a connected component of $G[D]$ such that $G[S]$ is a triangle and u and v have a neighbor u' and v' in A, respectively (such vertices exist by Lemma 24). Let $H_1 = G - u'$, $H_2 = G - u$, and $H_3 = G - v$. We generate seven instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [7]$, where $G_1 = H_1$, $G_2 = H_2 - v'$, $G_3 = H_2 - v$, $G_4 = H_2 - w$, $G_5 = H_3 - u'$, $G_6 = H_3 - u$, $G_7 = H_3 - w$.*

▶ **Lemma 26.** *Branching Rule 25 is correct.*

We verify the drop in measure in Lemma 30. We then look into the connected components of $G[D]$ which form triangle stars (with at least two pendant triangles). We show a lemma analogous to Lemma 24.

▶ **Lemma 27.** *Let S be a connected component of $G[D]$ such that $G[S]$ is a triangle star with at least two pendant triangles. There exist two nonadjacent vertices $u, v \in S$ such that u and v have a neighbor u' and v' in A, respectively (possibly $u' = v'$).*

▶ **Branching Rule 28.** *Let S be a connected component of $G[D]$ such that $G[S]$ is a triangle star with at least two pendant triangles. Let $u, v \in S$ be vertices as specified in Lemma 27, and let $u', v' \in A$ be neighbors of u, v, respectively. Also, let $u'', v'' \in S$ be neighbors of u, v, respectively, which are not the center of $G[S]$. Let $H_1 = G - u'$, $H_2 = G - u$, and $H_3 = G - u''$. We generate seven instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [7]$, where $G_1 = H_1$, $G_2 = H_2 - v$, $G_3 = H_2 - v'$, $G_4 = H_2 - v''$, $G_5 = H_3 - v$, $G_6 = H_3 - v'$, $G_7 = H_3 - v''$.*

Its correctness can be argued in the same way as we did for Branching Rule 25.

▶ **Lemma 29.** *Branching Rule 28 is correct.*

▶ **Lemma 30.** *In Branching Rules 25 and 28, $\mu(I_i) \leq \mu(I) - \frac{1}{2}$ for all $i \in [7]$.*

Proof. Observe that we delete a vertex from A or two vertices from S in every branch. Thus, we obtain $\mathsf{MM}(G_i) \leq \mathsf{MM}(G) - 1$ by Lemmas 19 and 23. It follows that $\mu(\mathcal{I}_i) = \frac{1}{2}(\mathsf{MM}(G_i) + \mathsf{IS}(G_i)) - \ell = \frac{1}{2}((\mathsf{MM}(G) - 1) + \mathsf{IS}(G)) - \ell = \mu(\mathcal{I}) - \frac{1}{2}$. ◀

Finally, we look into nontrivial connected components of $G[D]$ which are not triangles or triangle stars. We first prove two lemmas that help to develop a branching rule. Note that each nontrivial component (unless it is a triangle) has at least five vertices.

▶ **Lemma 31.** *Let H be a connected graph on at least four vertices that is not a star. Then, H has a path on four vertices.*

▶ **Lemma 32.** *Let H be a factor-critical graph that is not a triangle star and let v be an arbitrary vertex in $V(H)$. Then, $H - v$ has a vertex of degree at least two.*

▶ **Branching Rule 33.** *Let S be a nontrivial connected component of $G[D]$ with $|S| \geq 5$ which is not a triangle star. Choose a path (u, v, w, x) on four vertices in $G[S]$ (such a path exists by Lemma 31). We have seven branches $\mathcal{I}_i = (G_i, \ell)$ as follows:*

1. *Choose a vertex v' with at least two neighbors $v'_1, v'_2 \in S$ in $G - v$ (such a vertex exists by Lemma 32). Let $G_1 = G - v - v'$, $G_2 = G - v - v'_1$, and $G_3 = G - v - v'_2$.*
2. *Choose a vertex w' with at least two neighbors $w'_1, w'_2 \in S$ in $G - w$ (such a vertex exists by Lemma 32). Let $G_4 = G - w - w'$, $G_5 = G - w - w'_1$, and $G_6 = G - w - w'_2$.*
3. *Let $G_7 = G - N(\{v, w\})$.*

▶ **Lemma 34.** *Branching Rule 33 is correct.*

Proof. Observe that $\mathcal{I}$ is a yes-instance if and only if at least one of $(G-v,\ell)$, $G(G-w,\ell)$, and $G(G-N(\{v,w\}),\ell)$ is a yes-instance by the correctness of Branching Rule 12. We branch further on the first two instances $(G-v,\ell)$ and $G(G-w,\ell)$ as in Branching Rule 9 to end up with the seven instances given above. ◀

▶ **Lemma 35.** *In Branching Rule 33, $\mu(\mathcal{I}_i) \leq \mu(\mathcal{I}) - \frac{1}{2}$ for all $i \in [7]$.*

Proof. Note that every branch deletes at least two vertices from S (in particular, $\{u,x\} \subseteq N(\{v,w\})$). We thus have $\mathsf{MM}(G_i) \leq \mathsf{MM}(G) - 1$ by Lemma 23. It follows that $\mu(\mathcal{I}_i) = \frac{1}{2}(\mathsf{MM}(G_i) + \mathsf{IS}(G_i)) - \ell \leq \frac{1}{2}(\mathsf{MM}(G) - 1 + \mathsf{IS}(G)) - \ell = \mu(\mathcal{I}) - \frac{1}{2}$. ◀

If the branching reduction rules given so far are not applicable, then G is a bipartite graph whose partition is A and D: We have $C = \emptyset$, since otherwise we can apply Branching Rule 17. We also have that A is an independent set, since otherwise Branching Rule 21 applies. Moreover, every connected component S of $G[D]$ is trivial: Note that S consists of an odd number of vertices because $G[S]$ is factor-critical. If $|S| = 3$, then S is a triangle and hence we can apply Branching Rule 25. If $|S| \geq 5$, then we can apply Branching Rule 28 or Branching Rule 33. Thus, every connected component of $G[D]$ is trivial, implying that D is an independent set as well. Now, Branching Rule 9 actually decreases the measure:

▶ **Branching Rule 36.** *Choose a vertex $v \in V(G)$ with at least two neighbors $u, w \in V(G)$. We branch into $\mathcal{I}_i = (G_i, \ell)$ for $i \in [3]$, where $G_1 = G - v$, $G_2 = G - u$, and $G_3 = G - w$.*

It is correct since Branching Rule 9 is correct.

▶ **Lemma 37.** *Branching Rule 36 is correct.*

▶ **Lemma 38.** *In Branching Rule 36, $\mu(\mathcal{I}_i) \leq \mu(\mathcal{I}) - \frac{1}{2}$ for all $i \in [3]$.*

Proof. Recall that $\mathsf{VC}(H)$ denote the minimum vertex cover size of a graph H. Since G and G_i are both bipartite, we have $\mathsf{VC}(G) = \mathsf{MM}(G)$ and $\mathsf{VC}(G_i) = \mathsf{MM}(G_i)$ by Kőnig's theorem (Theorem 2). It follows that $\mu(\mathcal{I}) = \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell = \frac{1}{2}(\mathsf{VC}(G) + \mathsf{IS}(G)) - \ell = \frac{1}{2}|V(G)| - \ell$. Analogously, we obtain $\mu(\mathcal{I}_i) = \frac{1}{2}|V(G_i)| - \ell$. Since $|V(G_i)| = |V(G)| - 1$, we have $\mu(\mathcal{I}) - \mu(\mathcal{I}_i) = \frac{1}{2}(|V(G)| - |V(G_i)|) = \frac{1}{2}$. ◀

3.4 Correctness and Running Time Analysis

We prove Theorem 1 in this subsection. For the correctness of our algorithm (the outline is given in Section 3.1), we need to show that if the input $\mathcal{I}$ is a yes-instance, then our algorithm correctly determines that $\mathcal{I}$ is a yes-instance after branching $2k$ times (avoiding Termination 13). We will show that if we end up with a yes-instance $\mathcal{I}' = (G', \ell')$ after $2k$ branching steps, then $\frac{1}{2}(\mathsf{MM}(G') + \mathsf{IS}(G')) = \mathsf{IM}(G')$ (equivalently, $\mathsf{MM}(G') = \mathsf{IS}(G') = \mathsf{IM}(G')$ since $\mathsf{MM}(G') \geq \mathsf{IM}(G')$ and $\mathsf{IS}(G') \geq \mathsf{IM}(G')$) holds. We examine the structure of graphs G' with $\frac{1}{2}(\mathsf{MM}(G') + \mathsf{IS}(G')) = \mathsf{IM}(G')$ in Lemma 39. We find that for a yes-instance $\mathcal{I}' = (G', \ell')$ with $\frac{1}{2}(\mathsf{MM}(G') + \mathsf{IS}(G')) = \mathsf{IM}(G')$, our algorithm terminates with $\ell = 0$ returning yes (Termination 10) after reduction rules are applied exhaustively (Lemma 40).

▶ **Lemma 39.** *For a connected graph G, if $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$, then one of the following holds:*

- *G is an isolated edge.*
- *G is a triangle star.*

- *G is obtained from a connected bipartite graph G' with a bipartition $V(G') = U \cup W$ by adding exactly one pendant vertex to each vertex of U and adding at least one pendant triangle to each vertex of W. In particular,* $\mathsf{MM}(G) = \mathsf{IS}(G) = \mathsf{IM}(G) = \frac{1}{2}(|V(G)| - |W|)$.

We remark that the converse of Lemma 39 holds as well. We show that an instance (G, ℓ) with $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$ can be solved by exhaustively applying reduction rules.

▶ **Lemma 40.** *Let $\mathcal{I} = (G, \ell)$ be an instance of IMBA with $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$. If $\mathcal{I}$ is a yes-instance, then the instance $\mathcal{I}' = (G', \ell')$ obtained from $\mathcal{I}$ by exhaustively applying Reduction Rules 6, 7, and 14 has $\ell' = 0$.*

Combining all our lemmas, we prove the following:

▶ **Theorem 1.** *IMBA can be solved in $O^\star(49^k)$ time.*

Proof. We first show that our algorithm is correct. Let $\mathcal{S}$ be the set of instances corresponding to leaves in the search tree. For the correctness, we must show that if the input $\mathcal{I}$ is yes-instance, then there exists an instance $\mathcal{I}' \in \mathcal{S}$ which results in a termination returning yes (Termination 10). Note that each instance $\mathcal{I}' = (G', \ell') \in \mathcal{S}$ fulfills at least one of conditions for Termination 10, 11, and 13: Suppose that the condition for Termination 13 not is met. Then, G' does not have any vertex of degree at least two, since otherwise we can apply a branching rule in Section 3.3. Thus, every vertex has degree at most one and Reduction Rules 6 and 7 deletes every vertex in the graph, resulting in Termination 10 ($\ell' = 0$) or Termination 11 ($\ell' > 0$).

If $\mathcal{I}$ is a no-instance, then every instance $\mathcal{I}' \in \mathcal{S}$ is a no-instance as well by the correctness of our reduction rules (Lemmas 8 and 15) and branching rules (Lemmas 18, 22, 26, 29, 34, and 37). Thus, our algorithm reaches Termination 11 or 13, returning no.

Suppose that $\mathcal{I}$ is a yes-instance. Since our reduction rules and branching rules are correct, there exists an instance $\mathcal{I}' = (G', \ell') \in \mathcal{S}$ such that $\mathcal{I}'$ is a yes-instance. For the sake of contradiction, assume that our algorithm incorrectly concludes that $\mathcal{I}'$ is a no-instance, i.e., (i) $\ell' > 0$ (recall that we check for Termination 10 first) and (ii) $\frac{1}{2}|V(G')| < \ell'$ or there have been $2k$ branching steps (conditions for Termination 11 and 13, respectively). By the assumption that $\mathcal{I}'$ is a yes-instance, we have $\frac{1}{2}|V(G')| \geq \mathsf{IM}(G') \geq \ell'$. We thus may assume that branching rules have been applied $2k$ times. We have shown that every branching rule decreases the measure (Lemmas 20, 30, 35, 38, and 43) by at least $\frac{1}{2}$ and that the measure does not increase by applying reduction rules (Lemma 16). Since the measure is k for the input instance $\mathcal{I}$, we have $\mu(\mathcal{I}') \leq k - 2k \cdot \frac{1}{2} = 0$. Moreover, we have

$$\mu(\mathcal{I}') = \frac{1}{2}(\mathsf{MM}(G') + \mathsf{IS}(G')) - \ell' \geq \frac{1}{2}(\mathsf{MM}(G') + \mathsf{IS}(G')) - \mathsf{IM}(G') \geq 0.$$

Here, the first inequality follows because $\mathcal{I}'$ is a yes-instance, i.e., $\mathsf{IM}(G') \geq \ell'$, and the second inequality $\frac{1}{2}(\mathsf{MM}(G')+\mathsf{IS}(G'))-\mathsf{IM}(G') = \frac{1}{2}(\mathsf{MM}(G')-\mathsf{IM}(G'))+\frac{1}{2}(\mathsf{IS}(G')-\mathsf{IM}(G')) \geq 0$ holds for any graph G'. We thus have $\mu(\mathcal{I}') = 0$ and in particular $\frac{1}{2}(\mathsf{MM}(G') + \mathsf{IS}(G')) = \mathsf{IM}(G')$. Since Reduction Rules 6, 7, and 14 are exhaustively applied, we have $\ell' = 0$ by Lemma 40, a contradiction. Thus, our algorithm correctly determines that $\mathcal{I}'$ is a yes-instance.

For the running time, note that each node in the search tree has at most seven children. Moreover, the depth of the search tree is at most $2k$ by Termination 13. Thus, our algorithm runs in $O^\star(7^{2k}) = O^\star(49^k)$ time. ◀

4 Hardness for IMBMM and IMBIS

Here, we prove the hardness for IMBMM and IMBIS. Recall that IMBMM is parameterized by $\mathsf{MM}(G) - \ell$ and IMBIS is parameterized by $\mathsf{IS}(G) - \ell$, where $\mathsf{MM}(G)$ is the maximum matching size of G, $\mathsf{IS}(G)$ is the maximum independent set size of G. These negative results

complement Theorem 1 in answering our main question: is there a parameterization smaller than $\frac{1}{2}n - \ell$ which admits an FPT algorithm? Our negative results suggest that using $\mathsf{MM}(G)$ or $\mathsf{IS}(G)$ as an upper bound of ℓ fails.

We first show that IMBMM is W[2]-hard. The hardness holds for 2-degenerate graphs. This also complements the following two results: One is an XP algorithm for IMBMM [11] and the other is an FPT algorithm for IMBMM where the maximum degree is additionally included as part of the parameter [11].

▶ **Theorem 41.** *IMBMM is W[2]-hard even on 2-degenerate bipartite graphs.*

Proof. We reduce from Dominating Set, which is W[2]-hard:

Dominating Set	
Input:	An undirected graph G and $\ell \in \mathbb{N}$.
Question:	Does G have a dominating set D (i.e., $N(D) = V(G) \setminus D$) of size at most ℓ?
Parameter:	ℓ

Let $\mathcal{I} = (G, \ell)$ be an instance of Dominating Set such that G is connected and G has at least one cycle. Let G' be the graph obtained from G by subdividing every edge vv' of G (i.e., add a vertex v'', add two edges vv'' and $v'v''$, and delete the edge vv'). Let U denote the set of vertices added by subdivisions. Note that every vertex in U corresponds to an edge in G. It is straightforward to verify that G is 2-degenerate: Let $X \subseteq V(G')$. If $X \cap U \neq \emptyset$, then $e \in X \cap U$ has degree at most two in $G[X]$. Otherwise, we have $X \subseteq U$ and thus $G[X]$ is a graph without any edge. Moreover, G' is bipartite with a bipartition $V(G) \cup U$. We show that G has a dominating set of size ℓ if and only if G' has an induced matching of size $\ell' = |V(G)| - \ell$.

First, suppose that G has a dominating set D of size exactly ℓ. By definition, every vertex $v \in V(G) \setminus D$ has at least one neighbor in D in G. Let $v_D \in D$ be one of such neighbors. Now consider a matching M' in G' in which each vertex $v \in V(G) \setminus D$ is matched to the vertex (in G') corresponding to the edge vv_D (in G). We claim that M' is an induced matching of size ℓ'. Since G' is bipartite, it suffices to verify that for two vertices $v, v' \in V(G) \setminus D$, there is no edge between v_D and v'. By the choice of v_D, we have $N_{G'}(v_D) = \{v, w\}$, where w is some vertex in D, showing that v_D and v' are not adjacent to each other. We thus have shown that M is an induced matching. Note that M has size $|V(G) \setminus D| = |V(G)| - \ell$.

Conversely, suppose that G' has an induced matching M' of size exactly ℓ'. Since G' is bipartite, M' covers exactly ℓ' vertices of $V(G)$. Let $\overline{D} \subseteq V(G)$ be the set of vertices in $V(G)$ covered by M'. For every vertex $v \in \overline{D}$, let $v' \in U$ be the vertex such that $vv' \in M'$. Since M' is an induced matching, v' corresponds to an edge between v and some vertex not in $\overline{D}$. This implies that $V(G) \setminus \overline{D}$ is a dominating set of size $|V(G)| - \ell' = \ell$ in G.

To establish the W[2]-hardness of IMBMM, it remains to show that the parameter $k = \mathsf{MM}(G') - \ell'$ associated with the instance (G', ℓ') is upper-bounded by some function of ℓ. In fact, we show that $k \leq \ell$. To do so, we show that $\mathsf{MM}(G') = |V(G)|$, i.e., there is a matching in G' that covers every vertex in $V(G)$ using Hall's theorem. To apply Hall's theorem, we must show that $|N(S)| \geq |S|$ holds in G' for each $S \subseteq V(G)$. Note that $|N(S)|$ in G' equals the number of edges incident to S in G. For $S \subseteq V(G)$, let $S_1, \ldots, S_c$ denote the connected component of $G[S]$. We claim that $|N(S_i)| \geq |S_i|$ for each $i \in [c]$. If $S_i = V(G)$, then by the assumption that G has at least one cycle, we have $|N(S_i)| = |E(G)| \geq |V(G)| = |S_i|$. Assume that $S_i \neq V(G)$. For each $i \in [c]$, note that there are at least $|S_i| - 1$ edges in $G[S_i]$. Note also that there is at least edge in G where one endpoint is in S_i and the other is in

$V(G) \setminus S_i$ (which is nonempty) by the assumption that G is connected. Since $|N(S_i)|$ is the number of edges incident to S_i in G, we have $|N(S_i)| \geq (|S_i| - 1) + 1 = |S_i|$ for each $i \in [c]$. The sets $N(S_1), \ldots, N(S_c)$ are pairwise disjoint, since otherwise for two distinct connected components S_i and S_j, an edge $e \in N(S_i) \cap N(S_j)$ would connect S_i and S_j in G. Thus, we have $|N(S)| \geq \sum_{i \in [c]} |N(S_i)| \geq \sum_{i \in [c]} |S_i| = |S|$. By Hall's theorem, G' has a matching that covers every vertex in $V(G)$, i.e., $\mathsf{MM}(G') = |V(G)|$. It follows that $k = \mathsf{MM}(G') - \ell' = |V(G)| - (|V(G)| - \ell) = \ell$. ◀

We then show that IMBIS is NP-hard even if the parameter is zero. Our hardness holds even if a maximum independent set is given as part of input.

▶ **Theorem 42.** *IMBIS is NP-hard for $k = 0$ even if an independent set of maximum size is given as part of input.*

Proof. We reduce from the NP-hard Multicolored Independent Set problem: Its input is an undirected graph G and $\ell \in \mathbb{N}$. Additionally, we are given a partition $(C_1, \ldots, C_\ell)$ of $V(G)$ into ℓ cliques. The task is to determine whether G has an independent set of size ℓ. Consider the graph G' obtained by introducing ℓ vertices $v_1, \ldots, v_\ell$ and adding edges such that $N(v_i) = C_i$ for every $i \in [\ell]$. It is not difficult to see that G has an independent set of size ℓ if and only if G' has an induced matching of size ℓ. Note that the set $\{v_1, \ldots, v_\ell\}$ is an independent set of size ℓ in G' and that this is of maximum cardinality since the vertex set can be partitioned into ℓ cliques. Thus, we have $k = \ell - \mathsf{IM}(G) = 0$. Note that our hardness holds even if $\{v_1, \ldots, v_\ell\}$ is given as part of input for Induced Matching. ◀

5 Conclusion

In this work, we discovered a new parameter $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$ for which Induced Matching is fixed-parameter tractable. This parameter is smaller than below trivial guarantee $\frac{1}{2}n - \ell$. Our main result states that Induced Matching is solvable in $O^\star(49^k)$ time for $k = \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$. This stands in contrast to our negative results: the W[2]-hardness when parameterized by $\mathsf{MM}(G) - \ell$ and the NP-hardness for $\mathsf{IS}(G) - \ell = 0$

There remain several natural questions for future research. First, is Induced Matching fixed-parameter tractable for an even smaller parameter? We remark that our negative results in Section 4 indicates that the upper bound cannot be as tight as the maximum matching size or the maximum independent set size. Another question is whether the base 49 in the running time of our algorithm can be lowered. It would also be interesting to study whether Induced Matching has a polynomial kernel with respect to $k = \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell$.

To the best of our knowledge, we are the first to propose the below (or above) guarantee parameterization, where the bound from which the parameter is derived is the average of two values. We believe that this framework will be successful for other problems as well.

References

1 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Yota Otachi. Grundy distinguishes treewidth from pathwidth. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)*, pages 14:1–14:19, 2020. doi:10.4230/LIPIcs.ESA.2020.14.

2 Kathie Cameron. Induced matchings. *Discrete Appl. Math.*, 24(1-3):97–102, 1989. doi:10.1016/0166-218X(92)90275-F.

3 Kathie Cameron. Induced matchings in intersection graphs. *Discrete Math.*, 278(1-3):1–9, 2004. doi:10.1016/j.disc.2003.05.001.

4 Kathie Cameron, R. Sritharan, and Yingwen Tang. Finding a maximum induced matching in weakly chordal graphs. *Discrete Math.*, 266(1-3):133–142, 2003. doi:10.1016/S0012-365X(02)00803-8.

5 Kathie Cameron and Tracy Walker. The graphs with maximum induced matching and maximum matching the same size. *Discrete Math.*, 299(1-3):49–55, 2005. doi:10.1016/j.disc.2004.07.022.

6 Robert Crowston, Gregory Z. Gutin, Mark Jones, and Gabriele Muciaccia. Maximum balanced subgraph problem parameterized above lower bound. *Theor. Comput. Sci.*, 513:53–64, 2013. doi:10.1016/j.tcs.2013.10.026.

7 Robert Crowston, Mark Jones, and Matthias Mnich. Max-cut parameterized above the Edwards-Erdős bound. *Algorithmica*, 72(3):734–757, 2015. doi:10.1007/s00453-014-9870-z.

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 Marek Cygan, Fabrizio Grandoni, and Danny Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. *ACM Trans. Algorithms*, 13(3):43:1–43:22, 2017. doi:10.1145/3108239.

10 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.

11 Márcio Antônio Duarte, Felix Joos, Lucia Draque Penso, Dieter Rautenbach, and Uéverton S. Souza. Maximum induced matchings close to maximum matchings. *Theor. Comput. Sci.*, 588:131–137, 2015. doi:10.1016/j.tcs.2015.04.001.

12 William Duckworth, David F. Manlove, and Michele Zito. On the approximability of the maximum induced matching problem. *J. Discrete Algorithms*, 3(1):79–91, 2005. doi:10.1016/j.jda.2004.05.001.

13 Rok Erman, Lukasz Kowalik, Matjaz Krnc, and Tomasz Walen. Improved induced matchings in sparse graphs. *Discrete Appl. Math.*, 158(18):1994–2003, 2010. doi:10.1016/j.dam.2010.08.026.

14 Michael Etscheid and Matthias Mnich. Linear kernels and linear-time algorithms for finding large cuts. *Algorithmica*, 80(9):2574–2615, 2018. doi:10.1007/s00453-017-0388-z.

15 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Comb.*, 34(3):541–566, 2013. doi:10.1016/j.ejc.2012.04.008.

16 Shivam Garg and Geevarghese Philip. Raising the bar for vertex cover: Fixed-parameter tractability above a higher guarantee. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1152–1166, 2016. doi:10.1137/1.9781611974331.ch80.

17 Martin Charles Golumbic and Renu C. Laskar. Irredundancy in circular arc graphs. *Discrete Appl. Math.*, 44(1-3):79–89, 1993. doi:10.1016/0166-218X(93)90223-B.

18 Martin Charles Golumbic and Moshe Lewenstein. New results on induced matchings. *Discrete Appl. Math.*, 101(1-3):157–165, 2000. doi:10.1016/S0166-218X(99)00194-8.

19 Gregory Z. Gutin and Matthias Mnich. A survey on graph problems parameterized above and below guaranteed values. *CoRR*, abs/2207.12278, 2022. doi:10.48550/arXiv.2207.12278.

20 Takayuki Hibi, Akihiro Higashitani, Kyouko Kimura, and Augustine B O'Keefe. Algebraic study on Cameron–Walker graphs. *J. Algebra*, 422:257–269, 2015. doi:doi.org/10.1016/j.jalgebra.2014.07.037.

21 Yoichi Iwata, Keigo Oka, and Yuichi Yoshida. Linear-time FPT algorithms via network flow. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA 2014)*, pages 1749–1761, 2014. doi:10.1137/1.9781611973402.127.

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

23 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Exploiting c-closure in kernelization algorithms for graph problems. In *Proceedings of the 28th Annual European Symposium on Algorithms (ESA 2020)*, pages 65:1–65:17, 2020. doi:10.4230/LIPIcs.ESA.2020.65.

24 Tomohiro Koana, Christian Komusiewicz, and Frank Sommer. Essentially tight kernels for (weakly) closed graphs. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, pages 35:1–35:15, 2021. doi:10.4230/LIPIcs.ISAAC.2021.35.

25 Daniel Kobler and Udi Rotics. Finding maximum induced matchings in subclasses of claw-free and P_5-free graphs, and in graphs with matching and induced matching of equal maximum size. *Algorithmica*, 37(4):327–346, 2003. doi:10.1007/s00453-003-1035-4.

26 Christian Komusiewicz and Rolf Niedermeier. New races in parameterized algorithmics. In *Proceedings of 37th International Symposium on Mathematical Foundations of Computer Science 2012 (MFCS 2012)*, pages 19–30, 2012. doi:10.1007/978-3-642-32589-2_2.

27 Stefan Kratsch. A randomized polynomial kernelization for vertex cover with a smaller parameter. *SIAM J. Discret. Math.*, 32(3):1806–1839, 2018. doi:10.1137/16M1104585.

28 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. *J. ACM*, 67(3):16:1–16:50, 2020. doi:10.1145/3390887.

29 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Trans. Algorithms*, 11(2):15:1–15:31, 2014. doi:10.1145/2566616.

30 László Lovász and Michael D Plummer. *Matching theory*, volume 29. North-Holland, 1986.

31 Vadim V. Lozin. On maximum induced matchings in bipartite graphs. *Inf. Process. Lett.*, 81(1):7–11, 2002. doi:10.1016/S0020-0190(01)00185-5.

32 Jayakrishnan Madathil, Saket Saurabh, and Meirav Zehavi. Fixed-parameter tractable algorithm and polynomial kernel for max-cut above spanning tree. *Theory Comput. Syst.*, 64(1):62–100, 2020. doi:10.1007/s00224-018-09909-5.

33 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999. doi:10.1006/jagm.1998.0996.

34 Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *J. Comput. Syst. Sci.*, 75(2):137–153, 2009. doi:10.1016/j.jcss.2008.08.004.

35 Matthias Mnich, Geevarghese Philip, Saket Saurabh, and Ondrej Suchý. Beyond max-cut: λ-extendible properties parameterized above the poljak-turzík bound. *J. Comput. Syst. Sci.*, 80(7):1384–1403, 2014. doi:10.1016/j.jcss.2014.04.011.

36 Matthias Mnich and Rico Zenklusen. Bisections above tight lower bounds. In *Proceedings of the 38th International on Graph-Theoretic Concepts (WG 2012)*, pages 184–193, 2012. doi:10.1007/978-3-642-34611-8_20.

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

38 Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. *J. Discrete Algorithms*, 7(2):181–190, 2009. doi:10.1016/j.jda.2008.09.005.

39 N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. LP can be a cure for parameterized problems. In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, pages 338–349, 2012. doi:10.4230/LIPIcs.STACS.2012.338.

40 Rolf Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, (STACS 2010)*, pages 17–32, 2010. doi:10.4230/LIPIcs.STACS.2010.2495.

41 Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Paths, flowers and vertex cover. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA 2011)*, pages 382–393, 2011. doi:10.1007/978-3-642-23719-5_33.

42 Igor Razgon and Barry O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009. doi:10.1016/j.jcss.2009.04.002.

43 Larry J. Stockmeyer and Vijay V. Vazirani. NP-completeness of some generalizations of the maximum matching problem. *Inf. Process. Lett.*, 15(1):14–19, 1982. doi:10.1016/0020-0190(82)90077-1.

44 Mingyu Xiao and Shaowei Kou. Parameterized algorithms and kernels for almost induced matching. *Theor. Comput. Sci.*, 846:103–113, 2020. doi:10.1016/j.tcs.2020.09.026.

45 Michele Zito. Induced matchings in regular graphs and trees. In *Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1999)*, pages 89–100, 1999. doi:10.1007/3-540-46784-X_10.

A Missing Proofs from Section 3.2

▶ **Lemma 15.** *Reduction Rule 14 is correct.*

Proof. Let $G' = G - v$ be the graph obtained by deleting v. We show that $\mathsf{IM}(G) \geq \ell$ if and only if $\mathsf{IM}(G') \geq \ell$. First, observe that $\mathsf{IM}(G') \geq \ell$ implies $\mathsf{IM}(G) \geq \ell$, since any induced matching in G' is also an induced matching in G. We then show that if G has an induced matching M of size ℓ, then $\mathsf{IM}(G') \geq \ell$. If M does not cover v, then M is an induced matching in G' as well, implying that $\mathsf{IM}(G') \geq \ell$. Suppose that M covers v and let e denote the edge in M incident to v. Then, the edge uw is not part of M since u and w are adjacent to v. Thus, $(M \setminus \{e\}) \cup \{uw\}$ is an induced matching of size ℓ in G as well as G'. ◀

▶ **Lemma 16.** *Let $\mathcal{I} = (G, \ell)$ be an instance of IMBA and let $\mathcal{I}' = (G', \ell')$ be an instance obtained by applying Reduction Rule 6, 7, or 14. Then, $\mu(\mathcal{I}') \leq \mu(\mathcal{I})$.*

Proof. It is easy to see that when Reduction Rule 6 or 14 is applied, the measure does not increase as ℓ remains unchanged. There is no increase in the measure when applying Reduction Rule 7 either: We have $\mathsf{MM}(G') = \mathsf{MM}(G) - 1$ and $\mathsf{IS}(G') = \mathsf{IS}(G) - 1$ since every maximal matching contains an isolated edge and every maximal independent set contains exactly one endpoint of an isolated edge. Moreover, we have $\ell' = \ell - 1$. We thus have

$$\mu(\mathcal{I}') = \frac{1}{2}(\mathsf{MM}(G') + \mathsf{IS}(G')) - \ell' = \frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) - \ell = \mu(\mathcal{I}).$$ ◀

B Missing Proofs from Section 3.3

B.1 On Branching Rule 17

▶ **Branching Rule 17.** *Choose a vertex $v \in C$ with at least two neighbors $u, w \in N(v)$. We branch into three instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [3]$, where $G_1 = G - v$, $G_2 = G - u$, and $G_3 = G - w$.*

▶ **Lemma 18.** *Branching Rule 17 is correct.*

Proof. The correctness follows from the correctness of Branching Rule 9. ◀

▶ **Lemma 19.** *Let v be a vertex in $A \cup C$ and let $G' = G - v$ be the graph obtained by deleting v. Then, $\mathsf{MM}(G') \leq \mathsf{MM}(G) - 1$.*

Proof. By Theorem 4, every vertex in $A \cup C$ is matched in every maximum matching. Since G' misses one vertex from $A \cup C$, we have $\mathsf{MM}(G') \leq \mathsf{MM}(G) - 1$. ◀

▶ **Lemma 20.** *In Branching Rule 17, $\mu(\mathcal{I}_i) \leq \mu(\mathcal{I}) - \frac{1}{2}$ for all $i \in [3]$.*

Proof. By the definition of the Gallai–Edmonds decomposition, the neighbors of $v \in C$ are in $A \cup C$. It follows that $u, w \in A \cup C$. We thus have, by Lemma 19, that $\mathsf{MM}(G_i) \leq \mathsf{MM}(G) - 1$. We also have $\mathsf{IS}(G_i) \leq \mathsf{IS}(G)$. Consequently, $\mu(\mathcal{I}_i) = \frac{1}{2}(\mathsf{MM}(G_i) + \mathsf{IS}(G_i)) - \ell \leq \frac{1}{2}(\mathsf{MM}(G) - 1 + \mathsf{IS}(G)) - \ell = \mu(\mathcal{I}) - \frac{1}{2}$. ◀

B.2 On Branching Rule 21

▶ **Branching Rule 21.** *Choose two adjacent vertices $u, v \in A$. We branch into three instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [3]$. In the first two, we delete u or v, i.e., $G_1 = G - u$ and $G_2 = G - v$. In the third branch, we delete $N(\{u, v\})$, i.e., $G_3 = G - N(\{u, v\})$.*

▶ **Lemma 22.** *Branching Rule 21 is correct.*

Proof. By Theorem 4, $N(\{u, v\})$ has at least one vertex in D (note that Branching Rule 12 requires $N(\{u, v\}) \neq \emptyset$). The correctness of Branching Rule 21 follows from the correctness of Branching Rule 12. ◀

▶ **Lemma 43.** *In Branching Rule 21, $\mu(\mathcal{I}_i) \leq \mu(\mathcal{I}) - \frac{1}{2}$ for all $i \in [3]$.*

Proof. Since G_1 and G_2 each misses a vertex from A, we have $\mathsf{MM}(G_1) \leq \mathsf{MM}(G) - 1$ and $\mathsf{MM}(G_2) \leq \mathsf{MM}(G) - 1$ by Lemma 19. We show that $\mathsf{MM}(G_3) \leq \mathsf{MM}(G) - 1$ also holds. Any maximal matching M of G_3 contains uv, since it is isolated. Note, however, that u and v are matched to vertices in D in every maximum matching of G by Theorem 4. It follows that M is not a maximum matching in G, implying that $\mathsf{MM}(G_3) \leq \mathsf{MM}(G) - 1$. Since G_i ($i \in [3]$) arises from vertex deletions of G, we have $\mathsf{IS}(G_i) \leq \mathsf{IS}(G)$. Consequently, $\mu(\mathcal{I}_i) = \frac{1}{2}(\mathsf{MM}(G_i) + \mathsf{IS}(G_i)) - \ell \leq \frac{1}{2}(\mathsf{MM}(G) - 1 + \mathsf{IS}(G)) - \ell = \mu(G) - \frac{1}{2}$. ◀

B.3 On Branching Rule 25 and Branching Rule 28

▶ **Branching Rule 25.** *Let $S = \{u, v, w\}$ be a connected component of $G[D]$ such that $G[S]$ is a triangle and u and v have a neighbor u' and v' in A, respectively (such vertices exist by Lemma 24). Let $H_1 = G - u'$, $H_2 = G - u$, and $H_3 = G - v$. We generate seven instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [7]$, where $G_1 = H_1$, $G_2 = H_2 - v'$, $G_3 = H_2 - v$, $G_4 = H_2 - w$, $G_5 = H_3 - u'$, $G_6 = H_3 - u$, $G_7 = H_3 - w$.*

▶ **Branching Rule 28.** *Let S be a connected component of $G[D]$ such that $G[S]$ is a triangle star with at least two pendant triangles. Let $u, v \in S$ be vertices as specified in Lemma 27, and let $u', v' \in A$ be neighbors of u, v, respectively. Also, let $u'', v'' \in S$ be neighbors of u, v, respectively, which are not the center of $G[S]$. Let $H_1 = G - u'$, $H_2 = G - u$, and $H_3 = G - u''$. We generate seven instances $\mathcal{I}_i = (G_i, \ell)$ for $i \in [7]$, where $G_1 = H_1$, $G_2 = H_2 - v$, $G_3 = H_2 - v'$, $G_4 = H_2 - v''$, $G_5 = H_3 - v$, $G_6 = H_3 - v'$, $G_7 = H_3 - v''$.*

▶ **Lemma 24.** *Let S be a connected component of $G[D]$ such that $G[S]$ is a triangle. There exist two vertices $u, v \in S$ such that u and v have a neighbor u' and v' in A, respectively (possibly $u' = v'$).*

Proof. Let $S_1, \ldots, S_c$ be the connected components of $H - v$. Without loss of generality, assume that $|S_1| \geq \cdots \geq |S_c|$. We show that $|S_1| \geq 3$. Since H is factor-critical, $H - v$ has a perfect matching by definition. It follows that every connected component S_i has at least two vertices. For the sake of contradiction, assume that $|S_i| = 2$ for every $i \in [c]$. We claim that for every connected component $S_i = \{u_i, w_i\}$, v is adjacent to both u_i and w_i in H. If v is not adjacent to u_i (or w_i), then $G - w_i$ (or $G - v_i$, respectively) has no perfect matching

because u_i (or w_i, respectively) is isolated. Thus, v is adjacent to every vertex in H. This, however, implies that H is a triangle star, which contradicts our assumption. Thus, we have $|S_1| \geq 3$. Since $H[S_1]$ is connected, there is a vertex in S_1 of degree at least two in $H - v$. ◀

▶ **Lemma 26.** *Branching Rule 25 is correct.*

Proof. Observe that u is adjacent to u' and v in G. By the correctness of Branching Rule 9, $\mathcal{I}$ is a yes-instance if and only if one of the three instances $\mathcal{J}_j = (H_j, \ell)$ is a yes-instance for $j \in [3]$. Since v has two neighbors v', w in H_2, $\mathcal{J}_2$ is a yes-instance if and only if $\mathcal{I}_i$ is a yes-instance for some $i \in [2, 4]$ by the correctness of Branching Rule 9. Moreover, since u has two neighbors u', w in H_3, $\mathcal{J}_3$ is a yes-instance if and only if $\mathcal{I}_i$ is a yes-instance for some $i \in [5, 7]$ by the correctness of Branching Rule 9. Hence, $\mathcal{I}$ is a yes-instance if and only if one of the seven instances $\mathcal{I}_i = (G_i, \ell)$ is a yes-instance for some $i \in [7]$. ◀

▶ **Lemma 27.** *Let S be a connected component of $G[D]$ such that $G[S]$ is a triangle star with at least two pendant triangles. There exist two nonadjacent vertices $u, v \in S$ such that u and v have a neighbor u' and v' in A, respectively (possibly $u' = v'$).*

Proof. Let s be the center of $G[S]$, i.e., s has more than two neighbors in $G[S]$. Then, by the assumption that Reduction Rule 14 has been applied exhaustively, for every pendant triangle (w_1, w_2, s) in $G[S]$, at least one of w_1 or w_2 has at least one neighbor in A. Thus, the lemma holds. ◀

B.4 On Branching Rule 33

▶ **Lemma 31.** *Let H be a connected graph on at least four vertices that is not a star. Then, H has a path on four vertices.*

Proof. Let v be a vertex of maximum degree in H. We consider two cases: If $\deg(v) = |V(H)| - 1$, then by the assumption that H is not a star, we have two adjacent vertices $w, x \in N(v)$. Since $|N(v)| = |V(H)| - 1 \geq 3$, there is a vertex $u \in N(v)$ distinct from w and x. We thus have a path (u, v, w, x). Otherwise, we have $\deg(v) < |V(H)| - 1$. We can assume that $\deg(v) \geq 2$ because H is a connected graph on at least four vertices. Since H is connected, there exist two adjacent vertices $w \in N(v)$ and $x \in V(H) \setminus (N(v) \cup \{v\})$. Since $\deg(v) \geq 2$, there is a vertex $u \in N(v)$ distinct from w. We thus have a path (u, v, w, x). ◀

▶ **Lemma 32.** *Let H be a factor-critical graph that is not a triangle star and let v be an arbitrary vertex in $V(H)$. Then, $H - v$ has a vertex of degree at least two.*

Proof. Let $S_1, \ldots, S_c$ be the connected components of $H - v$. Without loss of generality, assume that $|S_1| \geq \cdots \geq |S_c|$. We show that $|S_1| \geq 3$. Since H is factor-critical, $H - v$ has a perfect matching by definition. It follows that every connected component S_i has at least two vertices. For the sake of contradiction, assume that $|S_i| = 2$ for every $i \in [c]$. We claim that for every connected component $S_i = \{u_i, w_i\}$, v is adjacent to both u_i and w_i in H. If v is not adjacent to u_i (or w_i), then $G - w_i$ (or $G - v_i$, respectively) has no perfect matching because u_i (or w_i, respectively) is isolated. Thus, v is adjacent to every vertex in H. This, however, implies that H is a triangle star, which contradicts our assumption. Thus, we have $|S_1| \geq 3$. Since $H[S_1]$ is connected, there is a vertex in S_1 of degree at least two in $H - v$. ◀

C Missing Proofs from Section 3.4

▶ **Lemma 39.** *For a connected graph G, if $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$, then one of the following holds:*

- *G is an isolated edge.*
- *G is a triangle star.*
- *G is obtained from a connected bipartite graph G' with a bipartition $V(G') = U \cup W$ by adding exactly one pendant vertex to each vertex of U and adding at least one pendant triangle to each vertex of W. In particular, $\mathsf{MM}(G) = \mathsf{IS}(G) = \mathsf{IM}(G) = \frac{1}{2}(|V(G)| - |W|)$.*

Proof. Suppose that G is a connected graph with $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$. We show that G satisfies one of the above. Since $\mathsf{MM}(G) = \mathsf{IS}(G) = \mathsf{IM}(G)$, G is a Cameron–Walker graph. We assume that G has at least two vertices, since otherwise $\mathsf{MM}(G) = \mathsf{IM}(G) = 0$ and $\mathsf{IS}(G) = 1$. We examine each case of Theorem 5.

- Suppose that G is a star with $n - 1$ pendant vertices. It is easy to see that $\mathsf{IS}(G) = n - 1$ and $\mathsf{IM}(G) = 1$. Thus, we obtain $n = 2$.
- The second case of Theorem 5 states that G is a triangle star. So we are done.
- Suppose that G is obtained from a bipartite graph G' with a bipartition $V(G') = U \cup W$ by adding at least one pendant vertex to each vertex of U and adding any number of pendant triangles to each vertex of W. Let n_u be the number of pendant vertices attached to u for every $u \in U$ and let n_w be the number of pendant triangles attached to w for every vertex $w \in W$. We show that $n_u = 1$ for each $u \in U$ and $n_w \geq 1$ for each $w \in W$. First, we show that $\mathsf{MM}(G) = \mathsf{IM}(G) = |U| + \sum_{w \in W} n_w$. Let M be a maximum (induced) matching of G. Since every vertex $u \in U$ has at least one pendant vertex adjacent to it, we can assume that u is matched to one of its pendant neighbors in M. The deletion of U and pendant vertices in $N(U)$ leaves $|W|$ triangle stars. The triangle star containing $w \in W$ has n_w pendant triangles. Thus, $|M| = |U| + \sum_{w \in W} n_w$.
  We then show that $\mathsf{IS}(G) = \sum_{u \in U} n_u + \sum_{w \in W} \max(n_w, 1)$. We can assume that a maximum independent set I contains all pendant vertices. Then, I contains no vertex of U. After deleting all pendant vertices and their neighbors (that is, U), a triangle star with n_w pendant triangles remains for each $w \in W$, which has an independent set of size $\max(n_w, 1)$. Thus, we have $\mathsf{IS}(G) = \sum_{u \in U} n_u + \sum_{w \in W} \max(n_w, 1)$.
  Since $n_u \geq 1$ and $\max(n_w, 1) \geq n_w$, we have $\mathsf{IS}(G) = \sum_{u \in U} n_u + \sum_{w \in W} \max(n_w, 1) \geq |U| + \sum_{w \in W} n_w = \mathsf{IM}(G)$. By the assumption that $\mathsf{IS}(G) = \mathsf{IM}(G)$, equality holds and hence $n_u = 1$ for each $u \in U$ and $\max(n_w, 1) = n_w$, that is, $n_w \geq 1$.

For the third case, note that since $|V(G)| = 2|U| + \sum_{w \in W}(2n_w + 1)$, we have $\mathsf{MM}(G) = \mathsf{IM}(G) = \mathsf{IS}(G) = |U| + \sum_{w \in W} n_w = \frac{1}{2}(|V(G)| - |W|)$. ◀

▶ **Lemma 40.** *Let $\mathcal{I} = (G, \ell)$ be an instance of IMBA with $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$. If $\mathcal{I}$ is a yes-instance, then the instance $\mathcal{I}' = (G', \ell')$ obtained from $\mathcal{I}$ by exhaustively applying Reduction Rules 6, 7, and 14 has $\ell' = 0$.*

Proof. Suppose that $\mathcal{I}$ is a yes-instance with $\frac{1}{2}(\mathsf{MM}(G) + \mathsf{IS}(G)) = \mathsf{IM}(G)$. Note that $\mathsf{IM}(G) \geq \ell$. We consider three cases of Lemma 39.

- Suppose that G consists of an isolated edge. Then, $\ell \leq 1$, and hence we have $\ell' = 0$ after an application of Reduction Rule 7.
- Suppose that G is a triangle star. Then, $\ell \leq \frac{1}{2}(|V(G)| - 1)$, and hence we have $\ell' = 0$ after one application of Reduction Rule 14 and $\frac{1}{2}(|V(G)| - 1)$ applications of Reduction Rule 7.

- Suppose that G arises from a connected bipartite graph with a bipartition $U \cup W$ as specified in Lemma 39. By Lemma 39, we have $\mathsf{IM}(G) = \frac{1}{2}(|V(G)| - |W|) \geq \ell$. Since every vertex $w \in W$ has at least one pendant triangle attached to it, Reduction Rule 14 deletes every vertex in W. Note that after the deletion of W, we have a disjoint union of $\frac{1}{2}(|V(G)| - |W|)$ isolated edges. It follows that Reduction Rule 7 applies ℓ times, resulting in an instance $\mathcal{I}' = (G', \ell')$ with $\ell' = 0$.

This concludes the proof. ◀

Finding and Counting Patterns in Sparse Graphs

Balagopal Komarath
IIT Gandhinagar, India

Anant Kumar
IIT Gandhinagar, India

Suchismita Mishra
Universidad Andrés Bello, Santiago, Chile

Aditi Sethia
IIT Gandhinagar, India

Abstract

We consider algorithms for finding and counting small, fixed graphs in sparse host graphs. In the non-sparse setting, the parameters treedepth and treewidth play a crucial role in fast, constant-space and polynomial-space algorithms respectively. We discover two new parameters that we call matched treedepth and matched treewidth. We show that finding and counting patterns with low matched treedepth and low matched treewidth can be done asymptotically faster than the existing algorithms when the host graphs are sparse for many patterns. As an application to finding and counting fixed-size patterns, we discover $\widetilde{O}(m^3)$-time[1], constant-space algorithms for cycles of length at most 11 and $\widetilde{O}(m^2)$-time, polynomial-space algorithms for paths of length at most 10.

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

Keywords and phrases Subgraph Detection and Counting, Homomorphism Polynomials, Treewidth and Treedepth, Matchings

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.40

Related Version *Full Version*: `http://bkomarath.rbgo.in/papers/KKMS23.pdf`

Supplementary Material *Dataset (Graphs)*: `https://github.com/anonymous1203/Spasm`

Acknowledgements The research work of S. Mishra is partially funded by Fondecyt Postdoctoral grant 3220618 of Agencia National de Investigatión y Desarrollo (ANID), Chile.

1 Introduction

Given simple graphs G, called the *pattern*, and H, called the *host*, a fundamental computational problem is to find or count occurrences of G in H. What does it mean for G to occur in H? The three most common notions of occurrence are characterized by mappings $\phi : V(G) \mapsto V(H)$. We say:

1. If $\{u, v\} \in E(G)$ implies $\{\phi(u), \phi(v)\} \in E(H)$ and ϕ is one-to-one, then we say that ϕ witnesses a subgraph isomorphic to G in H. The subgraph is obtained by taking the vertices and edges in the image of ϕ. The number of G-subgraphs of H is just the number of such subgraphs G' of H.
2. If $\{u, v\} \in E(G)$ is equivalent to $\{\phi(u), \phi(v)\} \in E(H)$ and ϕ is one-to-one, then ϕ witnesses an induced subgraph isomorphic to G in H. The induced subgraph is obtained by taking the vertices and *all* edges induced by those vertices in the image of ϕ.

[1] $\widetilde{O}$ hides factors that are logarithmic in the input size.

© Balagopal Komarath, Anant Kumar, Suchismita Mishra, and Aditi Sethia;
licensed under Creative Commons License CC-BY 4.0

3. If $\{u, v\} \in E(G)$ implies $\{\phi(u), \phi(v)\} \in E(H)$, then we say that ϕ is a homomorphism from G to H. Note that unlike a subgraph isomorphism, ϕ is not required to be one-to-one.

For any of these notions, the detection problem is clearly in NP. All three of them are also straightforward generalizations of the NP-hard problem CLIQUE. Therefore, the existence of efficient algorithms for finding or counting patterns under any of these notions is unlikely in general.

The class of pattern detection and counting problems remain interesting even if we restrict our attention to fixed pattern graphs. Williams [20] showed that the improved algorithms for finding triangles could be used to find faster algorithms for even NP-complete problems such as MAX2SAT. For fixed pattern graphs of size k, the brute-force search algorithm is as follows: Iterate over all k-tuples over $V(H)$ and check whether G occurs in the induced subgraph of H on the vertices in that k-tuple. This algorithm takes $\theta(n^k)$ time and constant space. Therefore, when we restrict our attention to fixed patterns, we seek improvements over this running time preferably keeping the space usage low. There are two broad techniques that reduce the running-time: the usage of fast matrix multiplication algorithms as a sub-routine and the exploitation of structural properties of pattern graphs.

If A is the adjacency matrix of the graph, then Nešetřil and Poljak [16] showed that one can obtain an $O(n^\omega)$-time algorithm for counting triangles using the identity $\mathsf{trace}(A^3) = 6\Delta$, where Δ is the number of triangles in the graph, where $\omega < 2.38$ is the matrix multiplication exponent. Using a simple reduction, they extended this to an algorithm to count $3k$-cliques in $O(n^{k\omega})$-time. They also showed that we can use improved algorithms for counting k-cliques to count any k-vertex pattern. Later, Kloks, Kratsch and Müller [12] showed how to use fast rectangular matrix multiplication to obtain similar improvements to the running time for counting cliques of all sizes, not just multiples of three. Note that the improvements obtained by these algorithms are applicable to all k-vertex patterns. i.e., they do not use the pattern's structure to obtain better algorithms. Since finding a k-clique requires $n^{\Omega(k)}$-time unless ETH is false, we need to exploit the structure of the pattern to obtain significantly better algorithms.

For patterns sparser than cliques, the run-time can be significantly improved over even fast matrix multiplication based (pattern finding) algorithms. The crucial idea is to exploit the structure of the pattern graph. A k-*walk polynomial* is a polynomial where the monomials correspond to walks that are k vertices long. For example, a walk (u, v, w, x) will correspond to the monomial $x_{uv}x_{vw}x_{wx}$ and a walk (u, v, u, v) to the monomial x_{uv}^3 [2]. Williams [21] showed that we can detect k-paths in graphs by (1) computing the k-walk polynomial and (2) checking whether it has multilinear monomials. We can compute the k-walk polynomial in linear-time using a simple dynamic programming algorithm and then multilinear monomials can be detected with high probability by evaluating this polynomial over an appropriate ring where the randomly chosen elements satisfy $a^2 = 0$. This yields is a $O(2^k(n + m))$-time algorithm for finding k-vertex paths as subgraphs in n-vertex, m-edge host graphs.

We now consider the problem of counting sparse patterns. For counting k-paths as subgraphs, the best known algorithm by Curticapean, Dell and Marx [4] takes only $O(f(k)n^{0.174k+o(k)})$-time for some function f. Coming to fixed pattern graphs, Alon, Yuster and Zwick [1] gave $O(n^\omega)$-time algorithms for counting cycle subgraphs of length at most 8 using an algorithm that *combines* fast matrix multiplication and exploitation of the structure of the pattern. Notice that this is the same as the time required for counting triangles (3-cliques).

[2] We write uv to denote the edge $\{u, v\}$.

The notion of graph homomorphisms was shown to play a crucial role in all the above improved algorithms for finding and counting non-clique subgraphs. More specifically, Fomin, Lokshtanov, Raman, Rao, and Saurabh [11] showed how the efficient construction of homomorphism polynomials (see Definition 17), a generalization of k-walk polynomials, can be used to detect subgraphs with small treewidth efficiently. Their algorithm can be seen as a generalization of Williams's algorithm [21] for k-paths to arbitrary graphs. Similarly, Curticapean, Dell and Marx [4] showed that efficient algorithms for counting subgraphs can be derived from efficient algorithms for counting homomorphisms of graphs of small treewidth. Their algorithm can be seen as a generalization of the cycle-counting algorithms of Alon, Yuster, and Zwick [1].

Algorithms for finding and counting patterns in *sparse* host graphs are also studied. An additional parameter, m, the number of edges in the host graph, is taken into account for the design and analysis of these algorithms. In the worst-case, m could be as high as $\binom{n}{2}$, and hence, an $O(n^t)$-time algorithm and an $O(m^{t/2})$-time algorithm for some t have the same asymptotic time complexity. However, it is common in practice that $m = o(n^2)$. For example, if the host graph models a road network, then $m = O(n)$, where the constant factor is determined by the maximum number of roads at any intersection. In such cases, an $O(m^{t/2})$-time algorithm is asymptotically better than an $O(n^t)$-time algorithm.

The broad themes of using fast matrix multiplication and/or structural parameters of the pattern to obtain improved algorithms are still applicable in the setting of sparse host graphs. Using fast matrix multiplication, Eisenbrand and Grandoni [8] showed that we can count k-cliques in $O(m^{k\omega/6})$-time. Kloks, Kratsch and Müller [12] showed that K_4 subgraphs can be counted in $O(m^{(\omega+1)/2})$-time. Again, since $\omega < 3$, this is better than the $O(m^2)$-time given by the brute-force algorithm. Using structural parameters of the pattern, Kowaluk, Lingas, and Lundell [15] obtained many improved algorithms in the sparse host graph setting. For example, their methods obtain an algorithm that runs in $O(m^4)$-time for counting P_{10} as subgraphs. In this work, we obtain an $\widetilde{O}(m^2)$-time algorithm for counting P_{10} (See Theorems 12,13,14,15 for similar improvements).

The model of computation that we consider is the unit-cost RAM model. In particular, we can store labels of vertices and edges in the host graph in a constant number of words[3]. In this model, algorithms based on fast matrix multiplication and/or treewidth mentioned above use polynomial space. However, the brute-force search algorithm uses only constant space as it only needs to store k vertex labels at a time (Recall that we regard k as a constant.). How much speed-up can we obtain while preserving constant space usage? The graph parameter *treedepth* plays a crucial role in answering this question. It is well known that we can count the homomorphisms from a pattern of treedepth d in $O(n^d)$-time while using only constant space (See Komarath, Rahul, and Pandey [13] for a construction of arithmetic formulas counting them. These arithmetic formulas can be implicitly constructed and evaluated in constant space.). Since all k-vertex patterns except k-clique has treedepth strictly less than k, this immediately yields an improvement over the running-time of brute-force while preserving constant space usage. In this work, we improve upon the treedepth-based algorithms for sparse host graphs where the pattern graph is a cycle of length at most 11 (See Theorem 2).

[3] In the TM model or the log-cost RAM model, storing labels of vertices would take $O(\log n)$ space.

1.1 Connection to arithmetic circuits for graph homomorphism polynomials

A popular sub-routine in these algorithms is an algorithm by Diaz, Serna and Thilikos [6] that efficiently counts the number of homomorphisms from a pattern of small treewidth to an arbitrary host graph. Indeed, it can be shown that this algorithm can be easily generalized to *efficiently construct* circuits for homomorphism polynomials instead of counting homomorphisms. Bläser, Komarath and Sreenivasaiah [2] showed that efficient constructions for homomorphism polynomials can even be used to detect *induced* subgraphs in some cases. They also show that many of the faster induced subgraph detection algorithms, such finding four-node subgraphs by Williams et al. [22] and five-node subgraphs by Kowaluk, Lingas, and Lundell [15] can be described as algorithms that efficiently construct these homomorphism polynomials. Therefore, arithmetic circuits for graph homomorphism polynomials provide a unifying framework for describing almost all the fast algorithms that we know for finding and counting subgraphs and finding induced subgraphs. Can we improve these algorithms by finding more efficient ways to construct arithmetic circuits for homomorphism polynomials? Unfortunately, it is known that for the type of circuit that is constructed, i.e., circuits that do not involve cancellations, the existing constructions are the best possible for *all* pattern graphs, as shown by Komarath, Pandey, and Rahul [13]. The situation is similar for constant space algorithms. The best known algorithms can be expressed as divide-and-conquer algorithms that evaluate small formulas constructed by making use of the graph parameter treedepth. Komarath, Pandey, and Rahul[13] also showed that the running-time of these algorithms match the best possible formula size for *all* pattern graphs. These arithmetic circuit lower bounds serve as a technical motivation for considering sparse host graphs, in addition to the practical motivation mentioned earlier.

1.2 Our findings

In this paper, we study algorithms for finding and counting patterns in host graphs that work well especially when the host is sparse. We discover algorithms that are (1) strictly better than the brute-force algorithm, (2) strictly better than the best-known algorithms when the host graph is sparse, (3) close to the best-known algorithms when the host graphs are dense. Our algorithms are based on two new structural graph parameters – the *matched treedepth* and *matched treewidth*. (See 19 and 21 for formal definitions). We show that they can be used to obtain improved running times for algorithms that use constant space and polynomial space respectively. Our algorithms are summarized in Table 1. In the table, the parameter m is the number of edges in the host graph. We denote using mtw the matched treewidth of the pattern and using mtd the matched treedepth of the pattern. The notation $\widetilde{\mathsf{O}}$ hides factors that are poly-logarithmic in the input (the host graph) size.

We now explain the relevance of our new parameters; state our algorithms, the relationships between various graph parameters, and some structural characterizations that we prove in this paper in the rest of this section.

Treedepth and matched treedepth

The matched treedepth of a graph is closely related to its treedepth. The constant space algorithm based on treedepth is essentially an divide-and-conquer algorithm over a elimination tree of the pattern graph that executes a brute-force search over each root-to-leaf path in the elimination tree. Therefore, it runs in time $\mathsf{O}(\mathsf{n}^{\mathsf{d}})$. We exploit the fact that the elimination tree is matched, which forces an additional constraint that the vertices in each root-to-leaf

Table 1 Pattern counting and detection algorithms for sparse host graphs.

Pattern	Type	Problem	Time	Space	Remarks
C_k	Subgraph	Counting	$\widetilde{O}(m^3)$	$O(1)$	$k \leqslant 11$
P_k	Subgraph	Counting	$\widetilde{O}(m^2)$	$\widetilde{O}(m^2)$	$k \leqslant 10$
C_k	Subgraph	Counting	$\widetilde{O}(m^2)$	$\widetilde{O}(m^2)$	$k \leqslant 9$
Any	Homomorphism	Counting	$\widetilde{O}(m^{\lceil \mathsf{mtd}/2 \rceil})$	$O(1)$	
Any	Homomorphism	Counting	$\widetilde{O}(m^{\lceil (\mathsf{mtw}+1)/2 \rceil})$	$\widetilde{O}(m^{\lceil (\mathsf{mtw}+1)/2 \rceil})$	
C_6	Induced subgraph	Detection	$\widetilde{O}(m^2)$	$\widetilde{O}(m^2)$	
$\overline{P_k}$	Induced subgraph	Detection	$\widetilde{O}(m^{\lceil (k-2)/2 \rceil})$	$\widetilde{O}(m^{\lceil (k-2)/2 \rceil})$	

path has to be covered by a matching. This allows the brute-force part of the algorithm to discover all d vertices on the path using only $d/2$ edges. The central algorithm that we use to obtain constant space algorithms is given below:

▶ **Theorem 1.** *Let G be a graph with $\mathsf{mtd}(G) = d$, then given an m-edge graph H as input, we can count the number of homomorphisms from G to H in $\widetilde{O}(m^{\lceil d/2 \rceil})$-time and constant space.*

It is well-known that the number of G-subgraphs, for any G, can be expressed as a linear combination of the number of homomorphisms from a related set of graphs called the *spasm of* G. The spasm of G contains exactly all graphs that can be obtained by iteratively merging the independent sets in G. Although the treedepth of the spasm of C_{11} is bounded by 6, however, the matched treedepth is not necessarily bounded by the treedepth. We analyze all graphs in the spasm of C_{10} and C_{11} (there are 501 such graphs) and show that the matched treedepth of each graph is at most 6. This yields the following algorithm:

▶ **Theorem 2.** *Given an m-edge graph H as input, we can count the number of C_k, where $k \leqslant 11$, as subgraphs in $\widetilde{O}(m^3)$-time and constant space.*

For comparison, the brute-force algorithm takes $O(m^6)$-time and constant space; and the treedepth based algorithm takes $O(n^6)$-time and constant space.

As seen from the proof of our algorithm for counting C_{11}, the spasm of a pattern can contain a large number of graphs even for relatively small patterns. Therefore, it would be nice to have theorems that upper-bound the matched treedepth. Unfortunately, the property $\mathsf{mtd}(G) \leqslant k$ is not even subgraph-closed unlike treedepth. For example, it can be proved that $\mathsf{mtd}(K_4 - e) = 3$ but $\mathsf{mtd}(C_4) = 4$. However, interesting structural observations can still be made for matched treedepth. The following is a theorem that upper-bounds matched treedepth in terms of treedepth.

▶ **Theorem 3.** *For any graph G, $\mathsf{mtd}(G) \leqslant 2 \cdot \mathsf{td}(G) - 2$.*

Theorem 3 implies that our constant-space algorithms from Theorem 1 for counting homomorphisms are asymptotically faster for *all* patterns, where the inputs are sparse host graphs, when compared to the treedepth-based algorithm.

The following theorem shows that the time complexity for counting homomorphisms of a pattern is lower-bounded by the time complexity for counting all of its induced subgraphs.

▶ **Theorem 4.** *Let G be a graph and G' is a connected, induced subgraph of G, then:*
1. $\mathsf{mtd}(G') \leqslant \mathsf{mtd}(G)$ *if* $\mathsf{mtd}(G)$ *is even.*
2. $\mathsf{mtd}(G') \leqslant \mathsf{mtd}(G) + 1$ *if* $\mathsf{mtd}(G)$ *is odd.*

In light of the importance of matched treedepth, it becomes crucial that we understand this structural parameter as much as possible. The graphs of treedepth 2 are exactly the class of star graphs. This is also the class of graphs with matched treedepth 2. However, for the graph C_4, we have $\mathsf{td}(\mathsf{C}_4) = 3$ and $\mathsf{mtd}(\mathsf{C}_4) = 4$. So it is interesting to know what are exactly the graphs where treedepth and matched treedepth coincide. The following theorem should be viewed as giving us a preliminary understanding of the relationship between these two parameters.

▶ **Theorem 5.** *Let* G *be a graph such that* $\mathsf{td}(\mathsf{G}) = 3$*. Then* $\mathsf{mtd}(\mathsf{G}) = 3$ *if and only if* G *is* $(\mathsf{C}_4, \mathsf{P}_6, \mathsf{T}_{3,3})$*-free.*

The graph $\mathsf{T}_{3,3}$ is the $(3, 3)$ tadpole graph.

Treewidth and matched treewidth

The treewidth-based dynamic programming algorithm of Díaz, Serna, and Thilikos [6] can be strengthened to output an *arithmetic circuit* that computes the *homomorphism polynomial* for the pattern. An arithmetic circuit is a directed acyclic graph where each internal node is labeled $+$ or $\times$, each leaf is labeled by a variable or a field constant, and there is a designated output node. Such a graph computes a polynomial over the underlying field in a natural fashion. We find that by using a dynamic programming algorithm over matched tree decompositions, we can improve the size of the arithmetic circuit for *sparse* host graphs. Our central theorem is given below:

▶ **Theorem 6.** *Let* G *be a graph with* $\mathsf{mtw}(\mathsf{G}) = \mathsf{t}$*, then given an* $\mathfrak{m}$*-edge host graph* H *as input, we can construct an arithmetic circuit computing the homomorphism polynomial from* G *to* H *in time* $\widetilde{\mathsf{O}}(\mathfrak{m}^{\lceil (\mathsf{t}+1)/2 \rceil})$.

For graphs where matched treewidth and treewidth coincide, the running time for counting homomorphisms is a quadratic improvement on the algorithm by Díaz, Serna, and Thilikos [6] for sparse graphs. Therefore, this is also the best possible improvement one can hope to get without improving upon the algorithm by Díaz, Serna and Thilikos [6]. What is the worst case? The following theorem implies that the resulting algorithm *cannot* be worse on sparse host graphs.

▶ **Theorem 7.** *For any graph* G*, we have* $\mathsf{mtw}(\mathsf{G}) \leqslant 2 \cdot \mathsf{tw}(\mathsf{G}) + 1$.

Unfortunately, unlike for treewidth, the parameter $\mathsf{mtw}(\mathsf{G})$ is not monotone over the subgraph partial order. We first observe an explicit graph family with lower tw and larger mtw. Consider the complete bipartite graph $\mathsf{K}_{\mathfrak{n},\mathfrak{n}}$ on $\mathfrak{n}$ vertices. Notice that $\mathsf{tw}(\mathsf{K}_{\mathfrak{n},\mathfrak{n}}) = \mathfrak{n}$.

▶ **Proposition 8.** $\mathsf{mtw}(\mathsf{K}_{\mathfrak{n},\mathfrak{n}}) = 2\mathfrak{n} - 2$ *for all* $\mathfrak{n} > 1$.

The following observation shows that there exists supergraphs of $\mathsf{K}_{\mathfrak{n},\mathfrak{n}}$ with lower mtw than that of $\mathsf{K}_{\mathfrak{n},\mathfrak{n}}$.

▶ **Observation 9.** *Consider the supergraph* G *of* $\mathsf{K}_{\mathfrak{n},\mathfrak{n}}$ *such that* $\mathsf{V}(\mathsf{G}) = \mathsf{V}(\mathsf{H})$*, and there are edges in one partition of* $\mathsf{K}_{\mathfrak{n},\mathfrak{n}}$ *such that the independent set of size* $\mathfrak{n}$ *becomes a path on* $\mathfrak{n}$ *vertices. Note that although* $\mathsf{mtw}(\mathsf{K}_{\mathfrak{n},\mathfrak{n}}) = 2\mathfrak{n} - 2$*, but* $\mathsf{mtw}(\mathsf{G}) = \mathfrak{n}$.

We show how we can use structural theorems about matched treewidth to prove algorithmic upper bounds. For example, to count the number of P_{10} subgraphs, we only have to show that all graphs in the spasm of P_{10} have low matched treewidth. The spasm of P_{10} is a large set that contains more than 300 graphs. Indeed, it is possible to analyze the matched treewidth for each of these graphs individually. However, it would be better if we have theorems that eliminate such tedious work.

We derive some structural theorems for low values of matched treewidth. Graphs with matched treewidth 1 are exactly trees. We also show $\mathsf{tw}(\mathsf{C}_5) = 2$ and $\mathsf{mtw}(\mathsf{C}_5) = 3$.

We characterize the matched treewidth of partial 2-trees using forbidden induced minors (See Definition 24) wherever possible. We show that C_5 is exactly the obstruction that forces higher matched treewidth for partial 2-trees.

▶ **Theorem 10.** *For any partial 2-tree* G*, the graph* G *is* C_5*-induced-minor-free if and only if* $\mathsf{mtw}(G) = 2$*.*

Notice that $\mathsf{tw}(G) = 2$ yields $O(n^3)$-time algorithms for counting homomorphisms. Even if $\mathsf{mtw}(G) = 3$, we obtain $\widetilde{O}(m^2)$-time algorithms for counting homomorphisms which is an improvement for sparse graphs. Does all treewidth 2 graphs have matched treewidth at most 3? No. The graph X in Figure 1 has treewidth 2 and matched treewidth 4. In fact, we can prove that X is exactly the obstruction that forces treewidth 2 graphs to have matched treewidth 4.

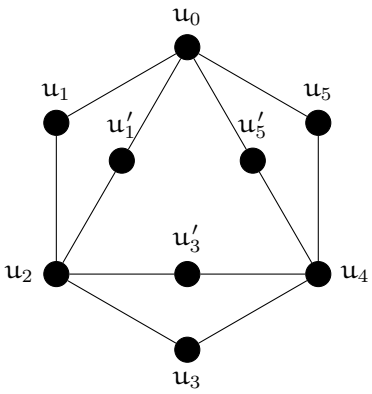


Figure 1 The graph X.

▶ **Theorem 11.** *For any partial 2-tree* G*, the graph* G *is* X*-induced-minor-free if and only if* $\mathsf{mtw}(G) \leqslant 3$*.*

This theorem implies that all X-induced-minor-free, treewidth 2 patterns have $\widetilde{O}(m^2)$-time homomorphism counting algorithms. This is an improvement for sparse host graph even over the fast matrix multiplication based algorithm given by Curticapean, Dell, and Marx [9] for counting homomorphisms from treewidth 2 graphs that runs in $O(n^\omega)$-time. Since the spasm of P_{10} does not contain any treewidth 4 graph or graph with an X-induced minor, we can show that there is an $\widetilde{O}(m^2)$-time algorithm for counting subgraph isormophisms of all paths on at most 10 vertices by showing that all treewidth 3 graphs in the spasm of P_{10} has matched treewidth 3. There are only 18 such graphs. Analyzing their matched treewidth yields the following theorem:

▶ **Theorem 12.** *Given an* m*-edge graph* H *as input, we can count the number of* P_k *subgraphs, where* $k \leqslant 10$*, in* $\widetilde{O}(m^2)$*-time.*

To the best of our knowledge, the best known path counting algorithms take $\Omega(n^4)$ time for paths on 10 vertices. Therefore, our algorithm is a significant improvement for sparse host graphs and no worse than the best known algorithm for dense host graphs. An easy corollary of the proof of this result is given below:

▶ **Theorem 13.** *Given an* m*-edge graph* H *as input, we can count the number of cycles of length at most* 9 *in* $\widetilde{O}(m^2)$*-time.*

These cycle counting algorithms are an improvement on sparse graphs over the $O(n^\omega)$-time algorithms for cycles of length at most 8 given by Alon, Yuster and Zwick [1].

We also show how to use our improved homomorphism polynomial construction algorithm to speed up detection of induced subgraphs. In particular, we show the following:

▶ **Theorem 14.** *Given an m-edge host graph as input, we can find an induced C_6 or report that none exists in $\widetilde{O}(m^2)$-time.*

This algorithm is no worse than the $O(n^4)$ time algorithm that can be derived using the techniques by Bläser, Komarath, and Sreenivasaiah [2]. For sparse graphs, our algorithm provides a quadratic improvement. We also show the following:

▶ **Theorem 15.** *Given an m-edge host graph as input, we can find an induced $\overline{P_k}$ or report that none exists in $\widetilde{O}(m^{(k-2)/2})$-time.*

This is also a quadratic improvement over the $O(n^{k-2})$ time algorithm given by Bläser, Komarath, and Sreenivasaiah [2] when the host graph is sparse. These algorithms are obtained by analyzing the matched treewidth *and* the automorphism structure of a set of graphs derived from the pattern.

From a technical standpoint, we see our algorithms as a natural combination of pattern detection and counting algorithms that work well on sparse host graph such as the $\widetilde{O}(m)$ algorithm for counting k-walks, the $\widetilde{O}(m^{k/2})$ algorithm for counting k-cliques, and $\widetilde{O}(m^{(k-1)/2})$ time algorithm for detecting induced $K_k - e$ by Vassilevska [18] and insights that improve the running-time on dense graphs by exploiting structural parameters treedepth and treewidth. We do not make use of fast matrix multiplication in any of our algorithms. Such algorithms, called *combinatorial algorithms*, are also of general interest to the community.

1.3 Related work

Algorithms for counting *induced subgraphs* are related to the problems that we consider but we do not consider any algorithms for it. This problem seems to be much harder. It is conjectured by Floderus, Kowaluk, Lingas, Lundell [9] that counting induced subgraphs for any k-vertex pattern graph is as hard as counting k-cliques for sufficiently large k. Several works have considered the parameterized complexity of counting subgraphs (See [5, 4, 17, 10, 7]) where the primary goal is to obtain a dichotomy of easy vs hard based on structural graph parameters. Some works have also considered restrictions on host graphs such as d-degeneracy [3]. The papers on parameterized complexity primarily chooses to focus on the growth-rate of the exponent for a family of patterns such as k-paths, k-cycles, or k-cliques and not the exact exponent for small graphs as we do in this paper.

2 Preliminaries

We consider simple graphs. We refer the reader to Douglas West's textbook [19] for basic definitions in graph theory. We use the following common notations for some well-known graphs: P_k for k-vertex paths, C_k for k-cycles, K_k for k-cliques, $K_k - e$ for k-clique with one edge missing, $K_{m,n}$ for complete bipartite graphs. A k-star is a $(k+1)$-vertex graph with a vertex u adjacent to vertices $v_1, \ldots, v_k$ and no other edges. A *star graph* is a k-star for some k. For a graph G and $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G induced by the vertices in S.

▶ **Definition 16.** *Given two graphs G and H, a graph homomorphism from G to H is a map $\phi : V(G) \to V(H)$ such that if $uv \in E(G)$, then $\phi(u)\phi(v) \in E(H)$.*

We denote by $\mathrm{Hom}(G, H)$, the set of all homomorphisms from G to H.

▶ **Definition 17.** *Given two graphs* G *and* H*, a homomorphism polynomial is an associated polynomial* $\mathsf{Hom}_G[H]$ *such that there is a one-to-one correspondence between its monomials and the homomorphisms from* G *to* H*. We define:*

$$\mathsf{Hom}_G[H] = \sum_{\phi \in \mathrm{Hom}(G,H)} \prod_{u \in V(G)} y_{\phi(u)} \prod_{\{u,v\} \in E(G)} x_{\{\phi(u),\phi(v)\}}$$

Note that H *has a subgraph isomorphic to* G *if and only if* P_G *has a multilinear monomial.*

We say that a graph G' is $(G_1, \ldots, G_m)$-free if no induced subgraph of G' is isomorphic to G_i for any i. We denote the complement of a graph G by $\overline{G}$. We have $V(G) = V(\overline{G})$ and the edges of $\overline{G}$ are exactly the non-edges of G and vice versa.

We assume that all pattern graphs are connected. Since our primary algorithms are all based on counting homomorphisms, this does not lose generality as the number of homomorphisms from a disconnected pattern is just the product of the number of homomorphisms from its components.

▶ **Definition 18.** *An* elimination tree (T, r) *of a connected graph* G *is a tree rooted at* $r \in V(G)$*, where the sub-trees of* r *are recursively elimination trees of the connected components of the graph* $G \setminus r$*. The elimination tree of an empty (no vertices or edges) graph is the empty tree. The depth of an elimination tree* (T, r) *is defined as the maximum number of vertices over all root-to-leaf paths in* T*. The* treedepth *of a graph* G*, denoted* $\mathsf{td}(G)$*, is the minimum depth among all possible elimination trees of* G*.*

Intuitively, treedepth measures the closeness of a given graph to star graphs which are exactly the connected graphs having treedepth 2. We introduce a related notion called matched treedepth that seems to be helpful when designing algorithms for finding or counting patterns in sparse host graphs.

▶ **Definition 19.** *A* matched elimination tree *for a graph* G *is an elimination tree such that the following conditions are true for any root-to-leaf path* $(v_1, \ldots, v_k)$*:*

- *If* k *is even, then* $v_1v_2, v_3v_4, \ldots, v_{k-1}v_k$ *is a matching in* G*.*
- *If* k *is odd, then there is some* i *such that* $E' = \{v_1v_2, \ldots, v_{i-1}v_i, v_iv_{i+1}, \ldots, v_{k-1}v_k\}$ *and* $E' \subseteq E(G)$*. We have that* $E' \setminus \{v_{i-1}v_i, v_iv_{i+1}\}$ *is a matching on* $(k-3)/2$ *vertices.*

The matched treedepth *of a graph* G*, denoted* $\mathsf{mtd}(G)$*, is the minimum depth among all possible matched elimination trees of* G*.*

The matched treedepth is always finite.

▶ **Definition 20.** *A* tree decomposition *of a graph* G *is a pair* $(T, B(t)_{t \in T})$ *where* T *is a tree and* $B(t)$*, called a* bag*, is a collection of subset of vertices of* G *corresponding to every node* $t \in T$*.*

- *(Connectivity Property) For all* $v \in V(G)$*, there is a node* $t \in V(T)$ *such that* $v \in B(t)$ *and all such nodes* t *that contain* v *form a connected component in* T*.*
- *(Edge Property) For all* $e \in E(G)$*, there is a node* $t \in V(T)$ *such that* $e \subseteq B(t)$*.*

The width of a tree decompostion (T, B) *is defined as the maximum bag size minus one, that is,* $\max_{t \in T} |B(t) - 1|$*. The* treewidth *of a graph* G*,* $\mathsf{tw}(G)$*, is the minimum possible width among all possible tree decompositions of* G*.*

Intuitively, treewidth measures the closeness of the given graph to trees which are exactly the graphs with treewidth 1. We introduce a related notion, called matched treewidth, closely related to treewidth, that seems to be useful for designing dynamic programming algorithms over sparse host graphs.

▶ **Definition 21.** *A* matched tree decomposition *for a graph* G *is a tree decomposition where for every bag in the tree decomposition, the subgraph of* G *induced by the vertices in that bag has either a perfect matching or a matching where exactly one vertex* v *in the bag is unmatched and* v *is adjacent to some vertex in the matching. We call such bags* matched*. The* matched treewidth *of a graph* G*,* $\mathrm{mtw}(G)$*, is the minimum possible width among all possible matched tree decompositions of* G*.*

Matched treewidth is finite for all graphs. This is not trivial unlike treewidth because a single bag tree decomposition that contains all the vertices in the graph need not be matched.

We call a tree decomposition *reduced* if no bag B is a subset of another bag. Given any tree decomposition T, we can obtain a reduced tree decomposition T' such that the width of T' is at most the width of T. Moreover, all bags in T' are also bags in T. This implies that if T is matched, then T' is matched as well.

There are several equivalent characterizations for treewidth. Below, we state the ones that we use in this paper.

▶ **Definition 22.** *A* k-tree *is a graph formed by starting with a* $(k+1)$*-clique and repeatedly adding a vertex connected to exactly* k *vertices of the existing* $(k+1)$*-clique. A* partial *of a graph* G *is a graph obtained by deleting edges from* G*. The set of all graphs with treewidth at most* k *is exactly the class of partial* k*-trees.*

We can construct a *standard tree decomposition* T for any k-tree as follows: The root bag of T contains the vertices in the initial $(k+1)$-clique. Let S be a k-sized subset of this clique such that a new vertex v is added to the k-tree by connecting it to all vertices in S. Then, we add a sub-tree to T that will be a standard tree decomposition of the k-tree constructed using $S \cup \{v\}$ as the starting $(k+1)$-clique.

A *chordal completion* of a graph G is a super-graph G' of G such that G' has no induced cycles of length more than 3. A chordal completion that minimizes the size of the largest clique is called *minimum chordal completion*. The treewidth of a graph G is the size of the largest clique in its minimum chordal completion.

Two paths P_1 and P_2 from u to v are *internally disjoint* if and only if P_1 and P_2 do not have any common internal vertex.

A graph G is *2-connected* or *biconnected*, if for any $x \in V(G)$, $G - x$ is connected. Equivalently, for any two vertices in G, there are at least 2 internally disjoint paths in G.

▶ **Definition 23.** *A series-parallel graph is a triple* (G, s, t) *where* s *and* t *are vertices in* G*. This class is recursively defined as follows:*

- *An edge* $\{s, t\}$ *is a series-parallel graph.*
- *(series composition) If* (G_1, s_1, t_1) *and* (G_2, s_2, t_2) *are series-parallel graphs, then the graph obtained by identifiying* s_2 *with* t_1 *is also series-parallel.*
- *(parallel composition) If* (G_1, s_1, t_1) *and* (G_2, s_2, t_2) *are series-parallel graphs, then the graph obtained by identifiying* s_1 *with* s_2 *and* t_1 *with* t_2 *is also series-parallel.*

A graph has treewidth at most 2 *is if and only if all of its biconnected components are series-parallel graphs.*

▶ **Definition 24.** *A graph* G *is said to be a* minor *of a graph* G' *if* G *can be obtained from* G' *either by deleting edges/vertices, or by contracting the edges. (The operation of contraction merges two adjacent vertices* u *and* v *in the graph and removes the edge* (u, v)*.) If* G *is obtained from an induced subgraph of* G' *by contracting the edges, then it is said to be an* induced minor *of* G'*.*

A graph G is called G'-induced-minor-free (G'-minor-free) if G' is not an induced minor (resp. minor) of G.

An edge subdivision is an operation which deletes the edge (u, v) and adds a new vertex w and the edges (u, w) and (w, v). A graph G' obtained from G by a sequence of edge subdivisions is said to be a *subdivision* of G.

2.1 Representation of graphs

We assume the following time complexities for basic graph operations. Any representation that satisfies these is sufficient.

- Given u and v, it can be checked in $\widetilde{O}(1)$-time whether uv is an edge.
- Iterating over all the edges $xy \in E(H)$ ordered by x can be done in $\widetilde{O}(m)$-time, where m is the number of edges in H.

These requirements are satisfied by the following adjacency-list representation. To represent a graph H, we store a red-black tree T that contains non-isolated vertices of H where vertices are ordered according to their labels. Consider a node in this tree that corresponds to a vertex u. This node stores another red-black tree T_u that stores all neighbors of u in H. Now, to check whether uv is an edge or not, we perform a lookup for u in T followed by a lookup for v in T_u if u was found. We can iterate over all edges xy ordered by x by performing an inorder traversal of T where for each node u, we perform an inorder traversal of T_u.

If the pattern does not contain any isolated vertices, then we can ignore isolated vertices in the host graph as well. If the pattern is $G = G' + v$, where v is an isolated vertex and G' is any graph, then the number of homomorphisms from G to H is obtained by multiplying the number of homomorphisms from G' to H by n, where n is the number of vertices in H. This can be calculated by simply storing the number of vertices of H in the data structure.

3 Overview of proofs

Due to space constraints, we cannot include all the proofs in our paper. In this section, we summarize the main ideas involved in our proofs. We refer the reader to the full version for details.

3.1 Homomorphism counting

For any k-vertex pattern, the brute-force algorithm that iterates over all k-tuples of vertices in the host graph can be used to enumerate all homomorphisms from the pattern to the host in $O(n^k)$-time. Suppose the pattern has a perfect matching, then instead of iterating over k-tuples of vertices, we can iterate over all $k/2$-tuples of edges to enumerate all homomorphisms in $O(m^{k/2})$-time. This yields the maximum possible asymptotic improvement for sparse host graphs that does not improve the general algorithm.

The brute-force algorithm can be improved using dynamic programming over tree decompositions of the pattern graph [6]. Broadly, instead of iterating over all k-tuples of vertices, we only have to brute-force over tuples of vertices of size at most b, where b is the bag size in a tree decomposition, for each bag. This yields algorithms that run in $O(n^{t+1})$-time, where t is the treewidth of the pattern. The key observation that helps us to obtain improved algorithms for sparse graphs is that if the subgraph of the pattern induced by every bag in the tree decomposition contains a perfect matching (or a matching plus an, not disjoint, edge for bags with an odd number of vertices), then we can obtain a similar (to the previous paragraph) improvement to the dynamic programming algorithm. i.e., we can

turn the $O(n^{t+1})$-time dynamic programming algorithm to an $O(m^{\lceil (t+1)/2 \rceil})$-time dynamic programming algorithm. However, demanding that each bag contain a perfect matching results in a parameter that is more constrained than treewidth, the matched treewidth. Once matched treewidth is defined, the proof of correctness of the algorithm (See Proof of Theorem 6) is a straightforward generalization of the algorithm in [6].

The dynamic programming algorithm over treewidth requires polynomial space because an intermediate value may be reused any number of times. To reduce this space uses we can use divide-and-conquer algorithm instead of dynamic programming algorithm. The treedepth of the pattern governs the running-time of these algorithms. Broadly, these algorithms iterate over tuples of vertices of size at most d, where d is the length of a root-to-leaf path in the elimination tree. As before, if the subgraph of the pattern induced by the vertices in all paths in the elimination tree contains a perfect matching (or a matching plus an, not disjoint, edge for paths with an odd number of vertices), then, we can turn the $O(n^d)$-time algorithm to an $O(m^{d/2})$-time algorithm. Unlike the generalization for treewidth, this generalization is not as straight-forward as we also want to retain constant-space usage. This means that we require the perfect matching on each root-to-leaf path in the elimination tree occur from top-to-bottom. i.e., suppose a path v_1, v_2, v_3, v_4 is a root-to-leaf path in an elimination tree, then the matching has to be $\{v_1, v_2\}$ and $\{v_3, v_4\}$ and *not* $\{v_1, v_4\}$ and $\{v_2, v_3\}$. This allows the algorithm to recursively discover the vertices in the images of the homomorphism through edges in the host graph while moving from top to bottom in the elimination tree. Additionally, we also need the ability to iterate over all the edges of the host graph ordered by an end-point so that we need not use additional space to know whether or not we have finished processing a vertex.

3.2 Analysis of matched treewidth and matched treedepth

To obtain running-time improvements, we also prove that matched treewidth and matched treedepth are indeed close to treewidth and treedepth respectively for many interesting patterns. First, we prove that the worst-case is not too bad. In the proof of Theorem 3, we prove that half of matched treedepth is always strictly less than treedepth, yielding improvements even in the worst-case. The proof starts with an elimination tree and provides an algorithm that obtains a matched elimination tree by iteratively modifying it. For this proof, we introduce the notion of a connected elimination tree, an elimination tree where each vertex is connected to all of its subtrees. We show a lemma that may be of interest elsewhere, that we can assume wlog that the elimination tree is connected. A top-down iterative algorithm then converts the connected elimination tree into a matched elimination tree. Each iteration consists of two phases: The first phase connects (in the pattern) the current node to all its children, possibly violating connectivity; and the second phase re-establishes connectivity, possibly introducing a non-adjacent chlid (in the pattern) to the current node. By alternating between these two phases, we satisfy the property of matched elimination tree locally, at the current node and then proceed to deeper levels. The proof for the fact that half of matched treewidth is at most treewidth (See proof of Theorem 7) is similar. We start with a tree decomposition and provide an iterative, top-down algorithm that obtains a matched tree decomposition.

For obtaining improved algorithms for fixed patterns, we need some tools to determine when a pattern has small matched treewidth. Towards this end, we are able to completely characterize the matched treewidth of partial 2-trees by using forbidden induced minors. These proofs involve several different ideas specific to the (treewidth, matched treewidth) combination and therefore seems difficult to generalize.

We use contradiction method to prove both Theorem 10 and Theorem 11. The overall idea for the proof of both the theorems are similar. That is, for the backward direction, we assume the existence of a partial 2-tree G, that has the forbidden graph an induced minor. From a matched tree decomposition of G, we construct a matched tree decomposition of the forbidden graph with the same width. This gives a contradiction. For the other direction, we suppose for the contradiction that G is a counterexample with minimum number of vertices. We show the existence of chordal completion of G, whose standard tree decomposition is also a matched tree decomposition of G. This gives a contradiction. More details are discussed below.

Proof Idea of Theorem 10. We prove both directions using proof by contradiction. For the backward direction, we suppose for contradiction that there is a graph G which is a partial 2-tree such that $\mathfrak{mtw}(G) = 2$ and G has a C_5-induced-minor. We consider a matched tree decomposition T of G with width 2. Note that a C_5 can be obtained from G by deleting some vertices and contracting some edges. For all the deleted vertices we delete it from all the bags of T. Similarly, whenever an edge uv is contracted, we replace u and v in all the bags of T by one of u or v. Note that the obtained tree decomposition T' might not be matched. Since the width of T is two, any bag of T' is of size at most 3. Again any bag of size 3 in T' form a P_3. If a bag B in T' has size 1, then we can remove B by directly connecting all children of B to the parent of B. We show that a bag B of size two in T' does not contain two non-adjacent vertices in C_5. Hence, T' is a matched tree-decomposition of C_5, of width at most 2. This is a contradiction.

For the other direction, we suppose for contradiction that G is a counter example with minimum number of vertices. We prove some properties of G. First, we show that G does not have any cut-vertex. Since G is a partial 2-tree, we also obtain a 2-tree G' that is a super-graph of G and has the same vertex set as G. The standard tree decomposition for G' is matched. Since G is a counter-example, this tree decomposition should not be matched for G. But this means that there is a bag $\{u, v, w\}$ in the tree decomposition where it forms a triangle in G' but (say) v is not adjacent to u and w in G. We now show, using the fact that partial 2-trees are K_4-minor free, that any two adjacent vertices in G' are at a distance of at most two in G. Therefore, we obtain an induced cycle of length at least 5 in G using the vertices u, v, w, and the length-two paths between v and u, v and w, and possibly u and w. ◀

Proof Idea of Theorem 11. For the backward direction, we suppose for the contradiction that, there exists a partial 2-tree that has matched treewidth 3 and has an X-induced-minor. So, X can be obtained from an induced subgraph X' of G by contracting some edges. Let $\mathcal{U}$ be a matched tree decomposition of G and T' be the tree decomposition of X' obtained from $\mathcal{U}$ by deleting vertices in the set $V(G) \setminus V(X')$ from all the bags. Since X is obtained from X' by contracting some edges, there exist three mutually vertex disjoint paths p_0, p_2, p_4 such that contraction of all the edges in p_i gives u_i, for all the $i \in \{0, 2, 4\}$. Suppose X'' be the graph obtained from X' by contracting all the edges in the paths p_0, p_2, p_4 and T'' be the tree decomposition of X'', obtained from T' by replacing any vertex of p_i with u_i (for all the $i \in \{0, 2, 4\}$) in all the bags of T'.

Note that X can also be obtained from X'' by contracting some edges. Hence, there exists internally vertex-disjoints paths Q_{i+1}, Q'_{i+1} from u_i to u_{i+2} in X'', where the index $i \in \{0, 2, 4\}$ is in modulo 6. We define a map $f : V(X'') \mapsto V(X)$ such that $f(u_i) = u_i$ and maps all the internal vertices in the path Q_{i+1} to u_{i+1}, for all $i \in \{0, 2, 4\}$. Let T be the tree decomposition of X from T'' by applying f to all vertices in all bags. Note that there is no size 0 bags in T. We can remove size 1 bags from T using standard techniques. Note that if

uv is an edge in X'', then $f(u)f(v)$ is an edge in X, whenever $f(u) \neq f(v)$. Hence, and bag of size 4 in T is matched, since $\mathcal{U}$ is an matched tree decomposition of G. We show that no bag of size 3 in T is an independent set in X. Furthermore, by using these facts we argue that some modification of T can give a matched tree decomposition of X that has width at most that of T. This gives a contradiction.

For the other direction, we use two facts about partial 2-trees. The first fact is the well-known series-parallel characterization of partial 2-trees and the second one is the following property. Let G be a partial 2-tree and u, v be two vertices in G. Then uv is an edge in any chordal completion of G, whenever there are three internally disjoints paths from u to v in G. We show that for any X-induced-minor-free connected partial 2-tree G, there exists an minimum chordal completion $\tilde{G}$ such that no independent set of size 3 in G is a clique in $\tilde{G}$. Suppose for the contradiction that G is a minimum counterexample for the above statement. Since G is a minimum counterexample, it is not an edge. Furthermore, we show that G is not series composition of two smaller graphs.

Let G with source vertex s and terminal vertex t be the parallel composition of G_1 and G_2. We break our proof into two explicit cases depending on whether G_1 or G_2 is an edge. Before that we prove the following properties that we use in both the case to achieve the contradiction. Then we proved the following claim, that is used in both the cases to achieve a contradiction. If G_1 is a series compositions of two or more G'_i for $1 \leqslant i \leqslant m$. Then, for all $1 \leqslant i \leqslant m$, if G'_i is an edge, then s'_i is not adjacent to t'_i in G'_i.

In the first case we assume that, G_2 is an edge. So, s_1t_1 is not an edge in G_1, as otherwise $G_1 = G$. We showed that G_1 is a parallel composition of two smaller graph then G is not a counterexample. This gives a contradiction. Hence, (G_1, s_1, t_1) is a series composition of smaller graphs, say $(G'_1, s'_1, t'_1), \ldots, (G'_m, s'_m, t'_m)$, for some $m \geqslant 2$. By using the above claim about G_1 and G being minimum counterexample, we showed that there exists $1 \leqslant i \leqslant m$ such that s'_i is not adjacent to t'_j. Furthermore, we prove the following two claims. The first claim says the non-existence of $1 \leqslant j(\neq i) < k(\neq i) \leqslant m$, such that $s'_jt'_j, s'_kt'_k$ are non-edges. Then we prove the second claim, by using the property of partial 2-trees (Let G be a partial 2-tree and u, v be two vertices in G. Then uv is an edge in any chordal completion of G, whenever there are three internally disjoints paths from u to v in G). It gives the existence of an minimum chordal completion of G'_i in which no clique of size 3 is an in dependent set in G'_i and $s'_it'_i$ is an edge. By using these two claims we achieve a contradiction.

In the other case, we have both G_1 and G_2 are non-edges. First, we show that either G_1 or G_2 is not a parallel composition of smaller graphs. Then we decompose G_1 and G_2 into series composed graphs, until we cannot do further. G(with different source or terminal vertex) can be represented as parallel composition of g'_1 and an edge. Now, we follow previous case to achieve a contradiction. ◀

3.3 Algorithms for small patterns

The proofs of improved algorithms for small patterns such as paths and cycles use both homomorphism counting algorithms (Theorem 6 and Theorem 1) and characterizations. *Subgraph counting* algorithms for patterns invoke homomorphism counting on all homomorphic images of the pattern, called the spasm of the pattern. Even for small graphs such as P_{10}, the spasm is a very large set. However, using our characterization theorems for matched treewidth of partial 2-trees, we can conclude that all partial 2-trees in $\mathsf{Spasm}(P_{10})$ has matched treewidth at most 4 immediately yielding $O(m^2)$-time algorithms for counting homomorphisms. We are left with an analysis of only treewdith 3 graphs in the spasm of

which there are only few. However, for constant-space algorithms, we had to analyze a large number of graphs in $\mathsf{Spasm}(C_{11})$ and $\mathsf{Spasm}(C_{10})$. If treedepth is at most 4, then matched treedepth is at most 6 and we immediately obtain an $O(m^3)$-time, constant-space algorithm for counting homomorphisms. For graphs with treedepth 5 or higher, we have to do this analysis manually.

Finally, we consider induced subgraph detection of the patterns C_6 and $\overline{P_k}$. For induced subgraph detection, we have to build arithmetic circuits for homomorphism polynomials of graphs that are related to the pattern. These ideas were used by Bläser, Komarath, and Sreenivasaiah [2], for example to obtain $O(n^4)$-time algorithm for induced C_6 detection and $O(n^{k-2})$-time algorithm for induced $\overline{P_k}$ detection. The overall idea is to express the induced subgraph isomorphism polynomial of any k-vertex pattern G as a linear combination of subgraph isomorphism polynomials of all k-vertex super-graphs G' of G. Then, by choosing a suitable prime p, and by doing arithmetic modulo p, we can eliminate the harder super-graphs G' from the linear combination. Finally, we can check whether the remaining subgraph isomorphism exist or not by checking whether multilinear terms exist or not in an appropriately scaled homomorphism polynomial (See [14]). In adapting these algorithms to sparse graphs, the major challenge is to ensure that that the construction of these appropriately scaled homomorphism polynomials is possible in the required time. First of all, the scaling is done not by multiplying the final result by a constant (since the scaling quite often has to be done by a multiple of p, which is not possible when doing arithmetic modulo p), but by avoiding certain monomials from the polynomial altogether. For this, we need to ensure that the matched tree decompositions we consider are restricted even further. Indeed, we show how to do this for all the relevant super-graphs of C_6 in $\widetilde{O}(m^2)$-time and for $\overline{P_k}$ in $\widetilde{O}(m^{\lceil (k-2)/2 \rceil})$-time to obtain the maximal possible improvement.

4 Conclusion

Due to space constraints, we conclude our extended abstract here. The remaining sections that contain all theorems and proofs are in the appendix of this paper.

References

1. N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, March 1997. doi:10.1007/BF02523189.
2. Markus Bläser, Balagopal Komarath, and Karteek Sreenivasaiah. Graph pattern polynomials. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 18:1–18:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.18.
3. Marco Bressan and Marc Roth. Exact and approximate pattern counting in degenerate graphs: New algorithms, hardness results, and complexity dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. doi:10.1109/FOCS52979.2021.00036.
4. Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 210–223, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055502.
5. Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 130–139. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.22.

6 Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Counting h-colorings of partial k-trees. In *Proceedings of the 7th Annual International Conference on Computing and Combinatorics*, COCOON '01, pages 298–307, Berlin, Heidelberg, 2001. Springer-Verlag.

7 Julian Dörfler, Marc Roth, Johannes Schmitt, and Philip Wellnitz. Counting induced subgraphs: An algebraic approach to #w[1]-hardness. *Algorithmica*, 84(2):379–404, 2022. doi:10.1007/s00453-021-00894-9.

8 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326:57–67, October 2004. doi:10.1016/j.tcs.2004.05.009.

9 Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Induced subgraph isomorphism: Are some patterns substantially easier than others? In Joachim Gudmundsson, Julián Mestre, and Taso Viglas, editors, *Computing and Combinatorics*, pages 37–48, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

10 Jacob Focke and Marc Roth. Counting small induced subgraphs with hereditary properties. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20–24, 2022*, pages 1543–1551. ACM, 2022. doi:10.1145/3519935.3520008.

11 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *J. Comput. Syst. Sci.*, 78(3):698–706, 2012. doi:10.1016/j.jcss.2011.10.001.

12 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Inf. Process. Lett.*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.

13 Balagopal Komarath, Anurag Pandey, and C. S. Rahul. Graph homomorphism polynomials: Algorithms and complexity. *CoRR*, abs/2011.04778, 2020. arXiv:2011.04778.

14 Ioannis Koutis. Faster algebraic algorithms for path and packing problems. In *International Colloquium on Automata, Languages, and Programming*, pages 575–586. Springer, 2008.

15 Miroslaw Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM J. Discret. Math.*, 27:892–909, 2013.

16 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2):415–419, 1985. URL: http://eudml.org/doc/17394.

17 Marc Roth, Johannes Schmitt, and Philip Wellnitz. Detecting and Counting Small Subgraphs, and Evaluating a Parameterized Tutte Polynomial: Lower Bounds via Toroidal Grids and Cayley Graph Expanders. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 108:1–108:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.108.

18 Virginia Vassilevska. *Efficient Algorithms for Path Problems in Weighted Graphs*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 2008.

19 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, September 2000.

20 Ryan Williams. Maximum two-satisfiability. *Encyclopedia of Algorithms*, pages 507–510, 2008. doi:10.1007/978-0-387-30162-4_227.

21 Ryan Williams. Finding paths of length k in o*(2k) time. *Inf. Process. Lett.*, 109(6):315–318, February 2009. doi:10.1016/j.ipl.2008.11.004.

22 Virginia Vassilevska Williams, Joshua R. Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pages 1671–1680, USA, 2015. Society for Industrial and Applied Mathematics.

The following appendices contain the remaining sections of the full paper.

A Matched treedepth

In this section, we introduce algorithms that count homomorphisms and subgraphs efficiently in constant space on sparse host graphs. The central theorem in this section is given below.

▶ **Theorem 1.** *Let* G *be a graph with* $\mathsf{mtd}(G) = d$*, then given an* m*-edge graph* H *as input, we can count the number of homomorphisms from* G *to* H *in* $\widetilde{O}(m^{\lceil d/2 \rceil})$*-time and constant space.*

Proof. The algorithm is given in Algorithm 1. We can compute the result needed by calling COUNT-HOM-MTD(G, E, r, H, ϕ), where E is an elimination tree for G of depth d, r is the root vertex in E, and ϕ is the empty homomorphism. For simplicity of presentation, we assume that each root-to-leaf path in E has an even number of vertices. Odd number of vertices in a root-to-leaf path is handled similarly.

We assume that the host graph H is represented using a symmetric adjacency list representation. This is mainly to ensure that we can iterate over all edges xy in H ordered by x in Line 3.

First, we prove that the algorithm is correct. We claim that the call COUNT-HOM-MTD (G, E, v, H, σ) where the parameters are as specified in the algorithm returns the number of homomorphisms from G_v to H that extends σ. This is proved by an induction on the height of v in E. Since v is a top node, the base case is when the height is 2. In this case, G_v is a star graph and it is easy to see that the algorithm works. We now prove the inductive case. The variable t computes the final answer. Denote by $s_{u,x}$ for vertex x in H the number of homomorphisms from G_u to H that extends $\tau = \sigma \cup \{v \mapsto x\}$. Notice that since $uv \in E(G)$, to extend τ, the vertex u must be mapped to some y such that $xy \in E(H)$. Therefore, iterating over all such y is sufficient. Notice that we can compute t as $\sum_\tau \prod_u s_{u,x}$. However, this would need storing $|V(G)||V(H)|$ variables. By iterating over the edges of H ordered by x, we can afford to reuse a single s_u for different x instead of keeping a separate $s_{u,x}$ for each x. Now, we show that s_u correctly computes $s_{u,x}$ once the main loop finishes with an x. By the inductive hypothesis, the variable c_w is the number of homomorphisms from G_w to H that extends $\sigma' = \sigma \cup \{v \mapsto x, u \mapsto y\}$. Therefore, we have $s_{u,x} = \sum_y \prod_w c_w$. Line 12 correctly computes this into s_u. Line 15 correctly updates t once an x is finished. Finally, we reset s_u to 0 before processing the next x.

Now, we prove the running-time and space usage of the algorithm. Notice that the depth of the recursion is bounded by the depth of the elimination tree E and each level of recursion stores only constantly many variables. Therefore, the space usage is constant. The main loop in Line 3 runs for $2m$ iterations. The inner-loops only have a constant number of iterations. Therefore, the recursive calls are made only $O(m)$ times. We process two levels of the elimination tree in a level. Therefore, the total running-time is given by $t(d) \leqslant O(m)t(d-2) + \widetilde{O}(1) = \widetilde{O}(m^{d/2})$.

Algorithm 1 COUNT-HOM-MTD(G, E, v, H, σ).

Require: G - The pattern graph
Require: E - Matched elimination tree for G
Require: v - A *top* vertex in E
Require: H - The host graph
Require: σ - A partial homomorphism from the ancestors of v to H

```
t ← 0
s_u ← 0 for all children u of v
for all edges xy ∈ E(H) ordered by x do
    for all children u of v in E do
        σ' ← σ ∪ {v ↦ x, u ↦ y}
        if σ' is an invalid homomorphism then
            continue
        end if
        for all children w of u do
            c_w ← COUNT-HOM-MTD(G, E, w, H, σ')
        end for
        s_u ← s_u + ∏_w c_w
    end for
    if xy is the last edge on x then
        t ← t + ∏_u s_u
        s_u ← 0 for all u
    end if
end for
return t
```

◀

▶ **Definition 25.** *An elimination tree* T *is* connected *if for every node* u *in* T *and a child* v *of* u *in* T, u *is adjacent to some node in* T_v

Now, we show that connected elimination trees are optimal.

▶ **Lemma 26.** *Every connected graph* G *has a connected elimination tree of depth* $\mathsf{td}(G)$.

Proof. We show how to construct a connected elimination tree T' from an elimination tree T without increasing its depth. Let $T' = T$ initially. Suppose there exists some node u in T' that violates the property (If not, we are done). Then, there exists a child v of u in T' such that there is no edge in G between u and any node in T'_v. Let w be a node in T'_v such that w is adjacent to some proper ancestor x of u. Such a w must exist because G is connected. Now, in T', remove the subtree T'_v and make it a subtree of the node x. Repeat this process until all nodes in T' satisfy the required property. This process must terminate since at each step, we reduce the number of nodes violating the property by at least one. This process cannot increase the depth of T' because the only modification is to move a subtree upwards to be a subtree of a proper ancestor of its parent. ◀

▶ **Theorem 3.** *For any graph* G, $\mathsf{mtd}(G) \leqslant 2 \cdot \mathsf{td}(G) - 2$.

Proof. We start with a connected elimination tree T with depth d of G and show how to construct a matched elimination tree of G from T. We use induction on d. For $d = 2$, the tree is already matched and has depth $2 = 2 \cdot 2 - 2$.

Our construction is iterative and top-down. Each iteration ensures that the current top-most node in the elimination tree is adjacent, in the graph G, to all its children and that the elimination tree is connected.

Each iteration consists of two phases iteratively executed until the desired property is satisfied. The first phase ensures that the root node r is adjacent in G to all its children in T. If r has a child v that is not adjacent in G to r, then since T is connected, there is some node w in T_v such that $rw \in E(G)$. Then, we make w a child of r in T, delete w from T_v, and make all children of w children of parent of w. The resulting tree is an elimination tree of depth at most $d+1$. After this phase, the root node is adjacent in G to all its children, the tree's depth has increased by at most one. However, it may not be connected.

In the second phase, we use the construction of Lemma 26 to make the tree connected without increasing the depth. Observe that the construction will keep the existing children of root as is and may add new chlidren to r that are not adjacent to r in G. Suppose a new node u was added as a new child to r in this second phase. The height of subtree rooted at u is at most $d-1$. Therefore, the tree (r, T_u) obtained by attaching u to r has height at most d. We can now execute phase 1 on all the trees (r, T_u) for all such u without increasing depth beyond $d+1$. This process must eventually terminate as we add at least one new child to the root every time.

At the end of the iteration, consider a grandchild x of r. If it is a leaf, since the tree is connected, x must be adjacent in G to its parent in T and we are done. Otherwise, the subtree T_x is a tree of depth at least 2 and at most $d-1$ that is connected. So by the induction hypothesis, we obtain that T_x is a matched elimination tree of depth at most $2(d-1)-2$. This means that the original tree is converted to a matched elimination tree of depth at most $2+2(d-1)-2 = 2d-2$ as required. ◀

▶ **Theorem 4.** *Let G be a graph and G' is a connected, induced subgraph of G, then:*

1. $\mathsf{mtd}(G') \leqslant \mathsf{mtd}(G)$ *if* $\mathsf{mtd}(G)$ *is even.*
2. $\mathsf{mtd}(G') \leqslant \mathsf{mtd}(G)+1$ *if* $\mathsf{mtd}(G)$ *is odd.*

Proof. We start with a matched elimination tree T of even (The odd case is similar) depth d for G and construct a matched elimination tree for G'. First, we delete all nodes in the elimination tree that are in G but not in G'. If a node u in T has parent v that was deleted, then we make u a child of the closest ancestor of v that is still in T. The final forest thus obtained is a tree T' because G' is connected. We assume without loss of generality that T' is a connected elimination tree.

Suppose r' is the root of T'. We will now modify T' into a matched elimination tree. We will first analyze paths in T' that correspond to even length root-to-leaf paths in T. If T' is not matched on this path, then there exists a node u closest to r' such that u is not connected in G' to a child v in T'. This is only possible if u was either matched to a child w of u in T or its parent w in T and w is not in G'. Therefore, we have $\mathsf{dist}_{T'}(r', u) + \mathsf{depth}(T'_v) < \mathsf{depth}(T)$ and this means we can afford to increase the length of any path that passes through edge uv by 1. Since T' is connected, we can now apply the transformation in the proof of Theorem 3 to match u with one of its descendants in T'_v. The depth is still at most d because this transformation increases the depth by at most 1 In effect, the increase in depth in this branch of the tree by pulling up a descendant is compensated by the fact that the unmatched vertex was introduced by deleting a vertex in this branch.

We can iteratively apply the above construction to make T' a matched elimination tree while keeping T' connected. However, applying the above transformation may introduce a node u in T' that is not matched to a child v *because* v's parent w was pulled up in the tree to

match with some other vertex. Such u also satisfy the inequality $\mathsf{dist}_{T'}(r', u) + \mathsf{depth}(T'_v) < \mathsf{depth}(T)$. Why? Any path in the tree T' that passed through both u and v earlier had to pass through w. But, the fact that w was pulled up implies that these paths had length strictly less than $\mathsf{depth}(T)$ by the argument in the previous paragraph. And shifting w to a position earlier in the path cannot increase its length.

For root-to-leaf paths in T of odd length, there might be a matched P_3 on vertices uvw such that v is not in G'. In this case too, by the same argument, the transformations increase the depth to at most $d + 1$ (In this case, we may have to pull up two distinct vertices for matching u and w.). When d is even, increasing the length of such paths by 1 does not increase the depth of the tree. When d is odd, increasing the length of such paths by 1 increases the depth by at most 1. ◀

B Matched treewidth

▶ **Theorem 12.** *Given an m-edge graph H as input, we can count the number of P_k subgraphs, where $k \leqslant 10$, in $\widetilde{O}(m^2)$-time.*

Proof. There are more than 300 graphs in $\mathsf{Spasm}(P_{10})$. We have verified that all of them have mtw at most 3. To minimize the work, we can filter out all graphs in the spasm that has $\mathsf{tw}(G') = 1$ or $\mathsf{tw}(G') = 2$ and G' is X-induced-minor-free. Observe that since X has 9 vertices 12 edges, it cannot be an induced minor in any of the graphs in $\mathsf{Spasm}(P_{10})$. Also, none of the forbidden minors for treewidth 4 can appear in $\mathsf{Spasm}(P_{10})$. Therefore, we only need to analyze graphs of treewidth 3 in $\mathsf{Spasm}(P_{10})$. There are only 18 such graphs. They are listed in a pdf file in the repository associated with this paper (`https://github.com/anonymous1203/Spasm`).

Since $\mathsf{Spasm}(P_k) \subseteq \mathsf{Spasm}(P_{k+1})$ for all $k \geqslant 2$, we can make the same claim for all paths on fewer than 10 vertices. We now use

$$\mathsf{Sub}_G[H] = \sum_{G'} \alpha_{G'} \mathsf{Hom}_{G'}[H] \tag{1}$$

to compute the result. ◀

Maximum Matching via Maximal Matching Queries

Christian Konrad
Department of Computer Science, University of Bristol, UK

Kheeran K. Naidu
Department of Computer Science, University of Bristol, UK

Arun Steward
Department of Computer Science, University of Bristol, UK

Abstract

We study approximation algorithms for Maximum Matching that are given access to the input graph solely via an *edge-query maximal matching oracle*. More specifically, in each round, an algorithm queries a set of potential edges and the oracle returns a maximal matching in the subgraph spanned by the query edges that are also contained in the input graph. This model is more general than the *vertex-query model* introduced by binti Khalil and Konrad [FSTTCS'20], where each query consists of a subset of vertices and the oracle returns a maximal matching in the subgraph of the input graph induced by the queried vertices.

In this paper, we give tight bounds for deterministic edge-query algorithms for up to three rounds. In more detail:

1. As our main result, we give a deterministic 3-round edge-query algorithm with approximation factor 0.625 on bipartite graphs. This result establishes a separation between the edge-query and the vertex-query models since every deterministic 3-round vertex-query algorithm has an approximation factor of at most 0.6 [binti Khalil, Konrad, FSTTCS'20], even on bipartite graphs. Our algorithm can also be implemented in the semi-streaming model of computation in a straightforward manner and improves upon the state-of-the-art 3-pass 0.6111-approximation algorithm by Feldman and Szarf [APPROX'22] for bipartite graphs.
2. We show that the aforementioned algorithm is optimal in that every deterministic 3-round edge-query algorithm has an approximation factor of at most 0.625, even on bipartite graphs.
3. Last, we also give optimal bounds for one and two query rounds, where the best approximation factors achievable are $1/2$ and $1/2 + \Theta(\frac{1}{n})$, respectively, where n is the number of vertices in the input graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Maximum Matching, Query Model, Algorithms, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.41

Funding C.K. is supported by EPSRC New Investigator Award EP/V010611/1. K.K.N. is supported by EPSRC DTP studentship EP/T517872/1.

1 Introduction

In this work, we study approximation algorithms for the Maximum Matching problem (MM) and its bipartite version, the Maximum Bipartite Matching problem (MBM), that are only given query access to the input graph $G = (V, E)$ via a *maximal matching oracle*. A *matching* $M \subseteq E$ in graph G is a subset of vertex-disjoint edges. The matching M is *maximum* if $|M| \geq |M'|$, for every other matching M'. The *matching number* $\mu(G)$ of a graph G is the size of a maximum matching. Furthermore, a matching M is *maximal* if it is inclusion-wise maximal, i.e., $M \cup \{e\}$ is not a matching, for every $e \in E \setminus M$. In each round i of the maximal matching edge-query model, the algorithm sends a set of potential edges $Q_i \subseteq V \times V$, denoted


© Christian Konrad, Kheeran K. Naidu, and Arun Steward;
licensed under Creative Commons License CC-BY 4.0

the query edges, to the oracle, which in turn responds with an arbitrary maximal matching in the subgraph $G[Q_i \cap E]$, i.e., the subgraph of G spanned by the query edges that are also contained in G. For brevity, we will refer to this model as the *edge-query model*.

Maximal Matching Queries. The study of algorithms for MM that solely execute maximal matching queries was introduced by binti Khalil and Konrad [8]. They considered a *vertex-query model*, where, in each round i, the algorithm queries a subset of vertices $U_i \subseteq V$, and the oracle responds with an arbitrary maximal matching in subgraph $G[U_i]$, i.e., the subgraph induced by vertices U_i. The edge-query model is more general than the vertex-query model since vertex queries can be simulated in the edge query model: For each query $U_i \subseteq V$ in the vertex-query model, querying the set of edges Q_i that turns U_i into a clique yields an equivalent edge-query algorithm.

The study of maximal matching query models is motivated by the fact that, in many computational models, including the data streaming model [20] and the Massively Parallel Computation model [17], computing maximal matchings is easy, while computing substantially larger matchings is more challenging. Computing maximal matchings can thus be regarded as a black-box subroutine, which allows for the design of matching algorithms that are independent of the underlying computational model.

For example, in the *semi-streaming model* for processing large graphs [12, 20], an algorithm makes few passes over the edges of the input graph in arbitrary order while maintaining a memory of size $O(n \operatorname{poly} \log n)$, where n is the number of vertices in the input graph. The Greedy matching algorithm, which inserts every arriving edge into an initially empty matching if possible, i.e., if none of its endpoints are already matched, yields a maximal matching and constitutes a one-pass semi-streaming algorithm. Since a maximal matching is at least half the size of a maximum matching, Greedy can also be regarded as a $\frac{1}{2}$-approximation semi-streaming algorithm for MM. While it is unknown whether it is possible to go beyond the approximation factor of $1/2$ in a single pass even on bipartite graphs (currently only approximation factors beyond $\frac{1}{1+\ln 2} \approx 0.59$ are ruled out [16]), improved results are known for multiple passes, and, indeed, most multi-pass semi-streaming algorithms solely execute Greedy on carefully selected subgraphs in each pass (e.g. [9, 2, 18, 5]). This includes the state-of-the-art[1] $(1-\epsilon)$-approximation algorithm for MBM by Assadi et al. [5], which executes Greedy $O(\frac{1}{\epsilon^2})$ times and thus runs in $O(\frac{1}{\epsilon^2})$ passes. This algorithm can easily be implemented in the Massively Parallel Computation model [17] and also constitutes the state-of-the-art result in this model. As such, maximal matching query models capture these algorithms and allow for a systematic study of what can and cannot be achieved.

Our Results. In this paper, we give tight approximation ratios for deterministic edge-query algorithms for MBM for up to three rounds. Our results for one and two rounds as well as the lower bound for three rounds also hold for MM. In Table 1, we illustrate our bounds and compare them to the respective tight bounds that holds in the vertex-query model [8].

One Round. We show that the best approximation factor achievable in a single round for MBM is $\frac{1}{2}$, which matches the vertex-query setting (Theorem 15-1). Querying all potential edges, i.e., the query $V \times V$, yields a matching upper bound, even in general graphs.

[1] We note that the algorithm by [5] yields a $(1-\epsilon)$-approximation in $O(\frac{1}{\epsilon^2})$ passes. Very recently, Assadi et al. [4] gave a $(1-\epsilon)$-approximation algorithms for MBM that operates in $O(\frac{1}{\epsilon} \cdot \log n)$-passes, which, for very small values of ϵ, asymptotically requires fewer rounds than [5].

Table 1 Optimal approximation ratios achievable for deterministic algorithms for MBM in the edge-query (this paper) and vertex-query models ([8]).

# Rounds	Vertex-query model ([8])	Edge-query model (this paper)
1	$\frac{1}{2}$	$\frac{1}{2}$ (Theorem 15-1)
2	$\frac{1}{2}$	$\frac{1}{2} + \Theta(\frac{1}{n})$ (Theorem 15-2)
3	$\frac{3}{5} = 0.6$	$\frac{5}{8} = 0.625$ (Theorems 1 and 15-3)

Two Rounds. The approximation factor can be very slightly improved in two rounds, even in general graphs. Consider the algorithm that queries all edges $V \times V$ in the first round, which produces a maximal matching M_1 in the input graph. Next, pick any edge $uv \in M_1$ and query all edges incident to u and v different to uv in the next round. Then, we will either find a 3-augmenting path that allows us to augment the edge uv, or the edge uv is not 3-augmentable. In both cases, we establish that the resulting matching is a $\frac{1}{2} + \Theta(\frac{1}{n})$-approximation, and we also prove that no algorithm can do better, even in bipartite graphs (Theorem 15-2). While the $\Theta(\frac{1}{n})$ additive term is not significant in terms of an improved approximation guarantee, it nevertheless illustrates that the edge-query and vertex-query models behave slightly differently in the two rounds setting.

Three Rounds. As our main result, we give a deterministic 3-round algorithm for MBM in the edge-query model that produces a 0.625-approximation (Theorem 1), and we show that this is best possible (Theorem 15-3). Our algorithm can be implemented in a straightforward way in the semi-streaming model and improves upon the previously best 3-pass semi-streaming 0.6111-approximation algorithm for MBM by Feldman and Szarf [13].

On 3-pass Semi-streaming Algorithms for MBM. The first 3-pass semi-streaming algorithm for MBM was implicit in [12], explicitly mentioned in [19], and analysed by Kale and Tirodkar [15], who showed that the approximation ratio is 0.6. Subsequently, binti Khalil and Konrad [8] proved that this algorithm constitutes an optimal 3-round vertex-query algorithm (and can thus also be implemented in the edge-query model). Various improvements have since been established via semi-streaming algorithms that cannot be implemented in the edge-query model. First, Esfandiari et al. [10] gave a 0.605-approximation algorithm, which was then further improved by Konrad [18] who gave a randomized 0.6067-approximation algorithm. Very recently, Feldman and Szarf [13] gave a 0.6111-approximation. In this paper, we improve the approximation factor to 0.625, again, with a 3-round deterministic edge-query algorithm.

While edge-query algorithms appear somewhat restricted in how they operate as compared to arbitrary semi-streaming algorithms, the literature illustrates that they are surprisingly powerful as they constitute the state-of-the-art algorithms in the 3-pass (this paper) and $(1-\epsilon)$-approximation [5] streaming settings for MBM.

Techniques. We will first discuss the ideas behind our 3-round algorithm for MBM in the edge-query model, and then give the intuition behind our lower bound results.

3-round Query Algorithm. Our 3-round algorithm computes a maximal matching in the first round, and then finds augmenting paths in the subsequent rounds. This is a well-established technique, and almost all known 2-pass and 3-pass streaming algorithms operate in this fashion (e.g. [19, 10, 15, 18, 13]). To this end, denote by M_1 a maximal matching in the bipartite input graph $G = (A, B, E)$ that we obtain by querying all potential edges $A \times B$. We observe that every augmenting path for M_1 starts with an edge in $G_L = G[A(M_1) \cup \overline{B(M_1)}]$

and ends with an edge in $G_R = G[\overline{A(M_1)} \cup B(M_1)]$, where $A(M_1)$ denotes the A-vertices matched in M_1, and $\overline{A(M_1)} = A \setminus A(M_1)$ ($B(M_1)$ and $\overline{B(M_1)}$ are defined similarly). In our second round, we therefore compute maximal matchings M_L and M_R in G_L and G_R, respectively. Observe that we can indeed compute both of these matchings with the single query $\left(A(M_1) \times \overline{B(M_1)}\right) \cup \left(\overline{A(M_1)} \times B(M_1)\right)$ in the edge-query model. At this stage, we are guaranteed that the set $M_1 \cup M_L \cup M_R$ contains various length-2 paths consisting of one edge of M_1 and one additional edge either from M_L or M_R, and the usual idea employed in the literature is to complete these length-2 paths to length-3 augmenting paths using an additional pass over the data/an additional query. Indeed, if we attempted to complete the length-2 paths by computing a maximal matching between the endpoints of length-2 paths in M_1 and the yet unmatched vertices in the third round then we obtain a 0.6-approximation.

Our key idea for the third query round that leads to an improvement over previous work is to simultaneously attempt to complete length-5 augmenting paths. To this end, denote by A' the endpoints of length-2 paths in $A(M_1)$, and by B' the endpoints of length-2 paths in $B(M_1)$. Our third round query consists of all potential edges interconnecting the vertices

$$A' \cup B' \cup \overline{A(M_1)} \cup \overline{B(M_1)} \ ,$$

i.e., we both attempt to complete length-2 paths to length-3 augmenting paths by considering the edges between A' and $\overline{B(M_1)}$ and between B' and $\overline{A(M_1)}$, but we also attempt to join two disjoint length-2 paths by connecting them via an edge between A' and B' to form a length-5 augmenting path. The main challenge in the analysis of this method is to address the complications that arise from the fact that the single maximal matching returned in round 3 completes both length-3 and length-5 augmenting paths.

Lower Bounds. The key idea behind our lower bound arguments is to keep track of the information revealed when the oracle returns a maximal matching M_i as a response to the query edges Q_i in round i. This approach was previously successfully employed by binti Khalil and Konrad [8] for obtaining optimal lower bounds in the vertex-query model. When a matching M_i is returned, the algorithm not only learns that the edges M_i are indeed contained in the input graph, but also that none of the edges of Q_i that connect vertices outside of $V(M_i)$ exist, which is due to the maximality of M_i in the subgraph $G[E \cap Q_i]$ of the input graph $G = (V, E)$. The main challenge lies in keeping track of the information revealed over the course of the algorithm while considering the complexity of *all* possible queries in each round. This is achieved by considering the vertex-induced subgraphs on carefully constructed partitions of the vertices while maintaining a *superset* of the information revealed to the algorithm in each part: We prove that, no matter the sequence of queries, the information about the edges that are guaranteed to exist and those that are guaranteed not to exist in each part of the partition is less than a certain superset of existing edges and non-existing edges that are easy to describe, up to isomorphism. Our arguments are substantially more involved than those for the vertex-query model [8], which is due to the fact that edge queries can have more complex structure.

Further Related Work. The study of graph algorithms with query access to the input dates back to the works of Feige [11] and Goldreich and Ron [14]. The literature distinguishes between *local queries* – such as vertex-degree queries, neighborhood queries, and edge-existence queries [11, 14, 7] – and *global queries* – such as (bipartite) independent set queries [6, 1], linear, or and cut queries [3], and maximal matching queries as studied in this paper [8]. We refer the reader to [1] and the references therein for an overview.

Outline. We first present our main result, a 3-round algorithm for MBM, in Section 2. Our lower bound results are discussed in Section 3, and we conclude with open questions in Section 4.

2 3-Round Algorithm

In this section, we present our 3-round query algorithm for MBM (see Algorithm 1). We prove the following result, which constitutes the main result of this paper:

▶ **Theorem 1.** *Algorithm 1 is a deterministic 3-round $\frac{5}{8}$-approximation algorithm for MBM in the edge-query model.*

Algorithm 1 Three Rounds using Maximal Matching Queries.

Input: A bipartite graph $G = (A, B, E)$ and a maximal matching oracle QUERY
Output: A large matching M_{out} of G

First round

$M_1 \leftarrow \text{QUERY}(G[A \cup B])$
$G_L = G[A(M_1) \cup \overline{B(M_1)}]$
$G_R = G[\overline{A(M_1)} \cup B(M_1)]$

Second round

$M_2 \leftarrow \text{QUERY}(G_L \cup G_R)$
Let $A' \subseteq A(M_1)$ and $B' \subseteq B(M_1)$ be the endpoints of length-2 paths in $M_1 \cup M_2$
$G' = G[A' \cup B' \cup \overline{A(M_1)} \cup \overline{B(M_1)}]$

Third round

$M_3 \leftarrow \text{QUERY}(G')$

Output

return M_{out}, the largest matching in $M_1 \cup M_2 \cup M_3$

Our algorithm operates on a bipartite input graph $G = (A, B, E)$, where only the vertex sets A and B are initially known to the algorithm. It only uses the edge-query maximal matching oracle to compute maximal matchings in subgraphs of G. Our algorithm initially computes a maximal matching M_1 of G in the first round. Then, in the second round, maximal matchings in the subgraphs G_L and G_R are computed (M_2 denotes the union of both matchings), where G_L consists of the edges connecting the B-vertices unmatched in M_1 to the A-vertices matched in M_1 and G_R is defined similarly with the roles of A and B reversed. The matching M_2 w.r.t. M_1 possibly finds some length-3 augmenting paths, which we denote by P, while the remaining ones make up length-2 alternating paths. Last, in the third round, a maximal matching M_3 is computed in the subgraph induced by the vertices unmatched by M_1 and the endpoints of the length-2 paths in $M_1 \cup M_2$ that are also in $A(M_1) \cup B(M_1)$. Each edge of the matching M_3 thus completes length-3 and length-5 augmenting paths, which we denote by Q. See Figure 1 for an example run of the algorithm where $G_{\text{in}} = G[A(M_1) \cup B(M_1)]$.

▶ **Observation 2.** *The size of Q is exactly the size of M_3.*

The goal of our analysis is to show that the size of the returned matching M_{out}, the largest matching in $M_1 \cup M_2 \cup M_3$, is always at least a $\frac{5}{8}$-approximation of a maximum matching M^*. To that end, we can always find a large maximal matching by appropriately augmenting M_1

with the augmenting paths $P \cup Q$. Although each edge in M_1 can be augmented by at most one augmenting path, the augmenting paths $P \cup Q$ are not necessarily vertex-disjoint since the vertices unmatched by M_1 may be incident to an edge in M_2 and also one in M_3. See Figure 1 for an example of this. However, we observe that the intersection multi-graph of the augmenting paths $P \cup Q$ has maximum degree 2 and, in particular, constitutes a collection of paths and even-length cycles. We can thus pick an independent set of non-overlapping augmenting paths of size $\frac{1}{2}(|P| + |Q|)$ and thus obtain the following:

$$|M_{\text{out}}| \geq |M_1| + \frac{1}{2}(|P| + |Q|). \tag{1}$$

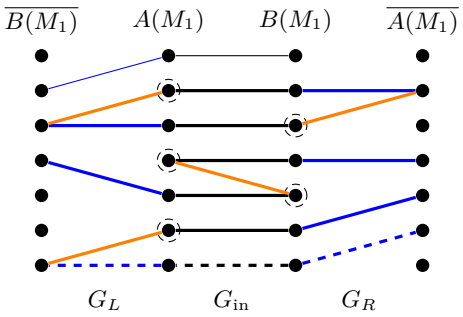


Figure 1 An example run of Algorithm 1 that showcases some possible intersections of the augmenting paths. The edges of M_1 are in black, the edges of M_2 are in blue, and the edges of M_3 are in orange. The dashed thick edges are those which belong to augmenting paths in P, whereas the solid thick edges are those which belong to Q. The vertices of A' and B' are circled.

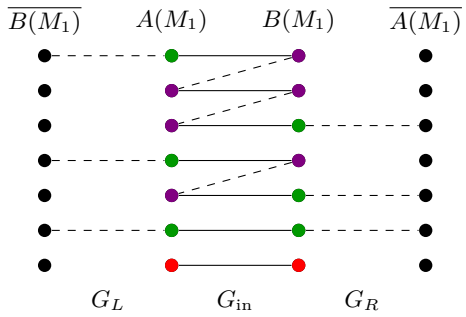


Figure 2 An example of the implied graph structure w.r.t. M^* and M_1. The edges of M_1 are the solid edges and the edges of $M^* \backslash M_1$ are dashed ones. The edges of $M^* \cap M_1$ are the solid edges not incident to any dashed ones. The red vertices represent A_* and B_*, the green ones represent A_{out} and B_{out}, and the violet ones represent A_{in} and B_{in}.

We highlight here that either finding a large matching in the first round or finding many augmenting paths in the second round leaves fewer augmenting paths to be found in the third round. Therefore, the size of Q must be a decreasing function of $|M_1|$ and $|P|$. We subsequently formalise this by systematically bounding the quantities required to bound the size of M_3, which is equivalent to the size of Q (Observation 2). Theorem 1 then immediately follows from Equation (1).

Let M^* be a maximum matching in G with size $\mu(G)$. The first query finds a maximal matching M_1 of G, which is always at least half the size of M^* (Observation 3). By considering a maximum matching M^* such that $M^* \cup M_1$ contains no even-length alternating paths or cycles (Lemma 4), each vertex in $A(M_1)$ and $B(M_1)$ are endpoints of exactly one edge in M^* (Lemma 5) and we have that M_1 relates exactly to the the number of edges of M^* in G_L and G_R, respectively (Lemma 6). We give these proofs in Appendix A for completeness. As such, for the remainder of the analysis, we assume this choice of M^*. See Figure 2 for an example of the implied structure.

▶ **Observation 3.** $|M_1| = (\frac{1}{2} + \epsilon) \cdot \mu(G)$, $0 \leq \epsilon \leq \frac{1}{2}$.

▶ **Lemma 4.** *There exists a maximum matching M^* such that $M^* \cup M_1$ has no even-length alternating paths or cycles.*

▶ **Lemma 5.** *Each vertex of $A(M_1)$ and $B(M_1)$ is the endpoint of an edge in M^*.*

▶ **Lemma 6.** $|M^* \cap G_L| = |M^* \cap G_R| = (\frac{1}{2} - \epsilon) \cdot \mu(G)$.

We now have that the vertices $A(M_1)$ and $B(M_1)$ can be partitioned based on the kind of edge in M^* that they are endpoints of, i.e., by Lemma 5. Let $A_* := A(M^* \cap M_1)$, $A_{\text{out}} := A(M^* \cap G_L)$, and $A_{\text{in}} := A(M_1) \setminus (A_* \cup A_{\text{out}})$. Similarly define $B_* := B(M^* \cap M_1)$, $B_{\text{out}} := B(M^* \cap G_R)$ and $B_{\text{in}} := B(M_1) \setminus (B_* \cup B_{\text{out}})$. See the coloured vertices of Figure 2 for an example of these partitionings.

▶ **Lemma 7.** $|A_{\text{in}}| = |B_{\text{in}}| = 2\epsilon \cdot \mu(G) - |M^* \cap M_1|$.

Proof. By construction of the partitions, we have that $|A_*| = |M^* \cap M_1| = |B_*|$, $|A_{\text{out}}| = |M^* \cap G_L|$ and $|B_{\text{out}}| = |M^* \cap G_R|$. Then, by Lemma 6, it follows that $|A_{\text{out}}| = |B_{\text{out}}| = (\frac{1}{2} - \epsilon) \cdot \mu(G)$. Finally, since $|A(M_1)| = |B(M_1)| = |M_1| = (\frac{1}{2} + \epsilon) \cdot \mu(G)$ by Observation 3, we have that $|A_{\text{in}}| = (\frac{1}{2} + \epsilon) \cdot \mu(G) - |A_{\text{out}}| - |A_*| = 2\epsilon \cdot \mu(G) - |M^* \cap M_1| = |B_{\text{in}}|$. ◀

Intuitively, we have that finding edges of M^* with the first query always puts us in a better situation. Therefore, we consider the effect that these edges have on the quantities that we bound. We introduce some helpful notation in this regard: For any matching M, we define $M(A_*)$ (and $M(B_*)$) as the edges of M that have one endpoint in A_* (resp. B_*), i.e., those which are incident to edges of $M^* \cap M_1$.

The second query finds a maximal matching M_2, which is the union of (vertex-disjoint) maximal matchings M_L in G_L and M_R in G_R. Each edge of M_2 forms the beginning of an alternating path in $M_1 \cup M_2$, some of which may immediately form length-3 augmenting paths P while the rest form length-2 alternating paths P', which are extended in the third round. Note that the endpoints of P' that are in $A(M_1)$ and $B(M_1)$ are the vertex sets A' and B', respectively. We then partition A' and B' such that $A'_{\text{out}} := A' \cap A_{\text{out}}$ and similarly define A'_{in}, A'_*, B'_{out}, B'_{in}, and B'_*.

▶ **Lemma 8.** $|M^* \cap M_1| \geq \frac{1}{2} \cdot (|M_L(A_*)| + |M_R(B_*)|)$.

Proof. By definition, every edge of the matchings $M_L(A_*)$ and $M_R(B_*)$ is incident to an edge of $M^* \cap M_1$. Hence, $|M^* \cap M_1| \geq |M_L(A_*)|$ and $|M^* \cap M_1| \geq |M_R(B_*)|$, which implies the result. ◀

▶ **Lemma 9.** $|M_2| \geq (\frac{1}{2} - \epsilon) \cdot \mu(G) + \frac{1}{2}(|M_L(A_*)| + |M_R(B_*)|)$

Proof. Consider the edges of $M^* \cap G_L$. Every edge of M_L is incident to at most two edges of $M^* \cap G_L$, and every edge of $M_L(A_*)$ is incident to at most one of the edges of $M^* \cap G_L$. By a counting argument, we have that $|M_L| \geq \frac{1}{2}(|M^* \cap G_L| - |M_L(A_*)|) + |M_L(A_*)|$. Then, by Lemma 6, it follows that $|M_L| \geq \frac{1}{2}\left((\frac{1}{2} - \epsilon) \cdot \mu(G) + |M_L(A_*)|\right)$. We similarly bound $|M_R|$ w.r.t. $M_R(B_*)$ and obtain the results since $|M_2| = |M_L| + |M_R|$. ◀

▶ **Lemma 10.** $|P'| = |A'_{\text{in}}| + |B'_{\text{in}}| + |A'_{\text{out}}| + |B'_{\text{out}}| + |A'_*| + |B'_*| = |M_2| - 2|P|$.

Proof. Each length-2 alternating path in P' has an endpoint in either $A(M_1)$ or $B(M_1)$, but not both; thus, we have that $|P'| = |A'| + |B'|$ and the first equality follows by definition of the partitions of A' and B'. For the subsequent equality, we have that every edge of M_L contributes to a length-2 alternating path except for the ones which contribute to length-3 augmenting paths. This gives $|M_L| - |P|$ of them which have an edge in M_L. A similar reasoning w.r.t. M_R shows that $|M_R| - |P|$ of them have an edge in M_R. The result then follows since the paths in P' either have an edge in M_L or in M_R, but never both. ◀

▶ **Lemma 11.** *$|A'_*| \leq |M_R(B_*)|$ and $|B'_*| \leq |M_L(A_*)|$.*

Proof. Consider any vertex in $a \in A'_*$. By definition, a is the endpoint of an edge $(a, b) \in M^* \cap M_1$, which implies that $b \in B_*$. Since a is the endpoint of a path $p \in P'$, we have that $p = (a, b, a_R)$ where (a_R, b) must be an edge of $M_R(B_*)$. Finally, every $a \in A'_*$ has a unique $(a_R, b) \in M_R(B_*)$ since each path in P' is vertex-disjoint and thus the first inequality follows. The second inequality follows similarly w.r.t. B_* and $M_L(A_*)$ instead. ◀

The third query finds a maximal matching M_3 in the graph $G' = G[A' \cup B' \cup \overline{A(M_1)} \cup \overline{B(M_1)}]$, which implies that the size of M_3 is at least half of $\mu(G')$. As such, it is sufficient to bound $\mu(G')$. To that end, we bound the number of edges of M^* in G', which we accomplish by decomposing G' into edge-disjoint subgraphs $G'_L = G[A' \cup \overline{B(M_1)}]$, $G'_{\text{in}} = G[A' \cup B']$ and $G'_R = G[B' \cup \overline{A(M_1)}]$. Using the quantities we have previously bounded, we then obtain our final bound on the size of M_3 as a decreasing function of $|M_1|$, in terms of ϵ, and $|P|$.

▶ **Lemma 12.** *$|M^* \cap G'_L| = |A'_{\text{out}}|$, $|M^* \cap G'_R| = |B'_{\text{out}}|$, and $|M^* \cap G'_{\text{in}}| \geq |A'_{\text{in}}| + |B'_{\text{in}}| - |A_{\text{in}}|$.*

Proof. By definition, every vertex of A'_{out} is incident to an edge in $M^* \cap G'_L$, and vice versa; hence, it holds that $|M^* \cap G'_L| = |A'_{\text{out}}|$. A similar argument shows that $|M^* \cap G'_R| = |B'_{\text{out}}|$ holds. To bound $|M^* \cap G'_{\text{in}}|$, consider an edge $(a, b) \in (M^* \backslash M_1) \cap G_{\text{in}}$. By definition, $a \in A_{\text{in}}$ and $b \in B_{\text{in}}$; however, $(a, b) \in M^* \cap G'_{\text{in}}$ if and only if $a \in A'_{\text{in}}$ and $b \in B'_{\text{in}}$. Thus, there are at most $(|A_{\text{in}}| - |A'_{\text{in}}|) + (|B_{\text{in}}| - |B'_{\text{in}}|)$ edges of $(M^* \backslash M_1) \cap G_{\text{in}}$ which are not in $M^* \cap G'_{\text{in}}$. By definition, it holds that $|(M^* \backslash M_1) \cap G_{\text{in}}| = |B_{\text{in}}|$ and it follows that $|M^* \cap G'_{\text{in}}| \geq |A'_{\text{in}}| + |B'_{\text{in}}| - |A_{\text{in}}|$. ◀

▶ **Lemma 13.** *$|M_3| \geq (\frac{1}{4} - \frac{3\epsilon}{2}) \cdot \mu(G) - |P|$.*

Proof. By rearranging the equation in Lemma 10 and applying Lemma 11, we have that

$$|A'_{\text{in}}| + |B'_{\text{in}}| + |A'_{\text{out}}| + |B'_{\text{out}}| \geq |M_2| - 2|P| - |M_R(B_*)| - |M_L(A_*)|. \tag{2}$$

We subsequently bound $\mu(G')$, which the result follows from since $|M_3| \geq \frac{1}{2} \cdot \mu(G')$.

$$\begin{aligned}
\mu(G') &\geq |M^* \cap G'_L| + |M^* \cap G'_R| + |M^* \cap G'_{\text{in}}| && \text{(edge-disjoint subgraphs)} \\
&\geq |A'_{\text{out}}| + |B'_{\text{out}}| + |A'_{\text{in}}| + |B'_{\text{in}}| - |A_{\text{in}}| && \text{(by Lemma 12)} \\
&\geq |M_2| - 2|P| - |M_L(A_*)| - |M_R(B_*)| - |A_{\text{in}}| && \text{(by Equation (2))} \\
&\geq (\frac{1}{2} - \epsilon) \cdot \mu(G) - \frac{1}{2}(|M_L(A_*)| + |M_R(B_*)|) - 2|P| - |A_{\text{in}}| && \text{(by Lemma 9)} \\
&\geq (\frac{1}{2} - \epsilon) \cdot \mu(G) - |M^* \cap M_1| - 2|P| - |A_{\text{in}}| && \text{(by Lemma 8)} \\
&= (\frac{1}{2} - \epsilon) \cdot \mu(G) - |M^* \cap M_1| - 2|P| - (2\epsilon \cdot \mu(G) - |M^* \cap M_1|) && \text{(by Lemma 7)} \\
&= (\frac{1}{2} - 3\epsilon) \cdot \mu(G) - 2|P|.
\end{aligned}$$ ◀

We are now ready to bound the size of the returned matching M_{out} w.r.t. $\mu(G)$, the size of the maximum matching M^*, thus proving Theorem 1.

▶ **Lemma 14.** *The large matching M_{out} returned by Algorithm 1 is always at least a $\frac{5}{8}$-approximation of the size of a maximum matching in the input graph G.*

Proof.

$$
\begin{aligned}
|M_{\text{out}}| &\geq |M_1| + \frac{1}{2}(|P| + |Q|) && \text{(by Equation (1))}\\
&= (\frac{1}{2} + \epsilon) \cdot \mu(G) + \frac{1}{2}(|P| + |M_3|) && \text{(by Observations 2 and 3)}\\
&\geq (\frac{1}{2} + \epsilon) \cdot \mu(G) + \frac{1}{2}(|P| + (\frac{1}{4} - \frac{3\epsilon}{2}) \cdot \mu(G) - |P|) && \text{(by Lemma 13)}\\
&= (\frac{5}{8} + \frac{\epsilon}{4}) \cdot \mu(G). && \blacktriangleleft
\end{aligned}
$$

3 Lower Bound Results

In this section, we give our lower bounds for edge-query algorithms for MM for up to 3 rounds, showing that our 3-round algorithm is best possible:

▶ **Theorem 15.** *There does **not** exist a deterministic algorithm on a n-vertex input graph for MM in the maximal matching edge-query model that achieves a better than*

1. $\frac{1}{2}$*-approximation in* 1 *round,*
2. $\left(\frac{1}{2} + \frac{2}{n}\right)$*-approximation in* 2 *rounds, and*
3. $\left(\frac{5}{8} + \frac{24}{n}\right)$*-approximation in* 3 *rounds.*

We prove our lower bounds by considering a game between a player, i.e., the algorithm, and an oracle in the edge-query model. The goal of the player is to learn a large matching in the underlying bipartite graph $G = (A, B, E)$ that is adversarially constructed by the oracle along the way. The player initially only knows the vertices A and B and is allowed to query the oracle with any set of edges $Q \subseteq A \times B$ in each round, typically basing the query on any information about G revealed in previous rounds. The oracle then returns an adversarially chosen maximal matching in the subgraph $G[E \cap Q]$, revealing as little information about a large matching as possible. Throughout the game, once information about G is revealed, it may not be altered in subsequent rounds.

Let Q_i be the player's query and let M_i be the maximal matching returned by the oracle in round i. The player learns that the edges M_i are present in G and that the edges in Q_i with both endpoints unmatched by M_i do not exist in G. The player thus learns about both *edges* and *non-edges*. As such, we use *structure graphs* to encapsulate the information known by the player up to graph isomorphisms, providing a simple representation in which to prove our lower bounds – similar to the work by binti Khalil and Konrad [8].

▶ **Definition 16** (Structure Graph [8])**.** *A 4-tuple (A, B, E, F) is a* bipartite structure graph *if E and F are disjoint sets of edges such that (A, B, E) and (A, B, F) are bipartite graphs. The set E corresponds to the set of edges learnt by the algorithm, and the set F corresponds to the set of non-edges learnt.*

A player always begins with the empty structure graph $H_0 = (A, B, E_0, F_0)$ where $E_0 = F_0 = \emptyset$. In a game of r rounds, the structure graphs $H_1, H_2, \ldots, H_r$ represent the information (edges and non-edges) learned by the player after each round, which are based purely on the player's queries $Q_1, Q_2, \ldots, Q_r$ and the oracle's adversarially returned matchings $M_1, M_2, \ldots, M_r$. Consider a player's structure graph H_i w.r.t. H_{i-1} for any $i \in [r]$. It consists of the edges $E_i = E_{i-1} \cup M_i$ and the non-edges $F_i = F_{i-1} \cup N_i$ where $N_i = Q_i \cap (\overline{A(M_i)} \times \overline{B(M_i)})$ ensures that M_i is maximal. The player's information at the end of a round is then necessarily a superset of the information known at the end of the previous round, i.e., $E_i \supseteq E_{i-1}$ and $F_i \supseteq F_{i-1}$. Hence, the structure graph H_i *dominates*

H_{i-1}, which we say in general for any structure graph that is a superset of the information of another up to graph isomorphism. The underlying graph G may then be any graph that is *consistent* with *all* the information H_r revealed to the player by the end of round r.

▶ **Definition 17** (Consistent). *Let $G = (A, B, E)$ be any bipartite graph and let $H_i = (A, B, E_i, F_i)$ be a bipartite structure graph. G is* consistent *with H_i iff $E \supseteq E_i$ and $E \cap F_i = \emptyset$.*

The largest matching a player who knows H_r may output is the maximum matching M_r^{out} in (A, B, E_r). Therefore, the oracle adversarially constructs the graph G so that G is consistent with H_r and so that G has the largest possible maximum matching. This implies that the approximation factor of H_r is $\frac{|M_r^{\text{out}}|}{\mu(G)}$. Note that H_r is strongly dependent on the player's sequence of queries $Q_1, Q_2, ..., Q_r$. Altering even a single query could alter H_r, the player's largest matching M_r^{out} and, most importantly, the approximation factor of H_r. Hence, the goal for proving our lower bound results is to find an upper bound on the approximation factor achieved by any sequence of queries $Q_1, \ldots, Q_r$ for $r = 1, 2$ and 3.

Before proceeding with our analysis, we first present the ideas that we employ to prove our lower bounds. To generally consider all possible queries in each round $i \in [r]$, we allow the oracle to commit to *more* information than is revealed to the player, denoted by the structure graph $\tilde{H}_i$. In particular, we show that $\tilde{H}_i$ dominates the player's structure graph H_i learned regardless of the query Q_i. Then, at the end of round i, the player is assumed to have knowledge of the oracle's structure graph $\tilde{H}_i$. This implies that $\tilde{H}_r$ dominates the structure graph learned by the player for any sequence of queries $Q_1, \ldots, Q_r$. We also allow the oracle to partition the vertices of the graph and consider the vertex-induced subgraph of each part independently. By making the partition a function of the query, we create desirable properties in each part. This, however, does not consider the edges that cross the partition, which are thus asserted as non-edges. Formally, we recombine the structure graphs learned in each part using the *disjoint union* (Definition 18) at the end of round r. Then, the approximation factor of the recombined structure graph follows naturally from the independent parts by Observations 19 and 20.

▶ **Definition 18** (Disjoint Union). *Let (A_x, B_x) and (A_y, B_y) represent an arbitrary partitioning of the vertices A and B into two parts and let $H_x = (A_x, B_x, E_x, F_x)$ and $H_y = (A_y, B_y, E_y, F_y)$ be any bipartite structure graphs. Then, their disjoint union is $H_x \dot\cup H_y = (A, B, E_x \cup E_y, F_x \cup F_y \cup (A_x \times B_y) \cup (A_y \times B_x))$.*

▶ **Observation 19.** *Let H_x and H_y be bipartite structure graphs on disjoint sets of vertices with largest output matchings M_x^{out} and M_y^{out}, respectively. Then, the largest output matching of $H_x \dot\cup H_y$ is of size $|M_x^{out}| + |M_y^{out}|$.*

▶ **Observation 20.** *Let H_x and H_y be bipartite structure graphs on disjoint sets of vertices with consistent graphs G_x and G_y, respectively. Then, there exists a graph G consistent with $H_x \dot\cup H_y$ such that $\mu(G) = \mu(G_x) + \mu(G_y)$.*

We begin our analysis with the following simplifying assumption, where its justification is given in Appendix B for completeness:

▶ **Assumption 21.** *In each round $1 \leq i \leq r$, we assume that the query Q_i does not contain any edges or non-edges already learned by the player.*

Let $A = A_{\text{in}} \cup A_{\text{out}}$ and $B = B_{\text{in}} \cup B_{\text{out}}$ be such that A_{in}, A_{out}, B_{in} and B_{out} are disjoint sets of vertices of size $\frac{n}{4}$ where n is the number of vertices and a multiple of 4. We further assert that $\frac{n}{4}$ is odd. Then, the player begins with only the knowledge of A and B, i.e., the empty structure graph H_0.

First Round. Let $\tilde{M}_1$ be a matching of size $\frac{n}{4}$ that matches A_{in} to B_{in}. We assert its maximality by letting $\tilde{N}_1^{\max} = A_{\text{out}} \times B_{\text{out}}$ be non-edges. Additionally, the non-edges $\tilde{N}_1^{\text{ind}} = (A_{\text{in}} \times B_{\text{in}}) \backslash \tilde{M}_1$ assert that it is an induced matching[2]. Then, we define $\tilde{H}_1 = (A, B, \tilde{E}_1, \tilde{F}_1)$ where $\tilde{E}_1 = \tilde{M}_1$ and $\tilde{F}_1 = \tilde{N}_1 = \tilde{N}_1^{\max} \cup \tilde{N}_1^{\text{ind}}$. See Figure 3 for an illustration.

▶ Remark. $\tilde{H}_1$ can be defined on any subset of vertices $A' \subseteq A$ and $B' \subseteq B$ such that $|A'_{\text{in}}| = |B'_{\text{in}}| = |A'_{\text{out}}| = |B'_{\text{out}}|$. We later use such generalisations, denoted as $\tilde{H}_1(A', B')$.

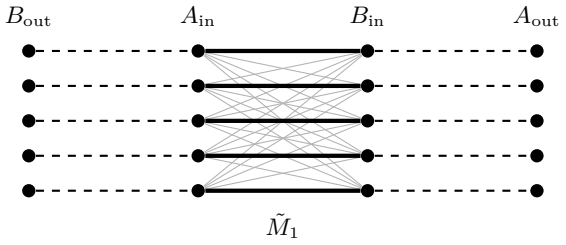


Figure 3 An illustration of structure graph $\tilde{H}_1$. The thick solid black edges represent the matching $\tilde{M}_1$. The non-edges $\tilde{N}_1^{\max}$ are implicit by the layout of the vertices and the non-edges $\tilde{N}_1^{\text{ind}}$ are the grey edges. The dashed black edges are a perfect matching in a worst-case underlying graph.

▶ **Lemma 22.** *Any structure graph H_1 learned by the player is dominated by $\tilde{H}_1$.*

Proof. Let Q_1 be any arbitrary query and let $M^*(Q_1)$ be a maximum matching in the query graph (A, B, Q_1). If $|M^*(Q_1)| \geq |\tilde{M}_1|$, then let $M_1 \subseteq M^*(Q_1)$ be a subset of size $|\tilde{M}_1|$. We assert the maximality of matching M_1 with the non-edges $N_1 = \overline{A(M_1)} \times \overline{B(M_1)}$. Otherwise, $|M^*(Q_1)| < |\tilde{M}_1|$ and we let $M_1 = M^*(Q_1)$, which is trivially a maximal matching among the edges of the query Q_1; hence, $N_1 = \emptyset$.

Finally, in either case, let σ be a graph isomorphism such that $M_1 \subseteq \sigma(\tilde{M}_1)$, which implies that $N_1 \subseteq \sigma(\tilde{N}_1)$. Therefore, the player's H_1 learned is always dominated by $\tilde{H}_1$. ◀

▶ **Lemma 23.** *The approximation factor of the structure graph $\tilde{H}_1$ is $\frac{1}{2}$.*

Proof. The largest matching $\tilde{M}_1^{\text{out}}$ that the player who knows $\tilde{H}_1$ may output is the matching $\tilde{M}_1$, which matches half of the vertices. Since no information regarding the edges in $A_{\text{in}} \times B_{\text{out}}$ or $A_{\text{out}} \times B_{\text{in}}$ has been revealed, we can choose a graph G consistent with $\tilde{H}_1$ that has a perfect matching, which is of size $2 \cdot |\tilde{M}_1|$, thus proving the result. In particular, it has a matching that arbitrarily matches A_{in} to B_{out} and A_{out} to B_{in} (see Figure 3). ◀

Lemma 22 shows that $\tilde{H}_1$ dominates all possible structure graphs learned by the player by the end of round 1, i.e., after query Q_1. Thus, Lemma 23 immediately implies Theorem 15-1.

Second Round. Let $\tilde{M}_L \subseteq (A_{\text{in}} \times B_{\text{out}})$ and $\tilde{M}_R \subseteq (A_{\text{out}} \times B_{\text{in}})$ be matchings of size $\left\lfloor \frac{|\tilde{M}_1|}{2} \right\rfloor$ such that $\tilde{M}_1 \cup \tilde{M}_L \cup \tilde{M}_R$ has no length-3 paths. Let $\tilde{N}_L^{\max} = A_{\text{in}} \backslash A(\tilde{M}_L) \times B_{\text{out}} \backslash B(\tilde{M}_L)$ and $\tilde{N}_R^{\max} = A_{\text{out}} \backslash A(\tilde{M}_R) \times B_{\text{in}} \backslash B(\tilde{M}_R)$ be the non-edges that assert the maximality of the matching $\tilde{M}_2 = \tilde{M}_L \cup \tilde{M}_R$ among the unknown edges. Let $\tilde{N}_L^{\text{ind}} = \big(A(\tilde{M}_L) \times B(\tilde{M}_L)\big) \setminus \tilde{M}_L$ and $\tilde{N}_R^{\text{ind}} = \big(A(\tilde{M}_R) \times B(\tilde{M}_R)\big) \setminus \tilde{M}_R$ be the non-edges required to make the matching

[2] Committing to an induced matching simplifies the arguments in the subsequent rounds without affecting the approximation factor of the player's structure graphs.

induced. Additionally, if $|\tilde{M}_1|$ is odd, then let $e^*_{\text{in}} \in \tilde{M}_1$ be the only edge with both endpoints unmatched by $\tilde{M}_L$ and $\tilde{M}_R$, and similarly let $a_{\text{out}} \in A_{\text{out}}$ and $b_{\text{out}} \in B_{\text{out}}$ be any vertices unmatched by $\tilde{M}_L$ and $\tilde{M}_R$. We assert that e^*_{in} is an isolated edge and that a_{out} and b_{out} are isolated vertices[3], which implies the non-edges $\tilde{N}_* = (A(e^*_{\text{in}}) \times B \setminus B(e^*_{\text{in}})) \cup (A \setminus A(e^*_{\text{in}}) \times B(e^*_{\text{in}})) \cup (A \times \{b_{\text{out}}\}) \cup (\{a_{\text{out}}\} \times B)$. Otherwise, if $|\tilde{M}_1|$ is even, we let $\tilde{N}_* = \emptyset$. Then, we define $\tilde{H}_2 = (A, B, \tilde{E}_1 \cup \tilde{M}_2, \tilde{F}_1 \cup \tilde{N}_2)$ where $\tilde{N}_2 = \tilde{N}^{\max}_L \cup \tilde{N}^{\max}_R \cup \tilde{N}^{\text{ind}}_L \cup \tilde{N}^{\text{ind}}_R \cup \tilde{N}_*$. See Figure 4 for an illustration where the non-edges $\tilde{N}_*$ have been removed for clarity.

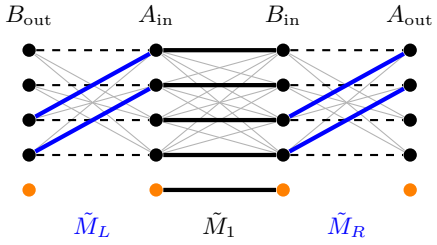


Figure 4 An illustration of the structure graph $\tilde{H}_2$. The thick blue edges represent the matching $\tilde{M}_2$ and the grey ones are the non-edges $\tilde{N}_2 \setminus \tilde{N}_*$. The orange vertices and their incident edge are isolated and only present if $|\tilde{M}_1|$ is odd. The black dashed edges represent a large maximum matching in a worst-case underlying graph.

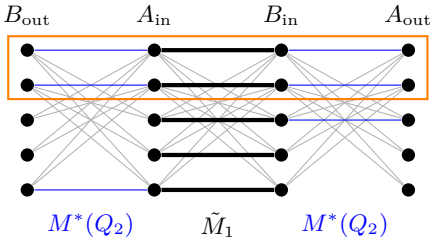


Figure 5 An example of the partitioning based on $M^*(Q_2)$. The blue edges represent the matching $M^*(Q_2)$. The vertices in the orange box represent part (A^+, B^+) and the remaining unboxed ones represent part (A^-, B^-). The grey edges show the additional non-edges asserted by the disjoint union.

Let Q_2 be an arbitrary query of round 2 and let $M^*(Q_2)$ be a maximum matching of (A, B, Q_2). By Assumption 21, we have that Q_2 only has edges in $A_{\text{in}} \times B_{\text{out}}$ and $A_{\text{out}} \times B_{\text{in}}$, which implies that the edges of $M^*(Q_2)$ either form vertex-disjoint length-3 or length-2 alternating paths w.r.t. $\tilde{M}_1$. As such, we partition the vertices of A and B to consider the length-3 and length-2 paths separately: Part (A^+, B^+) with N^+ many vertices consists of all vertices that lie on a length-3 path, whereas part (A^-, B^-) consists of the remaining $N^- = n - N^+$ many vertices, whose induced subgraph includes all the length-2 paths. See Figure 5 for an example of the partitioning.

▶ **Lemma 24.** *$\tilde{H}_1$ is partitioned w.r.t. parts (A^+, B^+) and (A^-, B^-) into structure graphs $\tilde{H}^+_1 = \tilde{H}_1(A^+, B^+)$ and $\tilde{H}^-_1 = \tilde{H}_1(A^-, B^-)$, respectively, where $\tilde{H}^+_1 \dot\cup \tilde{H}^-_1$ dominates $\tilde{H}_1$.*

With Lemma 24, whose proof is given in Appendix C for completeness, the player's information in each part at the start of round 2 is exactly the respective generalisations of $\tilde{H}_1$. As such, we show that the respective generalisations of $\tilde{H}_2$ dominate the player's structure graphs H^+_2 and H^-_2 learned in each part after query Q_2.

▶ **Lemma 25.** *Any structure graph H^+_2 learned by the player is dominated by $\tilde{H}^+_2 = \tilde{H}_2(A^+, B^+)$.*

▶ **Lemma 26.** *Any structure graph H^-_2 learned by the player is dominated by $\tilde{H}^-_2 = \tilde{H}_2(A^-, B^-)$.*

[3] Committing to the isolated edge and vertices simplifies the arguments in the subsequent round without affecting the approximation factor of the player's structure graphs.

Lemmas 25 and 26 are proven similarly to Lemma 22; hence, we give their proofs in Appendix C to save space. By Lemmas 25 and 26, the player's overall information at the end of round 2 is dominated by the structure graph $\tilde{H}_2^+ \dot\cup \tilde{H}_2^-$.

▶ **Lemma 27.** *The approximation factor of the structure graph $\tilde{H}_2^+ \dot\cup \tilde{H}_2^-$ is at most $\frac{1}{2} + \frac{2}{n}$.*

Proof. We claim that the largest matching is of size $|\tilde{M}_1|$ and that there exists a consistent graph G such that $\mu(G) = 2 \cdot |\tilde{M}_1| - 1$. Then, the approximation factor of $\tilde{H}_2^+ \dot\cup \tilde{H}_2^-$ is at most $\frac{|\tilde{M}_1|}{2\cdot|\tilde{M}_1|-1} = \frac{1}{2} + \frac{1}{4\cdot|\tilde{M}_1|-2} \leq \frac{1}{2} + \frac{1}{2\cdot|\tilde{M}_1|} = \frac{1}{2} + \frac{2}{n}$ for large enough n.

We now prove the first claim. Since there are no augmenting paths in $\tilde{M}_1^+ \cup \tilde{M}_2^+$ or $\tilde{M}_1^- \cup \tilde{M}_2^-$, the largest output matching is $\tilde{M}_2^{\text{out}} = \tilde{M}_1^+ \cup \tilde{M}_1^- = \tilde{M}_1$ by Observation 19. It remains to prove the second claim. Since $|\tilde{M}_1| = \frac{n}{4}$ is odd, w.l.o.g., $|\tilde{M}_1^+|$ is odd and $|\tilde{M}_1^-|$ is even; hence, we have that $|\tilde{M}_2^+| = |\tilde{M}_1^+| - 1$ and $|\tilde{M}_2^-| = |\tilde{M}_1^-|$. Since both matchings in both parts are maximal and induced, the only unknown edges are those in $(A_{\text{in}} \times B_{\text{out}}) \cup (A_{\text{out}} \times B_{\text{in}})$ that have only one endpoint matched by $\tilde{M}_2^+$ or $\tilde{M}_2^-$ and are not incident to the isolated edge or vertices. Thus, we may use these to construct consistent graphs in $\tilde{H}_2^+$ and $\tilde{H}_2^-$ with maximum matchings of size $2 \cdot |\tilde{M}_2^+| + 1 = 2 \cdot |\tilde{M}_1^+| - 1$, which includes the isolated edge, and $2 \cdot |\tilde{M}_2^-| = 2 \cdot |\tilde{M}_1^-|$, respectively. By Observation 20, this gives the graph G as required. ◀

Overall, we have that any arbitrary query Q_2 can be used to construct the relevant partition of the vertices where Lemmas 25–27 always hold, thus proving Theorem 15-2.

Third Round. We continue to consider the partition w.r.t. the query Q_2 where the player now knows generalisations of $\tilde{H}_2$ in each part. As such, it is sufficient to find a structure graph $\tilde{H}_3$ that dominates $\tilde{H}_2$ and then apply the disjoint union as before. Note that we consider only the even case of $\tilde{H}_2$ since, by Assumption 21, the endpoints of the isolated edge e^*_{in} and the isolated vertices a_{out} and b_{out} in the odd case of $\tilde{H}_2$ (see Figure 4) are not endpoints of any edge in a third round query, thus reducing it to an even case of $\tilde{H}_2$.

With knowledge of an even case of $\tilde{H}_2$ at the start of round 3, the player is aware of two edge-disjoint maximal and induced matchings $\tilde{M}_1$ and $\tilde{M}_2$, both of which are half the size of a perfect matching, such that $\tilde{M}_1 \cup \tilde{M}_2$ is exactly the edges of $|\tilde{M}_1|$ many vertex-disjoint length-2 paths, denoted by P. Therefore, by Assumption 21, any third round query may contain only the edges that either (a) *extend* a path in P, denoted by K^{ext}, or (b) provide a *replacement* edge for a path in P, denoted by K^{rep}. Observe that the *extending* or *replacement* edges are either incident to A_{in} (on the left) or incident to B_{in} (on the right). As such, we define the set of possible query edges as a union of (left and right) complete graphs, respectively:

$$K^{\text{ext}} := K_L^{\text{ext}} \cup K_R^{\text{ext}} = (A_{\text{in}}(P) \times B_{\text{out}}(P)) \cup (A_{\text{out}}(P) \times B_{\text{in}}(P)),$$
$$K^{\text{rep}} := K_L^{\text{rep}} \cup K_R^{\text{rep}} = \left(\overline{A_{\text{in}}(P)} \times \overline{B_{\text{out}}(P)}\right) \cup \left(\overline{A_{\text{out}}(P)} \times \overline{B_{\text{in}}(P)}\right)$$

where, for any set of vertices U, $U(P)$ denotes the U-endpoints of paths in P and $\overline{U(P)} := U \backslash U(P)$. We illustrate this in Figure 6.

Let Q_3 be an arbitrary query of round 3. We partition the vertices A and B to consider the paths in P that form vertex-disjoint 6-cycles with edges in $Q_3^{\text{ext}} = Q_3 \cap K^{\text{ext}}$ separately from the ones that do not: Part (A°, B°) with N° many vertices consists of all vertices that lie on the vertex-disjoint 6-cycles, including, for each 6-cycle, a vertex from $\overline{A_{\text{out}}(P)}$ and one from $\overline{B_{\text{out}}(P)}$, whereas part $(A^{\overline{\circ}}, B^{\overline{\circ}})$ consists of the remaining $N^{\overline{\circ}} = n - N^\circ$ many vertices, whose induced subgraph includes all the paths in P that do not form any 6-cycles with each other using the query edges Q_3^{ext}. See Figure 7 for an example of the partitioning.

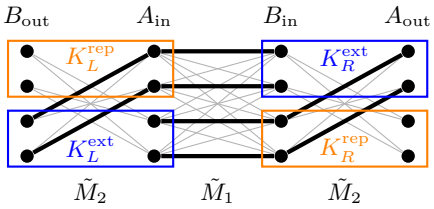


Figure 6 An illustration of the possible query edges by a player who knows $\tilde{H}_2$. The black edges represent the induced maximal matchings $\tilde{M}_1$ and $\tilde{M}_2$ and the grey ones are their corresponding non-edges $\tilde{N}_1$ and $\tilde{N}_2$. The possible query edges K_L^{ext}, K_R^{ext}, K_L^{rep} and K_R^{rep} are complete graphs on the vertices of their respective boxes.

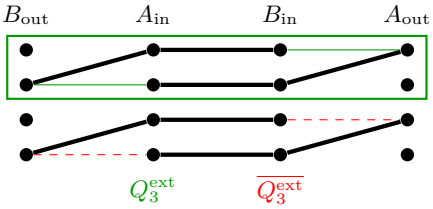
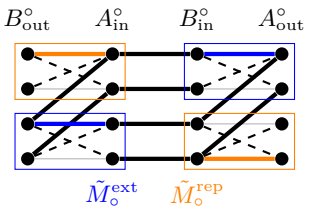


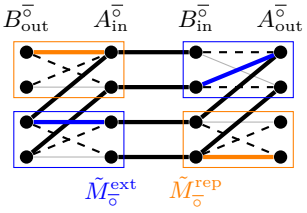
Figure 7 An example of the partitioning based on Q_3^{ext}. The green edges represent the edges necessarily in Q_3^{ext} and the dashed red edges represent the edges necessarily not in Q_3^{ext}, i.e., the edges in $\overline{Q_3^{\text{ext}}} = K^{\text{ext}} \setminus Q_3^{\text{ext}}$. The vertices in the green box represent part (A°, B°) and the remaining unboxed ones represent part $(A^{\overline{\circ}}, B^{\overline{\circ}})$.

▶ **Lemma 28.** *$\tilde{H}_2$ is partitioned w.r.t. parts (A°, B°) and $(A^{\overline{\circ}}, B^{\overline{\circ}})$ into structure graphs $\tilde{H}_2^\circ = \tilde{H}_2(A^\circ, B^\circ)$ and $\tilde{H}_2^{\overline{\circ}} = \tilde{H}_2(A^{\overline{\circ}}, B^{\overline{\circ}})$, respectively, where $\tilde{H}_2^\circ \dot\cup \tilde{H}_2^{\overline{\circ}}$ dominates $\tilde{H}_2$.*

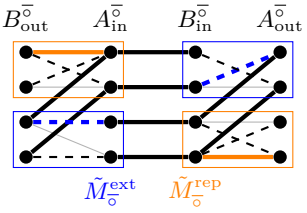
With Lemma 28, whose proof is given in Appendix C for completeness, the player's information in each part at the start of round 3 is exactly the respective generalisations of $\tilde{H}_2$. We show that the structure graphs H_3° and $H_3^{\overline{\circ}}$ learned by the player in each part are dominated by distinct structure graphs $\tilde{H}_3^\circ$ and $\tilde{H}_3^{\overline{\circ}}$, respectively, after query Q_3. We defer the corresponding proofs of Lemmas 29 and 30 to Appendix C to save space.



(a) A gadget of the structure graph $\tilde{H}_3^\circ$.



(b) First case of a gadget of the structure graph $\tilde{H}_3^{\overline{\circ}}$.



(c) Second case of a gadget of the structure graph $\tilde{H}_3^{\overline{\circ}}$.

Figure 8 Illustrations of the gadgets of the structure graphs $\tilde{H}_3^\circ$ and $\tilde{H}_3^{\overline{\circ}}$, respectively. The thick blue edges (solid and dashed) represent the matchings $\tilde{M}_\circ^{\text{ext}}$ and $\tilde{M}_{\overline{\circ}}^{\text{ext}}$. The thick orange edges represent the matchings $\tilde{M}_\circ^{\text{rep}}$ and $\tilde{M}_{\overline{\circ}}^{\text{rep}}$. The grey edges represent the non-edges $\tilde{N}_\circ^{\max}$ and $\tilde{N}_{\overline{\circ}}^{\max}$. The dashed edges (black and blue) represent maximum matchings in worst-case underlying graphs.

Let P° be the set of the $|\tilde{M}_1^\circ|$ many vertex-disjoint length-2 paths in part (A°, B°). Let $\mathcal{G}^\circ$ be a collection of at least $\frac{|\tilde{M}_1^\circ|}{4} - 1$ many vertex-disjoint gadgets such that each gadget is made up of two distinct vertices from $\overline{A_{\text{out}}^\circ(P^\circ)}$ and two from $\overline{B_{\text{out}}^\circ(P^\circ)}$, and four distinct paths from P° where two have their endpoints in A° while the other two have theirs in B°. Let $\tilde{M}_\circ^{\text{ext}} \subseteq K_\circ^{\text{ext}}$ be a matching such that each gadget has two edges that from a 6-cycle with two of the paths. Let $\tilde{M}_\circ^{\text{rep}} \subseteq K_\circ^{\text{rep}}$ be another matching such that each gadget has two edges, one from K_L^{rep} and one from K_R^{rep}, where only one of them is incident to the 6-cycle. Let the inter-gadget edges of $K_\circ^{\text{ext}} \cup K_\circ^{\text{rep}}$ be the non-edges $\tilde{N}_\circ^{\text{gad}}$. We assert the maximality of the matching $\tilde{M}_3^\circ = \tilde{M}_\circ^{\text{ext}} \cup \tilde{M}_\circ^{\text{rep}}$ by committing the edges of $K_\circ^{\text{ext}} \cup K_\circ^{\text{rep}}$ within each gadget that have both endpoints unmatched by $\tilde{M}_3^\circ$ to be the non-edges $\tilde{N}_\circ^{\max}$. Then, we define $\tilde{H}_3^\circ = (A^\circ, B^\circ, \tilde{E}_2^\circ \cup \tilde{M}_3^\circ, \tilde{F}_2^\circ \cup \tilde{N}_3^\circ)$ where $\tilde{N}_3^\circ = \tilde{N}_\circ^{\text{gad}} \cup \tilde{N}_\circ^{\max}$. See Figure 8a for an illustration of a single gadget.

▶ **Lemma 29.** *Any structure graph H_3° learned by the player is dominated by $\tilde{H}_3^\circ$.*

Let $P^{\overline{\circ}}$ be the set of the $|\tilde{M}_1^{\overline{\circ}}|$ many vertex-disjoint length-2 paths in part $(A^{\overline{\circ}}, B^{\overline{\circ}})$. Let $\mathcal{G}^{\overline{\circ}}$ be a collection of at least $\frac{|\tilde{M}_1^{\overline{\circ}}|}{4} - \frac{1}{2}$ many vertex-disjoint gadgets such that each gadget is made up of two distinct vertices from $\overline{A^{\overline{\circ}}_{\text{out}}(P^{\overline{\circ}})}$ and two from $\overline{B^{\overline{\circ}}_{\text{out}}(P^{\overline{\circ}})}$, and four distinct paths from $P^{\overline{\circ}}$ where two have their endpoints in $A^{\overline{\circ}}$ while the other two have theirs in $B^{\overline{\circ}}$. Let $\tilde{M}_{\overline{\circ}}^{\text{ext}} \subseteq K_{\overline{\circ}}^{\text{ext}}$ be a matching such that each gadget has two edges that form a length-8 path with three of the paths, leaving the other path unmatched. Let $\tilde{M}_{\overline{\circ}}^{\text{rep}} \subseteq K_{\overline{\circ}}^{\text{rep}}$ be another matching such that each gadget has two edges, one from K_L^{rep} and one from K_R^{rep}, where one of them is incident to the path unmatched by $\tilde{M}_{\overline{\circ}}^{\text{ext}}$. Let the inter-edges of $K_{\overline{\circ}}^{\text{ext}} \cup K_{\overline{\circ}}^{\text{rep}}$ be the non-edges $\tilde{N}_{\overline{\circ}}^{\text{gad}}$. We assert the maximality of matching $\tilde{M}_{\overline{\circ}}^{\text{rep}}$ among the edges of $K_{\overline{\circ}}^{\text{rep}}$ in each gadget by committing its edges that have both endpoints unmatched by $\tilde{M}_{\overline{\circ}}^{\text{rep}}$ to be the non-edges $\tilde{N}_{\overline{\circ}}^{\text{rep}}$. We assert the maximality of the matching $\tilde{M}_{\overline{\circ}}^{\text{ext}}$ among the edges of $K_{\overline{\circ}}^{\text{ext}}$ with the non-edges $\tilde{N}_{\overline{\circ}}^{\text{ext}}$ in two distinct cases of a gadget: $\tilde{N}_{\overline{\circ}}^{\text{ext}}$ is such that each gadget either has (1) the only two edges of $K_{\overline{\circ}}^{\text{ext}}$ with both endpoints unmatched by $\tilde{M}_{\overline{\circ}}^{\text{ext}}$, or (2) two edges of $K_{\overline{\circ}}^{\text{ext}}$ that form a length-8 path and are both incident to only the A-vertices or only the B-vertices of $\tilde{M}_{\overline{\circ}}^{\text{ext}}$. Note that the gadgets in the latter case are indeed maximal since, by construction of part $(A^{\overline{\circ}}, B^{\overline{\circ}})$, there are no 6-cycles among its query edges. Then, we define $\tilde{H}_3^{\overline{\circ}} = (A^{\overline{\circ}}, B^{\overline{\circ}}, \tilde{E}_2^{\overline{\circ}} \cup \tilde{M}_3^{\overline{\circ}}, \tilde{F}_2^{\overline{\circ}} \cup \tilde{N}_3^{\overline{\circ}})$ where $\tilde{N}_3^{\overline{\circ}} = \tilde{N}_{\overline{\circ}}^{\text{gad}} \cup \tilde{N}_{\overline{\circ}}^{\max}$. See Figures 8b and 8c for illustrations of the two cases of a gadget.

▶ **Lemma 30.** *Any structure graph $H_3^{\overline{\circ}}$ learned by the player is dominated by $\tilde{H}_3^{\overline{\circ}}$.*

By Lemmas 29 and 30, the player, whose structure graph is $\tilde{H}_2$ at the start of round 2, has its structure graph dominated by $\tilde{H}_3^\circ \dot\cup \tilde{H}_3^{\overline{\circ}}$ by the end of round 3. Note that, since there are no inter-gadget edges, $\tilde{H}_3^\circ \dot\cup \tilde{H}_3^{\overline{\circ}}$ is made up of the collection of gadgets $\mathcal{G} = \mathcal{G}^\circ \cup \mathcal{G}^{\overline{\circ}}$. As such, each gadget is either a gadget from $\tilde{H}_3^\circ$, the first case gadget from $\tilde{H}_3^{\overline{\circ}}$, or the second case gadget from $\tilde{H}_3^{\overline{\circ}}$. It follows then that there are at least $\frac{|\tilde{M}_1^\circ|}{4} - 1 + \frac{|\tilde{M}_1^{\overline{\circ}}|}{4} - \frac{1}{2} = \frac{N^\circ}{16} + \frac{N^{\overline{\circ}}}{16} - \frac{3}{2} = \frac{|\tilde{M}_1|}{4} - \frac{3}{2}$ many gadgets in $\mathcal{G}$ and, since each gadget has exactly four edges of $\tilde{M}_1 = \tilde{M}_1^\circ \cup \tilde{M}_1^{\overline{\circ}}$, there are at most $\frac{|\tilde{M}_1|}{4}$ many gadgets in $\mathcal{G}$.

▶ **Lemma 31.** *The largest matching in $\tilde{H}_3^\circ \dot\cup \tilde{H}_3^{\overline{\circ}}$ is of size at most $\frac{5 \cdot |\tilde{M}_1|}{4}$.*

Proof. Since each gadget is vertex-disjoint, we only need to consider the number of edges that each case of a gadget contributes to a largest output matching $\tilde{M}_3^{\text{out}}$. By Berge's theorem or similar, every edge with an endpoint of degree 1 is included in a largest output matching and thus it is easy to see that each gadget contributes exactly 5 edges to $\tilde{M}_3^{\text{out}}$. Finally, any edge of $\tilde{M}_1$ not included in a gadget contributes fewer edges, hence; we assume that all form part of a gadget, which implies that $|\mathcal{G}| = \frac{|\tilde{M}_1|}{4}$ and the result. ◀

▶ **Lemma 32.** *There exists a graph consistent with $\tilde{H}_3^\circ \dot\cup \tilde{H}_3^{\overline{\circ}}$ that has a maximum matching of size at least $2 \cdot |\tilde{M}_1| - 12$.*

Proof. Any graph consistent with $\tilde{H}_3^\circ \dot\cup \tilde{H}_3^{\overline{\circ}}$ may only consist of the edges within the vertex-disjoint gadgets. As such, we construct a maximum matching w.r.t. to each case of a gadget independently. Observe that all cases of a gadget, on 16 vertices, can be partitioned into 4 vertex-disjoint parts such that each consists of two A-vertices and two B-vertices, and they respectively correspond to the edges K_L^{ext}, K_R^{ext}, K_L^{rep} and K_R^{rep} unknown to the player who knows $\tilde{H}_2$. After query Q_3, at most a single non-edge f in each part is learned; hence, the remaining two edges incident to f can be used to construct a perfect matching in each part, which is thus a perfect matching, of size 8, in each gadget. Finally, we have that, since there are $|\mathcal{G}| \geq \frac{|\tilde{M}_1|}{4} - \frac{3}{2}$ many vertex-disjoint gadgets, the result holds. ◀

Overall, we have that the structure graph learned by the player after round 1 is dominated by $\tilde{H}_1$, for any arbitrary query Q_1. Then, in round 2, any arbitrary query Q_2 made by the player partitions the vertices into parts (A^+, B^+) and (A^-, B^-) such that the structure graphs learned in each part is dominated by the respective generalisations of $\tilde{H}_2$, i.e., $\tilde{H}_2^+$ and $\tilde{H}_2^-$, by the end of round 2. In round 3, each part is dominated by the respective generalisations of $\tilde{H}_3^\circ \dot\cup \tilde{H}_3^\circ$, which we denote as $\tilde{H}_3^+$ and $\tilde{H}_3^-$, respectively, for any arbitrary query Q_3. As such, the player's overall information by the end of round 3 is $\tilde{H}_3^+ \dot\cup \tilde{H}_3^-$ for any arbitrary sequence of queries Q_1, Q_2, Q_3. Finally, we prove Lemma 33, which implies Theorem 15-3.

▶ **Lemma 33.** *The approximation factor of the structure graph $\tilde{H}_3^+ \dot\cup \tilde{H}_3^-$ is at most $\frac{5}{8} + \frac{24}{n}$.*

Proof. Recall that $\frac{n}{4}$ is odd; hence, $\tilde{H}_2^+$ and $\tilde{H}_2^-$ are such that, w.l.o.g., $|\tilde{M}_1^+|$ is even and $|\tilde{M}_1^-|$ is odd, and are dominated be the respective generalisations of $\tilde{H}_3^\circ \dot\cup \tilde{H}_3^\circ$, that is, $\tilde{H}_3^+$ and $\tilde{H}_3^-$. As such, we consider both cases, particularly paying attention to $\tilde{H}_3^-$ and considering its isolated edge and vertices. It follows by Lemma 31 that $\tilde{H}_3^+$ has a largest matching of size at most $\frac{5\cdot|\tilde{M}_1^+|}{4}$; however, $\tilde{H}_3^-$ has one of size at most $\frac{5\cdot|\tilde{M}_1^-|}{4} - \frac{1}{4}$. Then, Lemma 32 immediately implies that $\tilde{H}_3^+$ has a consistent graph G_3^+ such that $\mu(G_3^+) \geq 2\cdot|\tilde{M}_1^+| - 12$; however, it implies that $\tilde{H}_3^-$ has a consistent graph G_3^- such that $\mu(G_3^-) \geq 2\cdot|\tilde{M}_1^-| - 13$. Finally, by Observations 19 and 20, the approximation factor of $\tilde{H}_3^+ \dot\cup \tilde{H}_3^-$ is at most $\frac{\frac{5}{4}|\tilde{M}_1| - \frac{1}{4}}{2\cdot|\tilde{M}_1| - 25} \leq \frac{5}{8} + \frac{8}{|\tilde{M}_1|} = \frac{5}{8} + \frac{24}{n}$ for large enough n. ◀

4 Conclusion

In this paper, we gave tight results on the approximation factor achievable by deterministic algorithms in the maximal matching edge-query model for up to 3 rounds. Our main result is a 0.625-approximation algorithm for MBM, which operates in three query rounds, and we proved that this is best possible. This algorithm can be implemented in the semi-streaming model and constitutes an improvement over the previously best 3-pass algorithm with approximation factor 0.6111 by Feldman and Szarf [13]. The best approximation factors achievable in one and two rounds are $\frac{1}{2}$ and $\frac{1}{2} + \Theta(\frac{1}{n})$, respectively, even in general graphs.

We conclude with three open questions:

1. *Randomization.* Our paper only considers deterministic query algorithms. Does randomization allow us to improve upon the results obtained in this paper?
2. *Adaptivity.* The algorithms considered in this paper are *adaptive* in the sense that the ith query can depend on the maximal matchings returned in rounds $1, \ldots, i-1$. Can we obtain interesting results if we allow multiple *non-adaptive* queries, i.e., queries that do not depend on the output produced by other queries?
3. *Semi-streaming Algorithms.* Is there a 3-pass semi-streaming algorithm for MBM with approximation factor better than 0.625 (that necessarily cannot be implemented as a deterministic edge-query algorithm)?

References

1 Raghavendra Addanki, Andrew McGregor, and Cameron Musco. Non-adaptive edge counting and sampling via bipartite independent set queries. *CoRR*, abs/2207.02817, 2022. doi:10.48550/arXiv.2207.02817.

2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013. doi:10.1016/j.ic.2012.10.006.

3 Sepehr Assadi, Deeparnab Chakrabarty, and Sanjeev Khanna. Graph Connectivity and Single Element Recovery via Linear and OR Queries. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2021.7.

4 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9–12, 2022*, pages 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.

5 Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 165–171. SIAM, 2021. doi:10.1137/1.9781611976496.18.

6 Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algorithms*, 16(4):52:1–52:27, 2020. doi:10.1145/3404867.

7 Soheil Behnezhad. Time-optimal sublinear algorithms for matching and vertex cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 873–884. IEEE, 2021. doi:10.1109/FOCS52979.2021.00089.

8 Lidiya Khalidah binti Khalil and Christian Konrad. Constructing large matchings via query access to a maximal matching oracle. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 26:1–26:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.26.

9 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1-2):490–508, 2012. doi:10.1007/s00453-011-9556-8.

10 Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Finding large matchings in semi-streaming. In Carlotta Domeniconi, Francesco Gullo, Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain*, pages 608–614. IEEE Computer Society, 2016. doi:10.1109/ICDMW.2016.0092.

11 Uriel Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM J. Comput.*, 35(4):964–984, 2006. doi:10.1137/S0097539704447304.

12 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/j.tcs.2005.09.013.

13 Moran Feldman and Ariel Szarf. Maximum matching sans maximal matching: A new approach for finding maximum matchings in the data stream model. In Amit Chakrabarti and Chaitanya Swamy, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*, volume 245 of *LIPIcs*, pages 33:1–33:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.APPROX/RANDOM.2022.33.

14 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008. doi:10.1002/rsa.20203.

15 Sagar Kale and Sumedh Tirodkar. Maximum matching in two, three, and a few more passes over graph streams. In Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, volume 81 of *LIPIcs*, pages 15:1–15:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.15.

16 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 1874–1893. SIAM, 2021. doi:10.1137/1.9781611976465.112.

17 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010. doi:10.1137/1.9781611973075.76.

18 Christian Konrad. A simple augmentation method for matchings with applications to streaming algorithms. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 74:1–74:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.74.

19 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.

20 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014. doi:10.1145/2627692.2627694.

A Proofs of Lemmas 4–6

▶ **Lemma 4.** *There exists a maximum matching M^* such that $M^* \cup M_1$ has no even-length alternating paths or cycles.*

Proof. Let $P(M^*)$ be the set of even length alternating paths or cycles in $M^* \cup M_1$. If it is non-empty, we can use any $p \in P(M^*)$ to find another maximum matching $\hat{M}^*$.

Observe first that the edges of p alternate between edges in M_1 and M^* and is of even length. Therefore, we construct $\hat{M}^*$ from M^*, without decreasing its size, by replacing the edges of $M^* \cap p$ with the edges of $M_1 \cap p$. Finally, $\hat{M}^*$ is indeed a maximum matching since the edges of $M_1 \cap p$ are vertex disjoint from the edges of $M^* \backslash (M^* \cap p)$, otherwise p would not be an even length alternating path.

The only difference between the edges in $M^* \cup M_1$ and $\hat{M}^* \cup M_1$ are the removed edges $M^* \cap p$ which means that p is no longer an alternating path in $\hat{M}^* \cup M_1$ whereas all others remain unchanged. Therefore, $|P(\hat{M}^*)| = |P(M^*)| - 1$. Repeating this process until no such paths exist produces a maximum matching where the claim holds. ◀

▶ **Lemma 5.** *Each vertex of $A(M_1)$ and $B(M_1)$ is the endpoint of an edge in M^*.*

Proof. Since M^* is a maximal matching, every edge of M_1 must be incident to at least one of its edges. By Lemma 4, there are no even length alternating paths in $M^* \cup M_1$; hence, every $(a, b) \in M_1$ is either an edge of M^* or part of an augmenting path in $M^* \cup M_1$. In either case, a and b are each endpoints of exactly one edge of M^*. ◀

▶ **Lemma 6.** $|M^* \cap G_L| = |M^* \cap G_R| = (\frac{1}{2} - \epsilon) \cdot \mu(G)$.

Proof. Firstly, every edge of $M^* \oplus M_1$ is part of an alternating path or cycle since M_1 is maximal. Then, by Lemma 4, there are only odd length alternating paths, i.e., augmenting paths. Finally, any augmenting path must begin with an edge of M^* in G_L and end with one in G_R (or vice versa) with all other edges along the path belonging to G_{in}. See Figure 2 for an example. Thus, the number of edges of M^* in G_L and G_R, respectively, is the number of vertex-disjoint augmenting paths, which we subsequently show is exactly $(\frac{1}{2} - \epsilon) \cdot \mu(G)$ and thus implies the result.

Since M_1 and M^* are maximal matchings, every edge of M^* must be incident to at least one edge of M_1, and vice versa. In the case that $e \in M^* \cap M_1$, both these conditions are satisfied and e is an isolated edge in $M^* \cup M_1$ as both are matchings. Hence, in all other cases, the edges of M^* and M_1 belong to an alternating path in $M^* \oplus M_1$, all of which are necessarily vertex-disjoint since no two edges from a matching may share the same endpoint. Then, by Lemma 4, these are necessarily (odd length) augmenting paths. Finally, since M^* is a maximum matching of size $\mu(G)$ and each vertex-disjoint augmenting path increases the size of M_1 by 1, there must be exactly $\mu(G) - |M_1|$ many paths and the claim follows by Observation 3. ◀

B Reason for Assumption 21

▶ **Assumption 21.** *In each round $1 \leq i \leq r$, we assume that the query Q_i does not contain any edges or non-edges already learned by the player.*

Reason. Let $H = (A, B, E, F)$ be the structure graph known by the player. Let $e \in E$ and $f \in F$. If $e \in Q_i$, then the oracle can add e to the returned matching M_i without revealing any information about the edges incident to e that the player could have otherwise learned; thus, the query $Q_i \backslash \{e\}$ could never reveal less information than Q_i. If $f \in Q_i$, then f would never be in M_i; hence, $Q_i \backslash \{f\}$ would be an equivalent query. ◀

C Proofs of Lemmas 24–26 and 28–30

▶ **Lemma 24.** *$\tilde{H}_1$ is partitioned w.r.t. parts (A^+, B^+) and (A^-, B^-) into structure graphs $\tilde{H}_1^+ = \tilde{H}_1(A^+, B^+)$ and $\tilde{H}_1^- = \tilde{H}_1(A^-, B^-)$, respectively, where $\tilde{H}_1^+ \dot{\cup} \tilde{H}_1^-$ dominates $\tilde{H}_1$.*

Proof. Consider part (A^+, B^+) and its vertex-induced subgraph. Since (A^+, B^+) consists of the vertices of length-3 alternating paths w.r.t. $\tilde{M}_1$, the inclusion of the vertices of each path includes a matching edge from $\tilde{M}_1$ and two unmatched vertices, one in A_{out} and one in B_{out}. Thus, the matching edges $\tilde{M}_1^+ = \tilde{M}_1 \cap (A^+ \times B^+)$ and the non-edges $\tilde{N}_1^+ = \tilde{N}_1 \cap (A^+ \times B^+)$ are exactly the edges and non-edges of $\tilde{H}_1(A^+, B^+)$. Part (A^-, B^-) follows similarly since the remaining vertex-induced subgraph must have two unmatched vertices for every matching edge. Finally, since no edges of $\tilde{E}_1$ cross the partition, the disjoint union of each part dominates the original structure graph. ◀

▶ **Lemma 25.** *Any structure graph H_2^+ learned by the player is dominated by $\tilde{H}_2^+ = \tilde{H}_2(A^+, B^+)$.*

Proof. Let $Q_2^+ := Q_2 \cap (A^+ \times B^+)$ be the query edges relevant to part (A^+, B^+). By construction of the partition, $M^*(Q_2) \cap (A^+ \times B^+) \subseteq Q_2^+$ is a perfect matching in (A^+, B^+) such that every edge in $\tilde{M}_1^+$ is incident to two edges in $M^*(Q_2)$. Let W_L^+ be the perfect matching edges incident to $A_{\text{in}}^+ = A(\tilde{M}_1^+)$ and let W_R^+ be the ones incident to $B_{\text{in}}^+ = B(\tilde{M}_1^+)$.

As such, $\tilde{M}_1^+ \cup W_L^+ \cup W_R^+$ is exactly the edges of the the $|\tilde{M}_1^+|$ many vertex-disjoint length-3 paths used to construct part (A^+, B^+). Therefore, we can construct matchings $M_L^+ \subseteq W_L^+$ and $M_R^+ \subseteq W_R^+$ of size $\left\lfloor \frac{|\tilde{M}_1^+|}{2} \right\rfloor$ such that $\tilde{M}_1^+ \cup M_L^+ \cup M_R^+$ has *no* length-3 paths. We then assert the maximality of $M_2^+ = M_L^+ \cup M_R^+$ by committing $N_L^+ = A_{\text{in}}^+ \backslash A(M_L^+) \times B_{\text{out}}^+ \backslash B(M_L^+)$ and $N_R^+ = A_{\text{out}}^+ \backslash A(M_R^+) \times B_{\text{in}}^+ \backslash B(M_R^+)$ as non-edges.

Finally, we let σ be a graph isomorphism[4] that relates this to $\tilde{H}_2^+$ where $M_L^+ = \sigma(\tilde{M}_L^+)$ and $M_R^+ = \sigma(\tilde{M}_R^+)$. Then, we have that $M_2^+ \subseteq \sigma(\tilde{M}_2^+)$ and $N_2^+ = N_L^+ \cup N_R^+ \subseteq \sigma(\tilde{N}_2^+)$. ◀

▶ **Lemma 26.** *Any structure graph H_2^- learned by the player is dominated by $\tilde{H}_2^- = \tilde{H}_2(A^-, B^-)$.*

Proof. Let $Q_2^- := Q_2 \cap (A^- \times B^-)$ be the query edges relevant to part (A^-, B^-). By construction of the partition, $M^*(Q_2) \cap (A^- \times B^-) \subseteq Q_2^-$ is a maximum matching in (A^-, B^-) such that every edge in $\tilde{M}_1^-$ is incident to at most one edge in $M^*(Q_2)$. Let W_L^- be the matching edges incident to $A_{\text{in}}^- = A(\tilde{M}_1^-)$ and let W_R^- be the ones incident to $B_{\text{in}}^- = B(\tilde{M}_1^-)$. If $|W_L^-| \geq \left\lfloor \frac{|\tilde{M}_1^-|}{2} \right\rfloor$, then we let $M_L^- \subseteq W_L^-$ be of size $\left\lfloor \frac{|\tilde{M}_1^-|}{2} \right\rfloor$ and assert its maximality among the query edges $Q_2^- \cap (A_{\text{in}}^- \times B_{\text{out}}^-)$ by letting $N_L^- = A_{\text{in}}^- \backslash A(M_L^-) \times B_{\text{out}}^- \backslash B(M_L^-)$. Otherwise, if $|W_L^-| < \left\lfloor \frac{|\tilde{M}_1^-|}{2} \right\rfloor$, then we let $M_L^- = W_L^-$ which is trivially maximal; thus, $N_L^- = \emptyset$. We similarly consider W_R^- to construct the maximal matching M_R^- with non-edges N_R^-. Thus, there are no length-3 paths in $\tilde{M}_1^- \cup M_L^- \cup M_R^-$.

Finally, we let σ be a graph isomorphism that relates this to $\tilde{H}_2^-$ where $M_L^- \subseteq \sigma(\tilde{M}_L^-)$ and $M_R^- \subseteq \sigma(\tilde{M}_R^-)$; thus, we have that $M_2^- = M_L^- \cup M_R^- \subseteq \sigma(\tilde{M}_2^-)$ and $N_2^- = N_L^- \cup N_R^- \subseteq \sigma(\tilde{N}_2^-)$. ◀

▶ **Lemma 28.** *$\tilde{H}_2$ is partitioned w.r.t. parts (A°, B°) and $(A^{\overline{\circ}}, B^{\overline{\circ}})$ into structure graphs $\tilde{H}_2^\circ = \tilde{H}_2(A^\circ, B^\circ)$ and $\tilde{H}_2^{\overline{\circ}} = \tilde{H}_2(A^{\overline{\circ}}, B^{\overline{\circ}})$, respectively, where $\tilde{H}_2^\circ \dot{\cup} \tilde{H}_2^{\overline{\circ}}$ dominates $\tilde{H}_2$.*

Proof. Consider part (A°, B°) and its vertex-induced subgraph. Since (A°, B°) consists of the vertices of length-6 cycles each with a corresponding vertex unmatched by both matchings $\tilde{M}_1$ and $\tilde{M}_2$, we have that, for every two edges of $\tilde{M}_1$ included, two edges of $\tilde{M}_2$, one incident to A_{in} and one to B_{in}, and two unmatched vertices, one in A_{out} and one in B_{out}, are added to (A°, B°). Therefore, it is a generalisation of $\tilde{H}_2$. Part $(A^{\overline{\circ}}, B^{\overline{\circ}})$ follows similarly since the remaining vertices must have the same properties. ◀

▶ **Lemma 29.** *Any structure graph H_3° learned by the player is dominated by $\tilde{H}_3^\circ$.*

Proof. Let $Q_3^\circ = Q_3 \cap (A^\circ \times B^\circ)$ be any query w.r.t. part (A°, B°). By construction, every path in P° forms a 6-cycle with another one using the edges of $Q_\circ^{\text{ext}} = Q_3^\circ \cap K_\circ^{\text{ext}}$. Let C be a set representing the $\frac{|\tilde{M}_1^\circ|}{2}$ many 6-cycles. Each 6-cycle in C has one vertex in $\overline{A_{\text{in}}^\circ(P^\circ)}$ and one in $\overline{B_{\text{in}}^\circ(P^\circ)}$ such that every edge of $Q_\circ^{\text{rep}} = Q_3^\circ \cap K_\circ^{\text{rep}}$ is incident to exactly one of these vertices while the other is in $\overline{A_{\text{out}}^\circ(P^\circ)} \cup \overline{B_{\text{out}}^\circ(P^\circ)}$. As such, we represent each 6-cycle as an edge between its endpoints in $\overline{A_{\text{in}}^\circ(P^\circ)}$ and $\overline{B_{\text{in}}^\circ(P^\circ)}$ and delete all its other vertices. This exactly simulates a structure graph that, using a graph isomorphism σ', is dominated by the generalisation $\tilde{H}_1(A', B')$ where $A' = \overline{A_{\text{in}}^\circ(P^\circ)} \cup \overline{A_{\text{out}}^\circ(P^\circ)}$, $B' = \overline{B_{\text{in}}^\circ(P^\circ)} \cup \overline{B_{\text{out}}^\circ(P^\circ)}$ and C has a one-to-one correspondence with $\tilde{M}_1'$. There is then a one-to-one correspondence of the edges $K_\circ^{\text{rep}}$ to the edges $(A'_{\text{in}} \times B'_{\text{out}}) \cup (A'_{\text{out}} \times B'_{\text{in}})$; hence, any query Q_3^{rep} w.r.t. $\tilde{H}_2^\circ$ has a one-to-one correspondence to a query Q' w.r.t. $\tilde{H}_1(A', B')$.

[4] Note that the player knows $\tilde{H}_1$ at the start of round 2 and $\tilde{H}_2$ is specified w.r.t. $\tilde{H}_1$; thus, the graph isomorphism from round 1 is implicitly considered in H_2 and $\tilde{H}_2$.

Using the round 2 analysis in a white-box manner, we partition the vertices A' and B' into parts (A^+, B^+) and (A^-, B^-) using the query Q', which then, using isomorphisms σ^+ and σ^-, are dominated by $\tilde{H}_2^+$ and $\tilde{H}_2^-$ by Lemmas 25 and 26, respectively. Let C^+ and C^- be the corresponding partition of C w.r.t. the partitioning of $\tilde{M}_1'$ into $\tilde{M}_1^+$ and $\tilde{M}_1^-$. Let $\tilde{M}_\circ^+, \tilde{M}_\circ^-$ be the matchings and $\tilde{N}_\circ^+, \tilde{N}_\circ^-$ be the non-edges in $\tilde{H}_3^\circ$ that correspond to the matchings $\tilde{M}_2^+, \tilde{M}_2^-$ and the non-edges $\tilde{N}_2^+, \tilde{N}_2^-$ in $\tilde{H}_2^+, \tilde{H}_2^-$, respectively. Recall that $\tilde{M}_2^+$ (and $\tilde{M}_2^-$) has an equal number of edges incident to A'_{in} and B'_{in} such that each edge in $\tilde{M}_1^+$ (resp. $\tilde{M}_1^-$), except for at most one, is incident to exactly one edge in $\tilde{M}_2^+$ (resp. $\tilde{M}_2^-$); hence, $\tilde{M}_\circ^+$ (resp. $\tilde{M}_\circ^-$) has an equal number of edges in K_L^{rep} and K_R^{rep} such that each 6-cycle in C^+ (resp. C^-), except for at most one, is incident to exactly one edge in $\tilde{M}_\circ^+$ (resp. $\tilde{M}_\circ^-$).

We now construct each vertex-disjoint gadget of $\mathcal{G}^\circ$ to consist of the endpoints of two distinct edges in $\tilde{M}_\circ^+$ (or $\tilde{M}_\circ^-$), one from K_L^{rep} and one from K_R^{rep}, including the vertices of the two respective cycles in C^+ (resp. C^-) that they are incident to, and two distinct unmatched vertices corresponding to vertices in (A^+, B^+) (resp. (A^-, B^-)), one from $\overline{A_{\text{out}}^\circ(P^\circ)}$ and one from $\overline{B_{\text{out}}^\circ(P^\circ)}$. Thus, the non-edges $\tilde{N}_\circ^+$ (resp. $\tilde{N}_\circ^-$) are such that each gadget only has non-edges where both vertices are unmatched by $\tilde{M}_\circ^+$ (resp. $\tilde{M}_\circ^-$). Then, $\tilde{M}_\circ^+ \cup \tilde{M}_\circ^- = \tilde{M}_\circ^{\text{rep}}$ and $\tilde{N}_\circ^+ \cup \tilde{N}_\circ^- = \tilde{N}_\circ^{\text{rep}}$

We currently have that each gadget consists of two 6-cycles, each using two vertex-disjoint edges of $Q_\circ^{\text{ext}}$. If the number of 6-cycles $|C^+|$ (or $|C^-|$) in part (A^+, B^+) (resp. (A^-, B^-)) is odd, then there is one 6-cycle that would not be in any gadget, i.e., the one without an incident edge in $\tilde{M}_\circ^+$ (resp. $\tilde{M}_\circ^-$); hence, in any case, there are at least $\frac{|C^+|+|C^-|-2}{2} = \frac{|\tilde{M}_1^\circ|}{4} - 1$ many gadgets in $\mathcal{G}^\circ$. We complete each gadget by letting the edges of $Q_\circ^{\text{ext}}$ for only one of the 6-cycles be the edges of the matching $\tilde{M}_\circ^{\text{ext}}$ while the edges for the other are committed as the non-edges $\tilde{N}_\circ^{\text{ext}}$, which are vertex-disjoint from $\tilde{M}_\circ^{\text{ext}}$ since the 6-cycles are vertex-disjoint. Then, with the non-edges $\tilde{N}_\circ^{\text{gad}}$, we have that $\tilde{M}_\circ^{\text{ext}} \cup \tilde{M}_\circ^{\text{rep}} = \tilde{M}_3^\circ$ is a maximal matching since $\tilde{N}_\circ^{\text{rep}} \cup \tilde{N}_\circ^{\text{ext}} \cup \tilde{N}_\circ^{\text{gad}} = \tilde{N}_\circ^{\max} \cup \tilde{N}_\circ^{\text{gad}} = \tilde{N}_3^\circ$. Therefore, any structure graph H_3° learned w.r.t. the player's query Q_3° is dominated by $\tilde{H}_3^\circ$ such that $M_3^\circ \subseteq \sigma(\tilde{M}_3^\circ)$ and $N_3^\circ \subseteq \sigma(\tilde{N}_3^\circ)$ using the graph isomorphism $\sigma = \sigma' \circ \sigma^+ \circ \sigma^-$. ◀

▶ **Lemma 30.** *Any structure graph $H_3^{\overline{\circ}}$ learned by the player is dominated by $\tilde{H}_3^{\overline{\circ}}$.*

Proof. Before making a query, the player knows $\tilde{H}_2^{\overline{\circ}}$ in part $(A^{\overline{\circ}}, B^{\overline{\circ}})$; thus, we have that there are an equal number of paths in $P^{\overline{\circ}}$ that have their endpoints in $A^{\overline{\circ}}$ and in $B^{\overline{\circ}}$, which are also the same as the number of vertices in $\overline{A_{\text{out}}^{\overline{\circ}}(P^{\overline{\circ}})}$ and $\overline{B_{\text{out}}^{\overline{\circ}}(P^{\overline{\circ}})}$, respectively. As such, we first construct each vertex-disjoint gadget of $\mathcal{G}^{\overline{\circ}}$ to consist of the endpoints of four distinct paths in $P^{\overline{\circ}}$, two with their endpoints in $A^{\overline{\circ}}$ and two in $B^{\overline{\circ}}$, and four distinct vertices, two from $\overline{A_{\text{out}}^{\overline{\circ}}(P^{\overline{\circ}})}$ and two from $\overline{B_{\text{out}}^{\overline{\circ}}(P^{\overline{\circ}})}$. Since $|P^{\overline{\circ}}| = |\tilde{M}_1^{\overline{\circ}}|$ is even, there are $\left\lfloor \frac{|\tilde{M}_1^{\overline{\circ}}|}{4} \right\rfloor \geq \frac{|\tilde{M}_1^{\overline{\circ}}|}{4} - \frac{1}{2}$ many gadgets in $\mathcal{G}^{\overline{\circ}}$.

Let $Q_3^{\overline{\circ}} = Q_3 \cap (A^{\overline{\circ}} \times B^{\overline{\circ}})$ be any query w.r.t. part $(A^{\overline{\circ}}, B^{\overline{\circ}})$. At this point, each gadget of $\mathcal{G}^{\overline{\circ}}$ has the same structure and receives a subset of the query edges $Q_3^{\overline{\circ}}$ that is independent from the other gadgets. As such, it is sufficient to consider all possible queries $Q'_{\overline{\circ}} \subseteq K_{\overline{\circ}}^{\text{ext}} \cup K_{\overline{\circ}}^{\text{rep}}$ w.r.t. a single gadget. In general, $Q'_{\overline{\circ}}$ can be partitioned into four smaller parts: $Q_L^{\text{ext}} = Q'_{\overline{\circ}} \cap K_L^{\text{ext}}$, $Q_R^{\text{ext}} = Q'_{\overline{\circ}} \cap K_R^{\text{ext}}$, $Q_L^{\text{rep}} = Q'_{\overline{\circ}} \cap K_L^{\text{rep}}$ and $Q_R^{\text{rep}} = Q'_{\overline{\circ}} \cap K_R^{\text{rep}}$, each being any possible query w.r.t. two A-vertices and two B-vertices.

▷ Claim 34. Let Q be any arbitrary set of edges and let g be any arbitrary edge w.r.t. two A-vertices and two B-vertices where the player knows only the empty structure graph. Then, the structure graph with one edge $e \neq g$ and one non-edge f such that e and f are vertex-disjoint dominates the structure graph learned by the player after query Q.

Proof. If Q is the empty query, then no information is learned by the player and the claim holds. Otherwise, let $e \neq g \in Q$ be an arbitrary edge in the query, if one exists. Since there are only two A-vertices and two B-vertices, there exists only one possible edge f which is vertex-disjoint from e. If $f \in Q$ then we commit f as a non-edge, which makes the edge e a maximal matching. Otherwise, $f \notin Q$ and we have that e is already maximal. In the case where $\nexists e \neq g \in Q$, we commit $f = g$ as a non-edge, which implies that the empty matching is maximal since Q has no other edges. ◁

By Claim 34, we have that the player learns at most the edges $e_L^{\text{rep}} \in K_L^{\text{rep}}$ and $e_R^{\text{rep}} \in K_R^{\text{rep}}$, possibly with the non-edges $f_L^{\text{rep}} \in K_L^{\text{rep}}$ and $f_R^{\text{rep}} \in K_R^{\text{rep}}$, after the queries Q_L^{rep} and Q_R^{rep}, respectively. Therefore, the player learns the matching $M_{\overline{\circ}}^{\text{rep}} \subseteq \tilde{M}_{\overline{\circ}}^{\text{rep}} = \{e_L^{\text{rep}}, e_R^{\text{rep}}\}$ and non-edges $N_{\overline{\circ}}^{\text{rep}} \subseteq \tilde{N}_{\overline{\circ}}^{\text{rep}} = \{f_L^{\text{rep}}, f_R^{\text{rep}}\}$. As such, in each gadget, edge e_L^{rep} is incident to a path p with its endpoints in $B^{\overline{\circ}}$ while edge e_R^{rep} is incident to a path q with its endpoints in $A^{\overline{\circ}}$. Furthermore, we have that there exists a unique edge $g_L^{\text{ext}} \in K_L^{\text{ext}}$ and a unique edge $g_R^{\text{ext}} \in K_R^{\text{ext}}$ each of which are incident to p and q – we never want these edges to belong to the player's maximal matching $M_{\overline{\circ}}^{\text{ext}}$.

We now consider the pair of queries Q_L^{ext} and Q_R^{ext} that, by construction of part $(A^{\overline{\circ}}, B^{\overline{\circ}})$, do not contain edges that form 6-cycles with any paths in $P^{\overline{\circ}}$. This implies that if $g_L^{\text{ext}} \in Q_L^{\text{ext}}$ (or $g_R^{\text{ext}} \in Q_R^{\text{ext}}$) then $g_R^{\text{ext}} \notin Q_R^{\text{ext}}$ (resp. $g_L^{\text{ext}} \notin Q_L^{\text{ext}}$). Furthermore, since every pair of Q_L^{ext} and Q_R^{ext} has a symmetrical pair, we only need to consider the pairs of queries where $g_L^{\text{ext}} \notin Q_L^{\text{ext}}$ and show that the edges and non-edges learned in a single gadget are subsets of the edges $\tilde{M}_{\overline{\circ}}^{\text{ext}}$ and (either case of) the non-edges $\tilde{N}_{\overline{\circ}}^{\text{ext}}$ w.r.t. a single gadget.

If either query is empty, by Claim 34, we have that the edges, which avoid g_R^{ext}, and non-edges learned are thus subsets of a first case gadget in $\tilde{H}_3^{\overline{\circ}}$. Otherwise, both queries are non-empty and there exists query edges $e_L \neq g_L^{\text{ext}} \in Q_L^{\text{ext}}$ and $e_R \in Q_R^{\text{ext}}$. We then have that they either form a length-8 path in the gadget or do not. Consider first the simpler latter case where the query edges necessarily form two length-5 paths and are the only query edges since they do not form length-8 paths or 6-cycles. Let $M_{\overline{\circ}}^{\text{ext}} = \{e_L^{\text{ext}}\}$ and $N_{\overline{\circ}}^{\text{ext}} = \{f_R^{\text{ext}}\}$ where $e_L^{\text{ext}} = e_L$ and $f_R^{\text{ext}} = e_R$. $M_{\overline{\circ}}^{\text{ext}}$ is naturally maximal, and the edges and non-edges learned are a subset of the second case gadget in $\tilde{H}_3^{\overline{\circ}}$.

Consider now the case where the query edges $e_L \neq g_L^{\text{ext}}$ and e_R form a length-8 path. If $e_R \neq g_R^{\text{ext}}$, let $M_{\overline{\circ}}^{\text{ext}} = \{e_L^{\text{ext}}, e_R^{\text{ext}}\}$ where $e_L^{\text{ext}} = e_L$ and $e_R^{\text{ext}} = e_R$. Since $e_L^{\text{ext}} \neq g_L^{\text{ext}}$ the endpoints of either p or q in the gadget remain unmatched by $M_{\overline{\circ}}^{\text{ext}}$. Then, if they exist, we add the edges $f_L^{\text{ext}} \in Q_L^{\text{ext}}$ and $f_R^{\text{ext}} \in Q_R^{\text{ext}}$ that are unmatched by $M_{\overline{\circ}}^{\text{ext}}$ to an initially empty set of non-edges $N_{\overline{\circ}}^{\text{ext}}$; hence, $M_{\overline{\circ}}^{\text{ext}}$ is maximal and the edges and non-edges learned are a subset of the first case gadget in $\tilde{H}_3^{\overline{\circ}}$. Otherwise, we have that $e_R = g_R^{\text{ext}}$ and let $N_{\overline{\circ}}^{\text{ext}} = \{f_L^{\text{ext}}, f_R^{\text{ext}}\}$ where $f_L^{\text{ext}} = e_L$ and $f_R^{\text{ext}} = e_R$. Any possible remaining query edges must be either only incident to or only not incident to $N_{\overline{\circ}}^{\text{ext}}$ since they may not form any 6-cycles. We pick at most one from Q_L^{ext} and one from Q_R^{ext} then add them to an initially empty matching $M_{\overline{\circ}}^{\text{ext}}$, which is thus maximal and, if both exist, forms a length-8 path that leaves the endpoints of either p or q in the gadget unmatched since $M_{\overline{\circ}}^{\text{ext}}$ does not contain g_L^{ext} or g_R^{ext}. If $M_{\overline{\circ}}^{\text{ext}}$ and $N_{\overline{\circ}}^{\text{ext}}$ are incident to each other, then the edges and non-edges learned are a subset of the second case gadget in $\tilde{H}_3^{\overline{\circ}}$. Otherwise, they are a subset of the first case gadget in $\tilde{H}_3^{\overline{\circ}}$. ◀

Distributed Quantum Interactive Proofs

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

Masayuki Miyamoto
Graduate School of Mathematics, Nagoya University, Japan

Harumichi Nishimura
Graduate School of Informatics, Nagoya University, Japan

Abstract

The study of distributed interactive proofs was initiated by Kol, Oshman, and Saxena [PODC 2018] as a generalization of distributed decision mechanisms (proof-labeling schemes, etc.), and has received a lot of attention in recent years. In distributed interactive proofs, the nodes of an n-node network G can exchange short messages (called certificates) with a powerful prover. The goal is to decide if the input (including G itself) belongs to some language, with as few turns of interaction and as few bits exchanged between nodes and the prover as possible. There are several results showing that the size of certificates can be reduced drastically with a constant number of interactions compared to non-interactive distributed proofs.

In this paper, we introduce the quantum counterpart of distributed interactive proofs: certificates can now be quantum bits, and the nodes of the network can perform quantum computation. The first result of this paper shows that by using distributed quantum interactive proofs, the number of interactions can be significantly reduced. More precisely, our result shows that for any constant k, the class of languages that can be decided by a k-turn classical (i.e., non-quantum) distributed interactive protocol with $f(n)$-bit certificate size is contained in the class of languages that can be decided by a 5-turn distributed quantum interactive protocol with $O(f(n))$-bit certificate size. We also show that if we allow to use shared randomness, the number of turns can be reduced to three. Since no similar turn-reduction *classical* technique is currently known, our result gives evidence of the power of quantum computation in the setting of distributed interactive proofs as well.

As a corollary of our results, we show that there exist 5-turn/3-turn distributed quantum interactive protocols with small certificate size for problems that have been considered in prior works on distributed interactive proofs such as [Kol, Oshman, and Saxena PODC 2018, Naor, Parter, and Yogev SODA 2020].

We then utilize the framework of the distributed quantum interactive proofs to test closeness of two quantum states each of which is distributed over the entire network.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Quantum computation theory

Keywords and phrases distributed interactive proofs, distributed verification, quantum computation

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.42

Related Version *Full Version*: https://arxiv.org/abs/2210.01390 [9]

Funding FLG was supported by the JSPS KAKENHI grants JP16H01705, JP19H04066, JP20H00579, JP20H04139, JP20H05966, JP21H04879 and by the MEXT Q-LEAP grants JPMXS0118067394 and JPMXS0120319794. MM would like to take this opportunity to thank the "Nagoya University Interdisciplinary Frontier Fellowship" supported by Nagoya University and JST, the establishment of university fellowships towards the creation of science technology innovation, Grant Number JPMJFS212. HN was supported by the JSPS KAKENHI grants JP19H04066, JP20H05966, JP21H04879, JP22H00522 and by the MEXT Q-LEAP grants JPMXS0120319794.

© François Le Gall, Masayuki Miyamoto, and Harumichi Nishimura;
licensed under Creative Commons License CC-BY 4.0

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 42; pp. 42:1–42:21

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

1 Introduction

1.1 Distributed Interactive Proofs

In distributed computing, efficient verification of graph properties of the network is useful from both theoretical and applied aspects. The study of this notion of verification in the distributed setting has lead to the notion of "distributed NP" in analogy with the complexity class NP in centralized computation: A powerful prover provides certificates to each node of the network in order to convince that the network has a desired property; If the property is satisfied, all nodes must output "accept", otherwise at least one node must output "reject". This concept of "distributed NP" has been formulated in several ways, including *proof-labeling schemes* (PLS) [19], *non-deterministic local decision* (NLD) [5], and *locally checkable proofs* (LCP) [10].

As a motivating example, consider the problem of verifying whether the network is bipartite or not. While this problem cannot be solved in $O(1)$ rounds without prover [29], it can easily be solved as following: The prover tells each node which part it belongs to, which requires only a 1-bit certificate per node, and then each node broadcasting this information to its adjacent nodes (here the crucial point is that if the network is non-bipartite, then at least one node will be able to detect it). On the other hand, it is known that there exist properties that require large certificate size to decide: Göös and Suomela [10] have shown that recognizing symmetric graphs (Sym) and non 3-colorable graphs ($\overline{\text{3Col}}$) require $\Omega(n^2)$-bit certificates per node in the framework of LCP (which is tight since all graph properties are locally decidable by giving the $O(n^2)$-bit adjacency matrix of the graph).

To reduce the length of the certificate for such problems, the notion of distributed interactive proofs (also called distributed Arthur-Merlin proofs) was recently introduced by Kol, Oshman and Saxena [18] as a generalization of distributed NP. In this model there are two players, the prover (often called Merlin), who has unlimited computational power and sees the entire network but is untrusted (i.e., can be malicious), and the verifier (often called Arthur) representing all the nodes of the network, who can perform only local computation and brief communication with adjacent nodes. Generalizing the concept of distributed NP, the nodes are now allowed to engage in multiple turns of interaction with the prover. As for distributed NP, there are two requirements of the protocol: if the input is legal (yes-instance) then all nodes must accept with high probability (*completeness*), and if the input is illegal then at least one node must reject with high probability (*soundness*).

In the setting of [18], each node has access to a private source of randomness, and sends generated random bits to the prover in Arthur's turn. For instance, a 2-turn protocol contains two interactions: Arthur first queries Merlin by sending a random string from each node, and then Merlin provides a certificate to each node. After that, nodes exchange messages with adjacent nodes to decide their outputs. The main complexity measures when studying distributed interactive protocols are the size of certificates provided to each node, the size of the random strings generated at each node and the size of the messages exchanged between nodes. Let us denote $\mathsf{dAM}[k](f(n))$ the class of languages that have k-turn distributed Arthur-Merlin protocols where Merlin provides $O(f(n))$-bit certificates, Arthur generates $O(f(n))$-bit random strings at each node and $O(f(n))$-bit messages are exchanged between nodes. Kol et al. [18] showed the power of interaction by giving a $\mathsf{dMAM}(\log n) = \mathsf{dAM}[3](\log n)$ protocol for graph symmetry (Sym) and a $\mathsf{dAMAM}(n \log n) = \mathsf{dAM}[4](n \log n)$ protocol for graph non-isomorphism (GNI), which are known to require $\Omega(n^2)$-bit certificate in LCP (see Appendix A for the definition of these problems).

This model has been further studied in several works. Naor, Parter and Yogev [25] showed that any $O(n)$-time centralized computation can be converted into a $\mathsf{dMAM}(\log n) = \mathsf{dAM}[3](\log n)$ protocol. Using this compiler, for instance, they constructed a $\mathsf{dMAMAM}(\log\log n) = \mathsf{dAM}[5](\log\log n)$ protocol for SetEquality and a special case of Sym. Crescenzi, Fraigniaud and Paz [3] initiated the study of distributed Arthur-Merlin protocols with shared randomness: in each Arthur's turn, Arthur generates a random string that can be seen from all nodes. In order to distinguish the two models we use dAM for the (standard) private randomness setting and dAM^{sh} for the shared randomness setting. They showed that dAM protocols can simulate dAM^{sh} protocols by giving additional $O(\log n)$-size certificates. The role of shared randomness was further investigated by Montealegre, Ramírez-Romero and Rapaport [23], who showed the computational power of small-certificate dAM^{sh} protocols without private randomness is relatively weak: for any constant k, $\mathsf{dAM}^{sh}[k]$ protocols with message size m can be converted to locally checkable proofs (LCPs) with message size $O(2^m + \log n)$.

Lower bounds on distributed Arthur-Merlin protocols for some concrete problems are known. Kol, Oshman and Saxena [18] showed that if the language Sym is in the class $\mathsf{dAM}[2](f(n))$, then $f(n) \in \Omega(\log\log n)$. As mentioned in [6], this lower bound can actually be improved to $f(n) \in \Omega(\log n)$. On the other hand, there is no known method to prove lower bounds when the number of turns is three or more.

1.2 Quantum Interactive Proofs

Quantum interactive proofs (QIP) were introduced by Watrous [31] in the centralized setting as a variant of classical interactive proofs (IP) in which the verifier can perform polynomial-time quantum computation (instead of polynomial-time classical computation), and the prover and verifier can exchange quantum bits (instead of classical bits). Kitaev and Watrous [17] first showed that QIP, the class of languages that can be decided by a quantum interactive protocol with polynomial number of interactions, is contained in EXP, the class of languages decided in exponential time. This containment was improved by Jain, Ji, Upadhyay, and Watrous [14], who showed that QIP is actually contained in PSPACE, which implies that QIP collapses to the complexity class IP ($\mathsf{QIP} = \mathsf{IP} = \mathsf{PSPACE}$).

While the above result shows that quantum interactive proofs are not more powerful than classical interactive proofs, there is a striking property of quantum interactive proofs that is not expected to hold for classical interactive proofs: in the quantum case the number of interactions can be significantly reduced. More precisely, Watrous first showed that any language in PSPACE can be decided by a three-turn QIP protocol [31]. After that, Kitaev and Watrous [17] showed that any QIP protocol with a polynomial number of interaction can be parallelized to three turns ($\mathsf{QIP} = \mathsf{QIP}[3]$). Marriott and Watrous [22] additionally showed that the verifier's turn in $\mathsf{QIP}[3]$ protocols can be replaced by a 1-bit coin flip ($\mathsf{QIP}[3]$=QMAM). Kempe, Kobayashi, Matsumoto, and Vidick [16] showed an alternative proof of $\mathsf{QIP} = \mathsf{QIP}[3]$.

1.3 Our Results

In this paper we introduce the quantum counterpart of distributed interactive proofs, which we call distributed quantum interactive proofs (or sometimes distributed quantum interactive protocols) and write dQIP, and show their power. Roughly speaking, distributed quantum interactive proofs are defined similarly to the classical distributed interactive proofs (i.e., distributed Arthur-Merlin proofs) defined above, but the messages exchanged between the prover and the nodes of the network can now contain quantum bits (qubits), the nodes can

now do any (local) quantum computation (i.e., each node can apply any unitary transform to the registers it holds), and each node can now send messages consisting of qubits to its adjacent nodes. In analogy to the classical case, the main complexity measures when studying distributed quantum interactive protocols are the size of registers exchanged between the prover and the nodes, and the size of messages exchanged between the nodes. We give the formal definition of dQIP in Section 2. The class $\mathsf{dQIP}[k](f(n))$ is defined as the set of all languages that can be decided by a k-turn dQIP protocol where both the size of the messages exchanged between the prover and the nodes, and the size of the messages exchanged between the nodes are $O(f(n))$ qubits.

Our first result is the following theorem.

▶ **Theorem 1.** *For any constant $k \geq 5$, $\mathsf{dAM}[k](f(n)) \subseteq \mathsf{dQIP}[5](f(n))$.*

Theorem 1 shows that by using distributed quantum interactive proofs, the number of interactions in distributed interactive proofs can be significantly reduced. To prove this result, we develop a generic *quantum* technique for turn reduction in distributed interactive proofs. Since no similar turn-reduction *classical* technique is currently known, our result gives evidence of the power of quantum computation in the setting of distributed interactive proofs as well.

We also show that if we allow to use randomness shared to all nodes (we denote this model by dQIP^{sh}), the number of turns can be further reduced to three turns.

▶ **Theorem 2.** *For any constant $k \geq 3$, $\mathsf{dAM}[k](f(n)) \subseteq \mathsf{dQIP}^{sh}[3](f(n))$.*

On the other hand, in the classical case, it is known that allowing shared randomness does not change the class [3]: $\mathsf{dAM}^{sh}[k](f(n)) \subseteq \mathsf{dAM}[k](f(n))$ for all $k \geq 3$.[1]

As mentioned above, for (classical) dAM protocols increasing the number of turns is helpful to reduce the complexity (in particular, the certificate size) for many problems. Our result thus shows if we allow quantum resource, such protocols can be simulated in five turns, and in three turns if we allow shared randomness. More precisely, we obtain the following corollary (see Appendix A for the precise definitions of these problems and Theorems 12 and 13 in Section 4 for a statement of the corresponding classical results):

▶ **Corollary 3.**

1. *There exist*
   - *a $\mathsf{dQIP}^{sh}[3](\log n)$ protocol for Asym,*
   - *a $\mathsf{dQIP}^{sh}[3](\log n)$ protocol for GNI,*
   - *a $\mathsf{dQIP}^{sh}[3](\log \log n)$ protocol for SetEquality,*
   - *a $\mathsf{dQIP}^{sh}[3](\log \log n)$ protocol for DSym,*
   - *a $\mathsf{dQIP}[5](\log n)$ protocol for GNI.*
2. *There exists a constant δ such that if a language $\mathcal{L}$ can be decided in $\mathrm{poly}(n)$ time and n^{δ} space, then $\mathcal{L} \in \mathsf{dQIP}[5](\log n)$ and $\mathcal{L} \in \mathsf{dQIP}^{sh}[3](\log n)$.*

We also introduce a *quantum* problem (i.e., the inputs are quantum states) which arises naturally when considering distributed quantum networks. More specifically, we consider the following task: There are two quantum states $|\psi\rangle$ and $|\phi\rangle$ as the inputs, each of which is distributed over the entire network (each node $u \in V$ has N_u-qubit of $|\psi\rangle$ and $|\phi\rangle$, where

[1] In fact, the authors of [3] showed $\mathsf{dAM}^{sh}[k](f(n)) \subseteq \mathsf{dAM}[k](f(n) + \log n)$ for all $k \geq 1$ where the additional $\log n$ comes from constructing a spanning tree, but for $k \geq 3$, a spanning tree can be constructed with $O(1)$-sized messages between the prover and the nodes in three turns [25], so $\log n$ can be removed.

$\sum_{u \in V} N_u = N$). The goal of the task is to measure closeness of these states. We call this problem N-qubit Distributed Quantum Closeness Testing (DQCT_N). For this task, we show the following theorem (see Appendix B for the definition of the trace distance).

▶ **Theorem 4.** *There is a* $\mathsf{dQIP}[5](O(1))$ *protocol for* DQCT_N*, where the completeness and the soundness conditions are defined as follows:*

- ***Completeness:*** *If* $|\psi\rangle = |\phi\rangle$ *and the prover is honest, the protocol accepts with probability 1.*
- ***Soundness:*** *If the protocol accepts with probability* $1 - 1/z$*,* $\mathrm{dist}(|\psi\rangle, |\phi\rangle) \leq \sqrt{2/z} + \varepsilon$ *for any small constant* $\varepsilon > 0$*. Here* $\mathrm{dist}(,)$ *is the trace distance.*

Without the prover, a naive approach is to accumulate all of the input to the leader node, and perform local operations at the leader node to measure their closeness. Obviously this approach is inefficient in the following sense: (1) it requires $\Omega(D)$-round of communication where D is the diameter of the network; (2) the amount of communication is linear in N, the size of the input quantum states. Theorem 4 means that in the dQIP setting, (1) it only needs 1-round of communication between the nodes; (2) the amount of communication (the size of messages per edge, and the size of messages between each node and the prover) is $O(1)$, regardless of the input size. Note that the main result of the recent paper [4] (see Section 1.5 for their result) immediately shows that if the two input quantum states are hold by some specific two nodes respectively and there is no input for the other nodes, DQCT_N in an n-node network can be done with $O(N \cdot \mathrm{poly}(n))$ size of quantum proofs and 1-round of communication between nodes, in the non-interactive setting. Our setting is more general in the sense that the input quantum states can be distributed over the entire network.

Lastly, in Appendix C, we show how to transform dQIP protocols with two-sided (i.e., completeness and soundness) bounded error into dQIP protocols with perfect completeness. We show that if we allow the communication between nodes in the middle of interaction (we call this model as dQIP^{com}), achieving perfect completeness is possible. Thus dQIP^{com} protocols can be converted to 5-turn protocols with perfect completeness using parallel repetition with a fairly small increase of the message size.

1.4 Organization and Overview of our Approach

We start by considering a more powerful model than dQIP, which allows nodes to use shared randomness. That is, at each turn the network can send a shared random string of limited length to the prover. We call this model dQIP^{sh}. In Section 3.1, we first show how to reduce the number of turns by half in the dQIP^{sh} model. This is shown by adapting to the distributed setting the method of Kempe et al. [16], which reduces the number of turns of QIP by half. More precisely, we show that for any $\ell \geq 1$, $(4\ell + 1)$-turn dQIP^{sh} protocols can be parallelized to $(2\ell + 1)$-turn.

The main idea of [16] is the following. In the first turn the honest prover provides the verifier with a snapshot state of the $(2\ell + 1)$-th turn, which includes the state of its private register and the message register in the original protocol. In the second turn the verifier flips a fair coin and sends it to the prover. In the remaining turns they perform the forward or backward simulation of the original protocol, according to the result of the coin flip.

We then go back to the dQIP model in Section 3.2 and show how to reduce the number of turns by half in the dQIP model by using the same argument as in the case of dQIP^{sh}, by using two additional turns in order to share the result of the coin flip. We can thus parallelize $(4\ell + 1)$-turn dQIP protocols to $(2\ell + 3)$-turn. Applying recursively this approach makes possible to reduce the number of turns down to 7 (corresponding to $\ell = 2$), but not lower.

After that, we focus on how to parallelize 7-turn dQIP protocols to 5-turn. Starting from 7-turn, we can reduce the number of turns to 5 in the dQIP^{sh} model, as in the QIP model. In dQIP model we need additional two turns, so we need a different approach to turn-reduction when we start 7-turn protocols. To parallelize 7-turn to 5-turn, we use a protocol similar to the Marriott-Watrous protocol [22]. Their protocol is used to show that $\mathsf{QIP}[3] \subseteq \mathsf{QMAM}$, where QMAM is the subclass of QIP[3] in which messages Arthur can send to Merlin are random bits. We construct a similar protocol, which can be used to parallelize 7-turn dQIP protocols to 5-turn dQIP protocols.

In Section 4 we then prove Theorem 1 and Theorem 2. We first discuss how to convert $\mathsf{dAM}[k](f(n))$ protocols to $\mathsf{dQIP}[k](f(n))$ protocols. This is achieved by doing all the computation of the dAM protocol in a reversible manner (i.e., unitary computation). Since the verification phase remains classical, the probability of being fooled is as low as the original dAM protocol, no matter what entangled state the malicious prover sends. This converted $\mathsf{dQIP}[k](f(n))$ protocol is then parallelized to 5-turn (3-turn in dQIP^{sh} model, respectively) by repeatedly using the technique we show in Section 3. We also have to discuss the size of messages. Since in $\mathsf{dAM}[k](f(n))$ protocols, the private registers of the nodes are used only to store a copy of certificates, the size of the private registers of nodes in the converted $\mathsf{dQIP}[k]$ protocol is $O(f(n))$. Therefore the size of the snapshot states given by the prover is also $O(f(n))$.

In Section 5, we tackle with the task to test closeness of two distributed quantum states (DQCT_N in Section 1.3). In the full version [9], we present a dQIP protocol for this task. The main difficulty is the implementation of the controlled SWAP gate, since there is no prior shared entanglement in the network. To resolve this issue, we utilize the protocol of Zhu and Hayashi [34] to make the nodes share the GHZ state $\frac{1}{\sqrt{2}}(|0^n\rangle + |1^n\rangle)$. Using the prover, we show the tests (described by some POVM measurement) in the protocol of [34] can be implemented in the distributed setting. Another difficulty is to avoid to be fooled by the malicious prover. This is achieved by carefully constructing the protocol, which ensures that the malicious prover cannot fraudulently increase the acceptance probability.

1.5 Related Works

Although there is no previous result about distributed interactive proofs with quantum resources, Fraigniaud, Le Gall, Nishimura and Paz [4] investigated the quantum version of *randomized proof-labeling schemes* (or equivalently, dMA protocols). They considered the following problem: N-bit inputs (where N is sufficiently larger than the number of nodes n) are given to several nodes in a network and the goal is to check if all inputs are equal. They gave a dQMA protocol with $O(\log N)$ certificate size, and showed that any classical dMA protocol requires $\Omega(N)$ certificate size, which shows the superiority of quantum certification in this setting. Another example is a very recent work by Le Gall, Miyamoto, and Nishimura [8], which studied *quantum* problems in the dQMA framework.

The study of distributed interactive proofs for some concrete problems has been developed recently, including two or three turn distributed Merlin-Arthur protocols for recognition of cographs, distance-hereditary graphs, and some geometric intersection graph classes [24, 15].

Research on quantum distributed algorithms that outperform classical distributed algorithms in several standard distributed models has been very active recently: there have been investigations showing the superiority of quantum distributed algorithm in the CONGEST model [20, 12, 2], the CONGEST CLIQUE model [13] and the LOCAL model [21].

2 Definitions

2.1 Distributed Interactive Proofs

In this section we describe classical distributed interactive proofs, following the definition by Kol, Oshman and Saxena [18].

In distributed interactive proofs the verifier consists of a network represented by connected graph $G = (V, E)$ with $|V| = n$ nodes, and each node $u \in V$ is given its input label $I(u)$ where $I : V \to \{0,1\}^*$ is a function. We let $\mathcal{G}$ be the set of all connected graphs on vertices V, and $\mathcal{I}$ be the set of functions that maps V to $\{0,1\}^*$. Define a language $\mathcal{L}$ by a subset

$$\mathcal{L} \subseteq \mathcal{G} \times \mathcal{I}.$$

Given a network configuration $(G, I) \in \mathcal{G} \times \mathcal{I}$ and a language $\mathcal{L}$, we consider an interactive protocol that consists of a series of interactions between a prover (*Merlin*) and a distributed verifier (*Arthur*). The goal of the protocol is to decide if $(G, I) \in \mathcal{L}$. The prover Merlin has unlimited computational power, and knows all information about (G, I). The verifier Arthur is distributed; it consists of the nodes of the network G, and initially each node u only knows its input $I(u)$. Merlin is not trusted and tries to convince Arthur that $(G, I) \in \mathcal{L}$ by sending bit strings (certificates). Arthur provides Merlin some random queries. There exist two types of randomness that can be used by Arthur: private randomness and shared randomness. In the private randomness setting, each node can generate random bits that cannot be seen by the other nodes. In the shared randomness setting, all random bits generated by Arthur are shared among all nodes. We denote the private randomness setting by dAM and the shared randomness setting by dAM^{sh}.

A k-turn distributed interactive protocol (also called k-turn distributed Arthur-Merlin protocol in [18]) has the interaction phase and the verification phase. The interaction phase begins with Merlin's turn if k is odd, and Arthur's turn otherwise. In the first turn, if k is odd, Merlin chooses a function $c_1 : \{0,1\}^* \to \{0,1\}^*$ determined by the network configuration (G, I), and sends $c_1(u)$ to each node u. In the second turn, Arthur picks a random string $r_2(u)$ at each node u, and sends them to Merlin. (In dAM^{sh} interactive protocols Arthur picks one random string and it can be seen by all nodes.) This series of interactions continues for k turns. More precisely, if the j-th turn is Merlin's turn, he sends a certificate $c_j(u)$ to each node u where $c_j : \{0,1\}^* \to \{0,1\}^*$ is a function of (G, I) and all of random strings $\{r_i(u)\}_{u \in V, i \in \{2,4,\ldots,j-1\}}$ received from Arthur, and if the j-th turn is Arthur's, he picks a random string $r_j(u)$ at each node u, and sends them to Merlin. If k is even then the interaction phase begins with Arthur's turn (i.e., Arthur first picks a random string $r_1(u)$ at each node u).

The protocol completes with the verification phase. In this phase every node u broadcasts a message M_u to its neighbors which may depend on its input $I(u)$, random strings u picked, and the certificates u received. Finally, u decides its output (accept or reject) by all information accumulated by u. Arthur accepts if and only if all nodes accept. We say that a protocol has completeness c and soundness s for a language $\mathcal{L}$ if the following conditions hold for the verifier G and the input label I:

1. (Completeness) If $(G, I) \in \mathcal{L}$, then there exists a prover P such that $\Pr(\text{all nodes accept}) \geq c$.
2. (Soundness) If $(G, I) \notin \mathcal{L}$, then for any prover P, $\Pr(\text{all nodes accept}) \leq s$.

As in [18], we define the class $\mathsf{dAM}[k](f(n))$ as the class of languages accepted by such k-turn distributed interactive protocols in which in each turn the prover and the verifier exchange $O(f(n))$ bits per node, and each node exchanges $O(f(n))$ bits with its neighbors during the verification procedure. The formal definition is as follows.

▶ **Definition 5** ([18]). *The class* $\mathsf{dAM}[k](f(n))$ *is the class of languages* $\mathcal{L} \subseteq \mathcal{G} \times \mathcal{I}$ *that have a* k*-turn distributed Arthur-Merlin protocol with completeness* $2/3$ *and soundness* $1/3$ *satisfying the following conditions:*

- *At each Merlin's turn, Merlin sends certificates of* $O(f(n))$ *bits per node, and at each Arthur's turn, each node sends* $O(f(n))$ *random bits to Merlin.*
- *The size of messages exchanged between two adjacent nodes in the verification phase is* $O(f(n))$ *bits.*

2.2 Distributed Quantum Interactive Proofs

In this section we define the quantum counterpart of distributed interactive proofs, which we call distributed quantum interactive proofs. We assume the reader is familiar with the basic notions of quantum computation such that bra-ket notation of qubits, quantum circuits, and density operators (see [27], for instance, for a good reference).

Distributed quantum interactive proofs are defined similarly to the classical distributed interactive proofs of Section 2.1, but now the messages exchanged between the prover and the nodes consist of qubits, the nodes can do any (local) quantum computation, and each node can send messages consisting of qubits to its adjacent nodes. To make this rigorous and define the complexity of the protocol, we need to carefully specify how the messages are encoded using quantum registers and who owns the registers during the computation.[2] Here is the formal definition.

▶ **Definition 6.** *A* k*-turn distributed quantum interactive proof (*dQIP*) is a protocol between a prover and a distributed verifier who interact in the following way:*

- ***The configuration:*** *The verifier consists of an* n*-node network* $G = (V, E)$*. Each node* u *begins with a quantum register* V_u*. We denote* V *the set of registers* $\{\mathsf{V}_u\}_{u \in V}$*. The prover begins with a quantum register* P*. In addition, there is a quantum message register* M_u *for each* u*. Let* M *be the set of registers* $\{\mathsf{M}_u\}_{u \in V}$*. The prover initially has the register* M *if* k *is odd, otherwise the node* u *initially has the register* M_u*. The initial state in* V *and* M *is the all-zero pure state* $|0 \cdots 0\rangle$*.*
- ***The interaction:*** *The interaction of a* dQIP *system is the repetition of prover's turn and verifier's turn. In the prover's turn, the prover performs arbitrary unitary transform denoted* P_i *to* (M, P) *and sends each* M_u *to* u *in the* i*-th turn. In the verifier's turn, the verifier can do any local (quantum) computation. More precisely, each node* u *performs an arbitrary unitary transform denoted* $V_{u,i}$ *to* $(\mathsf{V}_u, \mathsf{M}_u)$*. Then, each node* u *sends* M_u *to the prover in the* i*-th turn. We let* $V_i = \bigotimes_{u \in V} V_{u,i}$ *be the unitary transform applied by the verifier in the* i*-th turn.*
- ***The verification:*** *After the interaction phase, each node* u *prepares registers* $\mathsf{W}_{u,v}$ *for* $(u, v) \in E$ *which are initialized to* $|0 \cdots 0\rangle$ *and used for communication. Then* u *performs an arbitrary unitary transform on the registers* $\mathsf{V}_u, \mathsf{M}_u$ *and all* $\mathsf{W}_{u,v}$ *for* $(u, v) \in E$*. After that, each node communicates with its neighbors, performs a measurement, and then decides reject/accept based on the outcome of the measurement (a more formal description of this step is given to Section 2.2.1).*

[2] Since quantum information differs from classical information in several fundamental ways (in particular, quantum information cannot be copied and quantum message can share "entanglement"), when studying quantum communication complexity or quantum distributed computation, a quantum message is represented as a quantum register (i.e., a physical system comprising multiple qubits) and the action of sending a quantum message is represented by sending this quantum register. The message size corresponds to the size of the register.

Note that distributed quantum interactive proofs defined above can simulate random bits.[3] The size of the certificate sent from the prover to node u at each prover's turn is the size of the register M_u. The size of the message sent from node u to the prover at each verifier's turn is also the size of the register M_u. At the verification phase, the size of the message exchanged between node u and v is the size of the register $\mathsf{W}_{u,v}$. This leads to the following definition of the complexity class $\mathsf{dQIP}[k](f(n))$, as the natural quantum variant of the complexity class $\mathsf{dAM}[k](f(n))$ of Definition 5.

▶ **Definition 7.** *The class* $\mathsf{dQIP}[k](f(n))$ *is the class of languages* $\mathcal{L}$ *such that there exists a* k*-turn* dQIP *protocol for* $\mathcal{L}$ *with completeness* $\frac{2}{3}$ *and soundness* $\frac{1}{3}$ *satisfying the following conditions:*

- *The size of register* M_u *for each node* u *is* $O(f(n))$.
- *The size of register* $\mathsf{W}_{u,v}$ *exchanged between* u *and* v *in the verification phase is* $O(f(n))$ *for any* $(u,v) \in E$.
- *The conditions of completeness and soundness is the same as those of* dAM *protocols.*

2.2.1 Technical Details about the Verification Phase

We now give a more formal (and more technical) description of the verification phase of a k-turn dQIP protocol in Definition 6. The communication and measurement operations can be specifically described as follows: for any $(u,v) \in E$, the two registers $\mathsf{W}_{u,v}$ and $\mathsf{W}_{v,u}$ are swapped by the SWAP gate (see Appendix B for the definition of the SWAP gate). Here, we let V_{k+1} be the unitary transform that is performed in the verification phase. If k is odd then the interaction begins with the prover's turn, and the entire unitary transform is written by $Q = V_{k+1}P_k \cdots V_2 P_1$. If k is even then Q is written by $Q = V_{k+1}P_k \cdots P_2 V_1$. After that, each node u performs a POVM measurement $(\Pi_{\mathrm{acc},u}, \Pi_{\mathrm{rej},u} = I - \Pi_{\mathrm{acc},u})$ on register $\mathsf{V}_u, \mathsf{M}_u$ and $\mathsf{W}_{u,v}$ for $(u,v) \in E$ to obtain its output (see Appendix B for the definition of POVMs). Without loss of generality, we can assume $\Pi_{\mathrm{acc},u} = |0\rangle\langle 0| \otimes I$ for all $u \in V$, i.e., node u accepts the protocol iff the first qubit of register V_u is in the state $|0\rangle$.

2.2.2 Variants of the Definition

The above definition corresponds to the distributed quantum interactive proofs with private randomness where communication between nodes of the networks only happens after the interaction with the prover. This is the natural quantum analog of the definition of classical distributed interactive proofs by [18] given in Section 2.1.

A possible variant is distributed quantum interactive proofs with shared randomness, in which nodes are allowed to use shared randomness. In order to distinguish this model with the settings of private randomness, we denote it dQIP^{sh}. We denote $\mathsf{dQIP}^{sh}[k](f(n))$ the complexity class defined for this variant similarly to Definition 5, with the additional condition that at each turn the size of (shared) random bits sent to the prover is also $O(f(n))$-bit (i.e., each node u can send its message register and a random string s of size $O(f(n))$, but s must be the same as those of the other nodes).

Another variant, which we call dQIP^{com}, is the variant where nodes can communicate with each other in the middle of interaction with the prover. While in this paper we do not focus on this variant (since the classical version did not consider communication in the

[3] Concretely, simulating one random bit can be done by using the Bell pair $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ and keeping one qubit of the pair.

middle of the interaction with the prover either), we present a general result about this natural setting in Appendix C. We denote $\mathsf{dQIP}^{com}[k](f(n))$ the complexity class defined for this variant similarly to Definition 5.

3 General Turn Reduction Technique for Distributed Quantum Interactive Proofs

In this section we show a general reduction technique to reduce the number of turns by half while keeping the soundness parameter relatively low. The complexity only increases by the size of the private register (i.e., the amount of quantum memory used for local computation at each node).

3.1 Distributed Quantum Interactive Proofs with Shared Randomness

We first consider the case of dQIP^{sh} model, and show the following theorem.

▶ **Theorem 8.** *Let $\ell \geq 1$ be an integer, $\mathcal{L} \subseteq \mathcal{G} \times \mathcal{I}$ be a language that has a $\mathsf{dQIP}^{sh}[4\ell+1](f(n))$ protocol with completeness c and soundness s for some $c^2 > s$ where the protocol uses $g(n)$ space register at each node u in the interaction phase. Then $\mathcal{L}$ has a $\mathsf{dQIP}^{sh}[2\ell+1](f(n)+g(n))$ protocol with completeness $\frac{1+c}{2}$ and soundness $\frac{1+\sqrt{s}}{2}$.*

Proof. We will prove this theorem by adapting the method halving the number of turns of quantum interactive proofs given by [16] into the distributed setting. In their method, the prover first provides the snapshot state of the message register and the private register at (almost) half of turns in the original protocol. Then the verifier flips a coin and decides to execute either a forward-simulation (the simulation of the later half of the turns; $r = 0$ case in Figure 1) or a backward-simulation of the original protocol (the simulation of the inverse transformation of the first half of the turns; $r = 1$ case in Figure 1). The honest prover can perform the simulation according to the verifier's coin flip. On the other hand, due to the randomness of the verifier's choice, the malicious prover cannot fool the verifier. In order to implement this in the distributed setting, we only need to simulate the verifier's coin flip.

Let $\mathcal{L}$ be a language that has a k-turn dQIP^{sh} protocol where $k = 4\ell + 1$ for some integer $\ell \geq 1$, and let $Q = V_{k+1}P_k \cdots V_2 P_1$ be the unitary transform that is applied to register $(\mathsf{V}, \mathsf{M}, \mathsf{P})$ in the protocol. We show a $(2\ell + 1)$-turn dQIP^{sh} protocol in Figure 1.

$(2\ell + 1)$-turn dQIP^{sh} protocol.

1. The prover sends $(\mathsf{V}_u, \mathsf{M}_u)$ to the node u for all $u \in V$.
2. The network generates a random bit r and sends it to the prover.
3. **a.** The network applies $V_{2\ell+2}^{\dagger}$ to (V, M) if $r = 1$, and sends M.
   b. For $j = 1$ to $\ell - 1$ do the following:
   Each node u receives M_u from the prover. The network applies $V_{2\ell+2j+2}$ if $r = 0$, and $V_{2\ell-2j+2}^{\dagger}$ if $r = 1$. Then each node u sends M_u to the prover.
   c. Each node u receives M_u from the prover. The network applies $V_{4\ell+2}$ if $r = 0$, and applies $V_2^{\dagger}$ if $r = 1$. If $r = 0$, then each node u performs the POVM measurement $\{\Pi_{\mathrm{acc},u}, I - \Pi_{\mathrm{acc},u}\}$ on register $(\mathsf{V}_u, \mathsf{M}_u)$ and obtains its output. If $r = 1$, each node u measures the content of V_u and accepts iff the qubits in V_u are in $|0 \cdots 0\rangle$.

Figure 1 $(2\ell + 1)$-turn dQIP^{sh} protocol.

It is obvious that the protocol in Figure 1 is a $(2\ell+1)$-turn dQIP^{sh} protocol. The analysis of completeness and soundness can be shown in the same way as in Lemma 4.1 of [16]. ◀

Applying recursively Theorem 8 makes possible to reduce the number of turns down to 3: Let m be the minimum integer that satisfies $k \leq 2^m + 1$. Then, the number of turns can be reduced from k to 3 by applying Theorem 8 $m-1$ times. In Section 4 we discuss it more rigorously with mentioning error reduction.

3.2 Distributed Quantum Interactive Proofs without Shared Randomness

Next, we consider dQIP protocols and show the analogous result of Theorem 8 in the dQIP model. We can implement the protocol of Figure 1 in the dQIP model by simulating the step (2) of Figure 1 (the verifier's coin flip) without shared randomness. In order to simulate it, we need additional two turns: In the first turn the prover sends the information that represents a rooted spanning tree along with the snapshot state. The root (denoted by ℓ) of the spanning tree creates a Bell pair $\frac{1}{\sqrt{2}}|0\rangle_{\mathsf{M}_\ell}|0\rangle_{\mathsf{V}_\ell} + \frac{1}{\sqrt{2}}|1\rangle_{\mathsf{M}_\ell}|1\rangle_{\mathsf{V}_\ell}$ using one qubit of its message register and one qubit of its private register, then sends the message register to the prover in the second turn. The prover in the third turn creates $\frac{1}{\sqrt{2}}|0^n\rangle_{\mathsf{M}} + \frac{1}{\sqrt{2}}|1^n\rangle_{\mathsf{M}}$ using the CNOT gate, sends the register M_u to each u except the root ℓ, and keeps one-qubit. The construction of a rooted spanning tree in 1-turn requires $\Omega(\log n)$ witness size [19], but we can construct a rooted spanning tree in $O(1)$ witness size in 3-turn by the result of [25]. Therefore the size of witnesses is unchanged without a constant factor. (Note that the soundness error of the protocol of [25] is bounded by any small constant $\varepsilon > 0$, and this ε appears in Theorems 9 and 10.) We thus obtain the following theorem.

▶ **Theorem 9.** *Let $\ell \geq 1$ be an integer, $\mathcal{L} \subseteq \mathcal{G} \times \mathcal{I}$ be a language that has a $\mathsf{dQIP}[4\ell+1](f(n))$ protocol with completeness c and soundness s for some $c^2 > s$ where the protocol uses $g(n)$ space register at each node u in the interaction phase. Then $\mathcal{L}$ has a $\mathsf{dQIP}[2\ell+3](f(n)+g(n))$ protocol with completeness $\frac{1+c}{2}$ and soundness $\frac{1+\sqrt{s}}{2} + \varepsilon$ for arbitrary small constant $\varepsilon > 0$.*

Note that applying recursively Theorem 9 makes possible to reduce the number of turns down to 7 (corresponding to $\ell = 2$), but not lower: Let ℓ be the minimum integer that satisfies $k \leq 4\ell + 1$. Starting from k-turn, using Theorem 9, it is reduced to $2\ell + 3$. For $\ell' = \lfloor \frac{\ell}{2} \rfloor$, we have $4(\ell'+1)+1 \geq 2\ell+3$. Using Theorem 9 again, it is reduced down to $2(\ell'+1)+3$, which is at most $2\ell+1$ if $\ell \geq 3$. However if $\ell \leq 2$, we cannot reduce from $4\ell+1$ to $2\ell+1$ using Theorem 9, that is, the recursion stops at 7-turn. We now show the following theorem, which enables us to parallelize 7-turn protocols.

▶ **Theorem 10.** *Let $\mathcal{L} \subseteq \mathcal{G} \times \mathcal{I}$ be a language that has a $\mathsf{dQIP}[7](f(n))$ protocol with completeness c and soundness s where the protocol uses $g(n)$ space register at each node u in the interaction phase. Then $\mathcal{L}$ has a $\mathsf{dQIP}[5](f(n)+g(n))$ protocol with completeness $\frac{1+c}{2}$ and soundness $\frac{1+\sqrt{s}}{2} + \varepsilon$ for arbitrary small constant $\varepsilon > 0$.*

Proof. Fix the input x and a $\mathsf{dQIP}[7](f(n))$ protocol π for $\mathcal{L}$ described by a sequence of unitaries (corresponding to the interactions between the verifier and the honest prover) $P_1, V_2, P_3, V_4, P_5, V_6, P_7, V_8$ in this order, which has completeness c and soundness s. Our converted protocol is shown in Figure 2 (we call this protocol π'). We denote $\mathsf{R}_1 = \{\mathsf{R}_{u,1}\}_{u \in V}$ and $\mathsf{R}_2 = \{\mathsf{R}_{u,2}\}_{u \in V}$. In π', we consider the entire register is $(\mathsf{P}, \mathsf{R}_1, \mathsf{R}_2)$ where P is the prover's private register: Initially, there is no verifier's private register, and after receiving R_1, the private register of each node u is $\mathsf{R}_{u,1}$. Here we analyze the completeness and the soundness of π'. Define two quantum states $|\psi_4\rangle = V_4 P_3 V_2 P_1 |0 \cdots 0\rangle_{(\mathsf{P},\mathsf{R}_1,\mathsf{R}_2)}$ and $|\psi_5\rangle = P_5|\psi_4\rangle$.(Here we abuse the notation by thinking unitaries V_i act on both (M, V) and $(\mathsf{R}_1, \mathsf{R}_2)$, and also unitaries P_i act on both (P, M) and $(\mathsf{P}, \mathsf{R}_2)$ since they have the same size.)

Proof of completeness. Assume that $x \in \mathcal{L}$. The honest prover does the following.

- **Turn 1:** Prepare the quantum state $|\psi_4\rangle$ in the register $(\mathsf{P}, \mathsf{R}_1, \mathsf{R}_2)$. Send the register R_1.
- **Turn 3:** Broadcast b. If $b = 0$, apply the honest operation P_5 and send the register R_2. If $b = 1$, send the register R_2.
- **Turn 5:** If $b = 0$, apply the honest operation P_7. If $b = 1$, apply $P_3^\dagger$ (the inverse operation of the operation P_3).

After Step 5 of Figure 2, if $b = 0$, the entire quantum state is $V_8P_7V_6P_5V_4P_3V_2P_1 |0\cdots0\rangle$ and if $b = 1$, the entire quantum state is $P_1 |0\cdots0\rangle = (P_1 |0\cdots0\rangle_{(\mathsf{P},\mathsf{R}_2)}) \otimes |0\cdots0\rangle_{\mathsf{R}_1}$. Thus the acceptance probability of π' is $\frac{1+c}{2}$.

Proof of soundness. Assume that $x \notin \mathcal{L}$. Let $|\psi\rangle$ be the initial state in $(\mathsf{P}, \mathsf{R}_1, \mathsf{R}_2)$, that is, in Turn 1 of π', the verifier receives the register R_1 and its reduced state is $\mathrm{tr}_{(\mathsf{P},\mathsf{R}_2)}(|\psi\rangle\langle\psi|)$. Assume that, when the random bit in Turn 2 is $b = i$, the prover applies $U_i \otimes I_{\mathsf{R}_1}$ and sends R_2 in Turn 3, and applies $W_i \otimes I_{\mathsf{R}_1}$ and sends R_2 in Turn 5. Define unitaries Q_0 and Q_1 by $Q_0 = (I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_8)(W_0 \otimes I_{\mathsf{R}_1})(I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_6)(U_0 \otimes I_{\mathsf{R}_1})$ and $Q_1 = (I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_2^\dagger)(W_1 \otimes I_{\mathsf{R}_1})(I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_4^\dagger)(U_1 \otimes I_{\mathsf{R}_1})$, and let

$$|\alpha\rangle = \frac{1}{\|\Pi_{acc}Q_0 |\psi\rangle\|}\Pi_{acc}Q_0 |\psi\rangle \text{ and } |\beta\rangle = \frac{1}{\|\Pi_{init}Q_1 |\psi\rangle\|}\Pi_{init}Q_1 |\psi\rangle,$$

where Π_{acc} is the projection onto the acceptance state of π, and $\Pi_{init} = I_{(\mathsf{P},\mathsf{R}_2)} \otimes |0\cdots0\rangle\langle0\cdots0|_{\mathsf{R}_1}$.

Let p_i be the acceptance probability of π' when the random bit in Turn 2 is $b = i$. Then we have

$$\begin{aligned}
p_0 &= \|\Pi_{acc}Q_0 |\psi\rangle\|^2 = \frac{1}{\|\Pi_{acc}Q_0 |\psi\rangle\|}|\langle\psi| Q_0^\dagger\Pi_{acc}Q_0 |\psi\rangle|^2 = F(|\alpha\rangle\langle\alpha|, Q_0 |\psi\rangle\langle\psi| Q_0^\dagger)^2 \\
&= F(Q_0^\dagger |\alpha\rangle\langle\alpha| Q_0, |\psi\rangle\langle\psi|)^2, \\
p_1 &= \|\Pi_{init}Q_1 |\psi\rangle\|^2 = \frac{1}{\|\Pi_{init}Q_1 |\psi\rangle\|}|\langle\psi| Q_1^\dagger\Pi_{init}Q_1 |\psi\rangle|^2 = F(|\beta\rangle\langle\beta|, Q_1 |\psi\rangle\langle\psi| Q_1^\dagger)^2 \\
&= F(Q_1^\dagger |\beta\rangle\langle\beta| Q_1, |\psi\rangle\langle\psi|)^2.
\end{aligned}$$

Therefore, using Lemma 21 the acceptance probability p_{acc} of π' is bounded by

$$\begin{aligned}
p_{acc} = \frac{1}{2}(p_0 + p_1) &\leq \frac{1}{2}(1 + F(Q_0^\dagger |\alpha\rangle\langle\alpha| Q_0, Q_1^\dagger |\beta\rangle\langle\beta| Q_1)) \\
&= \frac{1}{2}(1 + F(|\alpha\rangle\langle\alpha|, Q_0Q_1^\dagger |\beta\rangle\langle\beta| Q_1Q_0^\dagger)).
\end{aligned}$$

We also have

$$F(|\alpha\rangle\langle\alpha|, Q_0Q_1^\dagger |\beta\rangle\langle\beta| Q_1Q_0^\dagger) = |\langle\alpha| Q_0Q_1^\dagger |\beta\rangle| = |\langle\alpha|\Pi_{acc}Q_0Q_1^\dagger |\beta\rangle| \leq \|\Pi_{acc}Q_0Q_1^\dagger |\beta\rangle\|$$

from the fact that $\Pi_{acc} |\alpha\rangle = |\alpha\rangle$. The reduced state of $|\beta\rangle$ satisfies $\mathrm{tr}_{(\mathsf{P},\mathsf{R}_2)}(|\beta\rangle\langle\beta|) = |0\cdots0\rangle\langle0\cdots0|_{\mathsf{R}_1}$ since $\Pi_{init} |\beta\rangle = |\beta\rangle$. Therefore, from the soundness of π, for any U_0, U_1, W_0, W_1 acting on $(\mathsf{P}, \mathsf{R}_2)$,

$$\begin{aligned}
&\|\Pi_{acc}(I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_8)(W_0 \otimes I_{\mathsf{R}_1})(I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_6)(U_0U_1^\dagger \otimes I_{\mathsf{R}_1})(I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_4)(W_1 \otimes I_{\mathsf{R}_1})(I_{(\mathsf{P},\mathsf{R}_2)} \otimes V_2) |\beta\rangle\|^2 \\
&= \|\Pi_{acc}Q_0Q_1^\dagger |\beta\rangle\|^2 \leq s.
\end{aligned}$$

Thus we have $p_{acc} \leq \frac{1}{2} + \frac{\sqrt{s}}{2}$, which completes the proof of soundness. ◀

dQIP[5]($f(n)+g(n)$) protocol π'.

1. **Turn 1:** The prover gives a $g(n)$-qubit quantum register $\mathsf{R}_{u,1}$ to each node u. The prover chooses arbitrary one node as a leader node leader.
2. **Turn 2:** leader chooses a bit $b \in \{0,1\}$ uniformly at random and sends it to the prover.
3. **Turn 3:** The prover sends one bit b_u and a $f(n)$-qubit quantum register $\mathsf{R}_{u,2}$ to each node u.
4. **Turn 4:** If $b_u = 0$, each node u applies $V_{u,6}$ to $(\mathsf{R}_{u,1}, \mathsf{R}_{u,2})$. If $b_u = 1$, u applies $V_{u,4}^{\dagger}$. u sends $\mathsf{R}_{u,2}$ to the prover.
5. **Turn 5:** The prover sends $\mathsf{R}_{u,2}$ to each node u.
6. **The verification phase:** If $b_u = 0$, each node u applies $V_{u,8}$ to $(\mathsf{R}_{u,1}, \mathsf{R}_{u,2})$ and u does the same verification as in the original 7-turn protocol. u outputs "accept" iff the output of the original protocol is "accept" and all random bits b_v where $v \in N(u)$ are the same as b_u. If $b_u = 1$, u applies $V_{u,2}^{\dagger}$ and outputs "accept" iff the register $\mathsf{R}_{u,1}$ is set to all-zero state and all random bits b_v where $v \in N(u)$ are the same as b_u.

Figure 2 dQIP[5]($f(n)+g(n)$) protocol π'.

dQIP protocol simulating a dAM protocol.

1. **The prover's turn:** The prover applies arbitrary unitary U_j to the register (P, M). Then M_u is sent to the node u.
2. **The verifier's turn:** Each node u stores the state in M_u to its private register V_u by the SWAP gate, and creates $\frac{1}{\sqrt{2^m}} \sum_{r\in\{0,1\}^m} |r\rangle_{\mathsf{M}_u} |r\rangle_{\mathsf{V}_u}$ in M_u and a fresh part of V_u. Then M_u is sent to the prover.
3. **The verification phase:** Each node measures its private register in the computational basis, then broadcasts the outcome to its neighbors, and decides the output of the protocol (accept or reject).

Figure 3 dQIP protocol simulating a dAM protocol.

4 Quantum Simulation of Distributed Arthur-Merlin Interactive Protocols

In this section we see how to convert dAM protocols to dQIP protocols, and parallelize the converted dQIP protocol to 5-turn. Assume the size of each witness and random bits is m. Let c_j be a function that represents the witnesses provided by Merlin at the j-th turn. That is, if random bits generated by Arthur in the i-th turn is $r_i = r_i(u_1)r_i(u_2)\cdots r_i(u_n) \in \{0,1\}^{mn}$, $c_j(r_2, r_4, \ldots, r_{j-1}) = c_j(u_1)c_j(u_2)\cdots c_j(u_n) \in \{0,1\}^{mn}$ represents the witness where $c_j(u)$ is provided to u. In order to simulate k-turn dAM protocols, each computation by the prover has to be converted to a form of reversible computation. Thus c_j must be realized by a unitary transform

$$U_{c_j} : |r_2, ..., r_{j-1}, b\rangle \to |r_2, ..., r_{j-1}, b \oplus c_j\rangle .$$

The protocol proceeds as Figure 3. Let c and s be the completeness and the soundness of the original dAM[k] protocol, respectively. We show the following theorem.

▶ **Theorem 11.** *The protocol in Figure 3 has completeness c and soundness s.*

Proof.

Completeness. Assume that $(G, I) \in \mathcal{L}$ and the prover receives the M_u part of the quantum state $\frac{1}{\sqrt{2^m}} \sum_{r_{j-1}(u) \in \{0,1\}^m} |r_{j-1}(u)\rangle_{\mathsf{M}_u} |r_{j-1}(u)\rangle_{\mathsf{V}_u}$ from the node u in the $(j-1)$-th turn. At the j-th turn the honest prover applies the SWAP gate to (P, M), obtaining

$$\frac{1}{\sqrt{2^{\frac{j-1}{2}mn}}} \left(\sum_{r_2, r_4, ..., r_{j-1} \in \{0,1\}^{mn}} |r_2, r_4, ..., r_{j-1}\rangle_{\mathsf{P}} |0\rangle_{\mathsf{M}} |c_1, r_2, c_3, r_4, ..., c_{j-2}, r_{j-1}\rangle_{\mathsf{V}} \right).$$

Then, the prover also applies U_{c_j} to $(\mathsf{P}, \mathsf{M}_u)$, obtaining the following state

$$\begin{aligned} &\frac{1}{\sqrt{2^{\frac{j-1}{2}mn}}} \left(\sum_{r_2, r_4, ..., r_{j-1} \in \{0,1\}^{mn}} |r_2, r_4, ..., r_{j-1}\rangle_{\mathsf{P}} |c_j\rangle_{\mathsf{M}} |c_1, r_2, ..., c_{j-2}, r_{j-1}\rangle_{\mathsf{V}} \right) \\ &= \frac{1}{\sqrt{2^{\frac{j-1}{2}mn}}} \left(\sum_{r_2, r_4, ..., r_{j-1} \in \{0,1\}^{mn}} |r_2, r_4, ..., r_{j-1}\rangle_{\mathsf{P}} \bigotimes_{u \in V} |c_j(u)\rangle_{\mathsf{M}_u} |c_1, ..., r_{j-1}\rangle_{\mathsf{V}} \right). \end{aligned}$$

At the verification phase, the quantum state in V is a mixed state

$$\frac{1}{2^{\frac{k-1}{2}mn}} \sum_{r_2, ..., r_{k-1} \in \{0,1\}^{mn}} |c_1, r_2, c_3, ..., r_{k-1}, c_k\rangle \langle c_1, r_2, c_3, ..., r_{k-1}, c_k|_{\mathsf{V}},$$

and u obtains one of the state $|c_1(u), r_2(u), c_3(u), ..., r_{k-1}(u), c_k(u)\rangle$ uniformly at random as the outcome of its measurement. Then u broadcasts the outcome to its adjacent nodes. From the completeness of the original $\mathsf{dAM}[k]$ protocol the acceptance probability of this verification phase is at least c.

Soundness. Assume that $(G, I) \notin \mathcal{L}$. A malicious prover may use some other unitary instead of U_{c_j} at the j-th turn. Let $\sum_{r_{j-1} \in \{0,1\}^{mn}} |r_{j-1}\rangle_{\mathsf{M}} |r_{j-1}\rangle_{\mathsf{V}}$ be the Bell pairs created by the verifier in the $(j-1)$-th turn. The witness provided by the prover in the j-th turn is stored into the private register V. In the verification phase, node u_i obtains $|x_i, r_{j-1}(u)\rangle$ for some $x_i \in \{0,1\}^m$ as the outcome of its measurement. From the soundness of dAM protocols, the original protocol is accepted for at most s of all random strings generated by Arthur. On the other hand, each node u obtains $|r_2(u), r_4(u), ..., r_{k-1}(u)\rangle$ uniformly at random by its measurement regardless of the prover's action. Thus the acceptance probability of this dQIP protocol is at most s. ◀

Using this theorem and the results in Section 3, we can show Theorems 1 and 2.

Proofs of Theorem 1 and Theorem 2. Assume without loss of generality $k = 4\ell+1$ for $\ell \geq 1$ and $m = f(n)$. For any $\mathsf{dAM}[k](m)$ protocol, we have a $\mathsf{dQIP}[k](m)$ protocol which simulates it using Theorem 11. Applying Theorem 9 we can parallelize it to a $\mathsf{dQIP}[2\ell+3](m)$ protocol with completeness $\frac{1+c}{2}$ and soundness $\frac{1+\sqrt{s}}{2}$. Note that we can assume $g(n) = O(m)$ since the register provided by the honest prover at the first turn of the parallelized protocol contains a $(4\ell+1)mn$-qubit state in registers V and M such that the total state is represented as

$$\frac{1}{\sqrt{2^{(\ell+1)mn}}} \sum_{r_2, ..., r_{2\ell+2} \in \{0,1\}^{mn}} |r_2, \ldots, r_{2\ell+2}\rangle_{\mathsf{P}} |c_1, r_2, \ldots, c_{2\ell+1}, r_{2\ell+2}, 0^{mn}, \cdots, 0^{mn}\rangle_{(\mathsf{V},\mathsf{M})}$$

and each node u receives its reduced state of $(4\ell+1)m = O(m)$-qubit on $(\mathsf{V}_u, \mathsf{M}_u)$. Thus the size of witnesses and the size of messages in the verification phase are both $O(m)$. We assume that the parameters c and s of the original $\mathsf{dAM}[k](m)$ protocol are $c = 1-\varepsilon$ and $s = \delta$ for small constant $\varepsilon > 0$ and $\delta > 0$ since we can use the standard technique of parallel repetition by [3] (the protocol is executed in parallel a constant number of times, and the leader node, which is determined by the prover as a root of a spanning tree, adopts the majority of the outcomes in the verification phase). Note that the witness size does not change by parallel repetition since the protocol has $k \geq 3$ turn and the construction of a spanning tree can be done with $O(1)$-size witnesses using three turns [25]. (Here, the soundness error ε' of the construction of the spanning tree affects the analysis, so we set the soundness $s+\varepsilon'$ for the parallel repetition.) Now we assume that the completeness and the soundness of the original $\mathsf{dAM}[k](m)$ protocol are $c = 1-\frac{1}{12a^2}$ and $s = \frac{1}{12a^2}$ for $a = k-1$, respectively. By Theorem 11, the converted $\mathsf{dQIP}[k](m)$ protocol has the same completeness and soundness. By Theorem 9, which reduces the number of turns down to 7, and Theorem 10, which reduces the number of turns down to 5, and the same analysis as Lemma 4.2 of [16], we get a $\mathsf{dQIP}[5](m)$ protocol with completeness $1-\frac{2c}{k-1} = 1-\frac{1}{6a^3}$ and soundness $1-\frac{1-s}{(k-1)^2} < 1-\frac{1}{2a^2}$. Then we use another parallel repetition for quantum interactive protocols developed in [11]. (The parallel repetition in [11] accepts iff all outcomes in repetitions are "accept". See Theorem 4.9 in [11].) More precisely, using $2a^3$ time repetitions the completeness and soundness become $\left(1-\frac{1}{6a^3}\right)^{2a^3} > 1-\frac{1}{3} = \frac{2}{3}$ and $\left(1-\frac{1}{2a^2}\right)^{2a^3} < \frac{1}{e^a} < \frac{1}{3}$, which completes the proof of Theorem 1.

In the case of dQIP^{sh}, we can parallelize a $k = (4\ell+1)$-turn protocol to a $(2\ell+1)$-turn protocol by using Theorem 8. Therefore Theorem 2 can be shown similarly to the proof of Theorem 1 by applying Theorem 8, instead of Theorem 9 and Theorem 10. ◀

4.1 Applications of Theorem 1 and 2

We can apply Theorem 1 and Theorem 2 to the following dAM protocols by [25] (see Appendix A for the definition of these problems), obtaining Corollary 3:

▶ **Theorem 12** ([25]). *There exist*

- *a* $\mathsf{dAM}[4](\log n)$ *protocol for* Asym,
- *a* $\mathsf{dAM}[O(1)](\log n)$ *protocol for GNI,*
- *a* $\mathsf{dAM}[5](\log\log n)$ *protocol for* SetEquality,
- *a* $\mathsf{dAM}[5](\log\log n)$ *protocol for* DSym.

▶ **Theorem 13** ([25]). *There exists a constant* δ *such that if a language* $\mathcal{L}$ *can be decided in* $\mathrm{poly}(n)$ *time and* n^δ *space, then* $\mathcal{L} \in \mathsf{dAM}[O(1)](\log n)$.

▶ **Corollary 3.** 1. *There exist*

- *a* $\mathsf{dQIP}^{sh}[3](\log n)$ *protocol for* Asym,
- *a* $\mathsf{dQIP}^{sh}[3](\log n)$ *protocol for GNI,*
- *a* $\mathsf{dQIP}^{sh}[3](\log\log n)$ *protocol for* SetEquality,
- *a* $\mathsf{dQIP}^{sh}[3](\log\log n)$ *protocol for* DSym.
- *a* $\mathsf{dQIP}[5](\log n)$ *protocol for GNI.*

2. *There exists a constant* δ *such that if a language* $\mathcal{L}$ *can be decided in* $\mathrm{poly}(n)$ *time and* n^δ *space, then* $\mathcal{L} \in \mathsf{dQIP}[5](\log n)$ *and* $\mathcal{L} \in \mathsf{dQIP}^{sh}[3](\log n)$.

5 Testing Closeness of Two Quantum States

Verification of the GHZ state

In this section we briefly explain how to show Thereom 4. Technically, our protocol can be viewed as the distributed implementation of the SWAP test [1]. To do this, we need to implement the controlled SWAP gate, but it is not possible by local operations at each node if the inputs are distributed since there is no prior entanglement in our setting. To resolve this issue, we create the quantum state that is called the GHZ state using the prover. Let $|GHZ\rangle$ be the n-qubit GHZ state

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0^n\rangle + |1^n\rangle).$$

The detail of our approach is omitted from the main body of the paper due to space constraint. It can be found in the full version [9]. Ultimately, we present a dQIP protocol for verification of the GHZ state, and show the following theorem.

▶ **Theorem 16.** *There exists a* dQIP$[5](O(1))$ *protocol* $\mathcal{P}_{GHZ}$ *that satisfies the following properties:*

- *(completeness): If the prover is honest, the protocol accepts with probability 1 and the network outputs the GHZ state.*
- *(soundness): If the protocol accepts with probability* δ*, then the reduced state* ρ *of the output register satisfies*

$$\langle GHZ|\,\rho\,|GHZ\rangle \geq 1 - \varepsilon.$$

The dQIP protocol for DQCT_N

In the full version [9], using the protocol $\mathcal{P}_{GHZ}$, we present a protocol $\mathcal{P}_{\mathsf{DQCT}}$, which satisfies the desired conditions appeared in Theorem 4.

References

1 Harry Buhrman, Richard Cleve, John Watrous, and Ronald De Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001.

2 Keren Censor-Hillel, Orr Fischer, François Le Gall, Dean Leitersdorf, and Rotem Oshman. Quantum Distributed Algorithms for Detection of Cliques. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, pages 35:1–35:25, 2022.

3 Pierluigi Crescenzi, Pierre Fraigniaud, and Ami Paz. Trade-Offs in Distributed Interactive Proofs. In *Proceedings of the 33rd International Symposium on Distributed Computing (DISC 2019)*, pages 13:1–13:17, 2019.

4 Pierre Fraigniaud, François Le Gall, Harumichi Nishimura, and Ami Paz. Distributed Quantum Proofs for Replicated Data. In *Proceedings of the 12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, pages 28:1–28:20, 2021.

5 Pierre Fraigniaud, Amos Korman, and David Peleg. Local distributed decision. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 708–717, 2011.

6 Pierre Fraigniaud, Pedro Montealegre, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. On distributed Merlin-Arthur decision protocols. In *Proceedings of the International Colloquium on Structural Information and Communication Complexity (SIROCCO 2019)*, pages 230–245, 2019.

7 Christopher A Fuchs and Jeroen Van De Graaf. Cryptographic distinguishability measures for quantum-mechanical states. *IEEE Transactions on Information Theory*, 45(4):1216–1227, 1999.

8 François Le Gall, Masayuki Miyamoto, and Harumichi Nishimura. Distributed Merlin-Arthur Synthesis of Quantum States and Its Applications, 2022. `arXiv:2210.01389`.

9 François Le Gall, Masayuki Miyamoto, and Harumichi Nishimura. Distributed Quantum Interactive Proofs, 2022. `arXiv:2210.01390`.

10 Mika Göös and Jukka Suomela. Locally checkable proofs in distributed computing. *Theory of Computing*, 12(1):1–33, 2016.

11 Gus Gutoski. Quantum strategies and local operations. *arXiv preprint*, 2010. `arXiv:1003.0038`.

12 Taisuke Izumi, François Le Gall, and Frédéric Magniez. Quantum Distributed Algorithm for Triangle Finding in the CONGEST Model. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, pages 23:1–23:13, 2020.

13 Taisuke Izumi and François Le Gall. Quantum distributed algorithm for the all-pairs shortest path problem in the congest-clique model. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC 2019)*, pages 84–93, 2019.

14 Rahul Jain, Zhengfeng Ji, Sarvagya Upadhyay, and John Watrous. QIP= PSPACE. *Journal of the ACM*, 58(6):1–27, 2011.

15 Benjamin Jauregui, Pedro Montealegre, and Ivan Rapaport. Distributed interactive proofs for the recognition of some geometric intersection graph classes. In *Proceedings of 29th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2022)*, pages 212–233, 2022.

16 Julia Kempe, Hirotada Kobayashi, Keiji Matsumoto, and Thomas Vidick. Using entanglement in quantum multi-prover interactive proofs. *Computational Complexity*, 18(2):273–307, 2009.

17 Alexei Kitaev and John Watrous. Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing (STOC 2000)*, pages 608–617, 2000.

18 Gillat Kol, Rotem Oshman, and Raghuvansh R Saxena. Interactive distributed proofs. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 255–264, 2018.

19 Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

20 François Le Gall and Frédéric Magniez. Sublinear-time quantum computation of the diameter in congest networks. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (PODC 2018)*, pages 337–346, 2018.

21 François Le Gall, Harumichi Nishimura, and Ansis Rosmanis. Quantum Advantage for the LOCAL Model in Distributed Computing. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, pages 49:1–49:14, 2019.

22 Chris Marriott and John Watrous. Quantum Arthur–Merlin games. *Computational Complexity*, 14(2):122–152, 2005.

23 Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. Shared vs Private Randomness in Distributed Interactive Proofs. In *Proceedings of the 31st International Symposium on Algorithms and Computation (ISAAC 2020)*, pages 51:1–51:13, 2020.

24 Pedro Montealegre, Diego Ramírez-Romero, and Ivan Rapaport. Compact distributed interactive proofs for the recognition of cographs and distance-hereditary graphs. In *Proceedings of the International Symposium on Stabilizing, Safety, and Security of Distributed Systems (SSS 2021)*, pages 395–409, 2021.

25 Moni Naor, Merav Parter, and Eylon Yogev. The power of distributed verifiers in interactive proofs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 1096–115, 2020.

26 Ashwin Nayak and Peter Shor. Bit-commitment-based quantum coin flipping. *Physical Review A*, 67(1):012304, 2003.

27 Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

28 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

29 Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM Journal on Computing*, 41(5):1235–1265, 2012.

30 Robert W Spekkens and Terry Rudolph. Degrees of concealment and bindingness in quantum bit commitment protocols. *Physical Review A*, 65(1):012310, 2001.

31 John Watrous. PSPACE has constant-round quantum interactive proof systems. *Theoretical Computer Science*, 292(3):575–588, 2003.

32 John Watrous. *The Theory of Quantum Information*. Cambridge University Press, 2018.

33 Mark M. Wilde. *Quantum Information Theory*. Cambridge University Press, 2017.

34 Huangjun Zhu and Masahito Hayashi. Efficient verification of hypergraph states. *Physical Review Applied*, 12(5):054047, 2019.

A Problems

In this appendix we formally define the problems Set Equality, Graph Asymmetry, Dumbbell Symmetry and Graph Non-Isomorphism.

▶ **Definition 17** (Set Equality [25]). *Let G be a graph and I be an input such that the label $I(u)$ for each node u contains two lists of ℓ elements $\mathcal{A}_u = \{a_{u,1}, \dots, a_{u,\ell}\}$ and $\mathcal{B}_u = \{b_{u,1}, \dots, b_{u,\ell}\}$ where $\ell \leq n$ is an integer and each element in $\mathcal{A}_u$ and $\mathcal{B}_u$ can be represented in $O(\log n)$-bit. The language* SetEquality *is the set of graphs and labels such that $\{\mathcal{A}_u\}_{u\in V} = \{\mathcal{B}_u\}_{u\in V}$ as multisets.*

▶ **Definition 18** (Graph Asymmetry [25]). *The language* Asym *is the set of all connected graphs that do not have a nontrivial automorphism.*

▶ **Definition 19** (Dumbbell Symmetry [18]). *Let m, k be positive integers and let $n = 2m + 2k + 1$. An n-vertex connected graph $G = (\{0, 1, \dots, n-1\}, E)$ is a dumbbell graph if it satisfies following conditions:*

- *Let G_0 be the vertex-induced subgraph of G on vertices $\{0, \dots, m-1\}$ and G_1 be the vertex-induced subgraph of G on vertices $\{m, \dots, 2m-1\}$.*
- *G_0 and G_1 are connected to each other by the following path of length $2k+2$*
$$0 - (2m) - (2m+1) - \cdots - (2m+2k) - (m).$$
- *E consists of all edges in G_0 and G_1, and the path-edges.*

The automorphism σ is given as follows:

$$\sigma(i) = \begin{cases} m+i & \text{if } i \in \{0, \dots, m-1\} \\ i-m & \text{if } i \in \{m, \dots, 2m-1\} \\ 4m+2k-i & \text{if } i \in \{2m, \dots, 2m+2k\} \end{cases}$$

The language DSym *is the set of all dumbbell graphs G such that $\sigma(G)$ is isomorphic to G.*

▶ **Definition 20** (Graph Non-Isomorphism [18]). *The language GNI is the set of all pairs of graphs (G_0, G_1) where G_0 is not isomorphic to G_1. We assume that the communication graph is G_0, and nodes cannot communicate on G_1-edges.*

In [25], it was shown that SetEquality $\in$ dAM[2]($\log n$), SetEquality $\in$ dAM[4]($\log \log n$), Asym $\in$ dAM[4]($\log n$), DSym $\in$ dAM[4]($\log \log n$), GNI $\in$ dAM[k]($\log n$) for some constant $k > 4$. [4] For GNI, there also exists a dAM[4]($n \log n$) protocol showed by [18].

B Quantum information

We assume that the readers are familiar with basic concepts of quantum information such as density matrices, measurements, and quantum circuits (See, e.g., [28, 32, 33]).

Let $\mathcal{H}$ be a finite dimensional Hilbert space, and ρ, σ be any quantum states in $\mathcal{H}$. The fidelity of two quantum states ρ, σ is defined as $F(\rho, \sigma) = \mathrm{tr}\left[\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}}\right]$. Note that for two pure states $\rho = |\psi_\rho\rangle\langle\psi_\rho|$ and $\sigma = |\psi_\sigma\rangle\langle\psi_\sigma|$ we have $F(\rho, \sigma) = |\langle\psi_\rho|\psi_\sigma\rangle|$. Let $\mathrm{dist}(\rho, \sigma) = \frac{1}{2}\|\rho - \sigma\|_{\mathrm{tr}}$ be the trace distance of ρ, σ, where $\|A\|_{\mathrm{tr}} = \mathrm{tr}\sqrt{A^\dagger A}$. For two pure states $|\psi\rangle\langle\psi|, |\phi\rangle\langle\phi|$ we denote $\mathrm{dist}(|\psi\rangle, |\phi\rangle)$. Here we summarize some useful inequalities about the fidelity and the trace distance, which are used multiple times in this paper.

▶ **Lemma 21.** *For any quantum states ρ, σ, ξ in $\mathcal{H}$, we have*

1. [7]: $1 - F(\rho, \sigma) \leq \mathrm{dist}(\rho, \sigma) \leq \sqrt{1 - F(\rho, \sigma)^2}$,
2. [26, 30]: $F(\rho, \sigma)^2 + F(\xi, \sigma)^2 \leq 1 + F(\rho, \xi)$.

A *POVM (Positive Operator Valued Measure)*, which is a general framework for quantum measurement, consists of a set of positive semi-definite matrices $\{M_m\}_m$ that satisfies $\sum_m M_m = I$. If a quantum state ρ is measured by a POVM $\{M_m\}_m$, the outcome i of the measurement is obtained with probability $\mathrm{tr}(M_i\rho)$. For arbitrary system of n-qubit, the quantum measurement corresponds to the POVM $\{M_m\}_m$ where $M_m = |m\rangle\langle m|$ for $m \in \{0, 1, \ldots, 2^n - 1\}$ is called the measurement in the computational basis.

The SWAP gate U_{swap} is the gate acting on two-qubit as $U_{swap}|a\rangle|b\rangle = |b\rangle|a\rangle$ for $a, b \in \{0, 1\}$. The SWAP test is a basic tool for quantum computing, which acts on the two registers $\mathsf{R}_1, \mathsf{R}_2$ as follows: (1) Prepare the state $|+\rangle$ on the 1-qubit ancilla register B; (2) Perform the SWAP gate on $(\mathsf{R}_1, \mathsf{R}_2)$ iff the qubit in B is $|1\rangle$; (3) Perform Hadamard gate on B and measure it in the computational basis $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$.

C Perfect completeness

In this appendix we consider the dQIP^{com} model, and show how to transform a protocol with two-sided bounded error into a protocol with perfect completeness. The number of turns of the transformed protocol increases by four turns and the size of message registers remains unchanged. This is shown by implementing the result of [17] in a distributed manner. The main difference from the centralized setting is the rejection condition.

In the case of the distributed setting, assume each node outputs 0 if it accepts otherwise outputs 1. Then the protocol rejects iff the output is not all-zero. The method of Kitaev and Watrous in the case of the centralized setting includes the operation that an additional register is prepared by the verifier, and it is incremented iff the private register is in the rejection state (i.e., $|1\rangle$). In the distributed setting, the prover determines the leader node, and the leader performs this operation. However, as mentioned above, in the case of distributed setting, the protocol rejects even if the leader is in the acceptance state but some other node is in the rejection state. Hence the leader needs the help of the prover to confirm that the protocol is in the rejection state. These operations require four additional turns but do not change the size of the message register.

[4] If we add the condition that the nodes can communicate on G_1-edges, there is a dAM[4]($\log n$) protocol [25].

Let $\mathcal{L}$ be a language that has a k-turn dQIP^{com} system with completeness c and soundness s where $c - s > \delta$ for some constant $\delta > 0$. We show a distributed implementation of Kitaev and Watrous in Figure 4. Note that this protocol works for dQIP^{com} but does not work for dQIP (and dQIP^{sh}) since nodes need to communicate with each other in the middle of the protocol in order to check if its private register is in the acceptance state. The following theorem can be shown easily by adapting the proof of [17] to the distributed setting.

▶ **Theorem 22.** *Any $\mathsf{dQIP}^{com}[k](f(n))$ protocol with completeness c and soundness s where $c-s > \delta$ for some constant $\delta > 0$ that uses $(g(n)+f(n)\deg(u))$-qubit private register at node u can be transformed to a $\mathsf{dQIP}^{com}[k+4](f(n)\Delta+g(n))$ protocol that uses $(g(n)+f(n)\deg(u))$-qubit private register at node u, with perfect completeness and soundness $1-\delta^2+\varepsilon$ where Δ is the maximum degree of the network and $\varepsilon > 0$ is arbitrary small constant.*

Proof. We call the protocol of Figure 4 $\mathcal{P}$ in order to distinguish from the original protocol. The proof uses almost the same argument from Ref. [17]. We assume that in the original protocol each node u accepts iff the first qubit of its private register V_u is $|0\rangle$. Then, in step 3, $\mathcal{P}$ is rejected when one of the following conditions holds; (1) the output qubits is not in $|0^n\rangle$, but the state in $\mathsf{O} = \{\mathsf{O}_u\}_{u\in V}$, which is sent from the prover, is $|0^n\rangle$ (it means the case that at least one node rejects the original protocol); (2) the state in O is neither $|0^n\rangle$ nor $|1^n\rangle$. Now we only consider the case that the protocol $\mathcal{P}$ is not rejected in step 3. Let $|\psi\rangle = \alpha_{acc}|\psi_{acc}\rangle + \alpha_{rej}|\psi_{rej}\rangle$ be the state in (V,W) before the measurement of the original protocol where $|\psi_{acc}\rangle$ represents the state that the output qubits of all nodes is $|0\rangle$ and $|\psi_{rej}\rangle$ represents the state that the output qubit of at least one node is $|1\rangle$. Thus, after step 3, the state[5] in $(\mathsf{O},\mathsf{V},\mathsf{W})$ is written as

$$\alpha_{acc}|0^n\rangle_{\mathsf{O}}|\psi_{acc}\rangle + \alpha_{rej}|1^n\rangle_{\mathsf{O}}|\psi_{rej}\rangle.$$

Let U be the unitary that is applied by the prover between step 5 and 6. The state in $(\mathsf{O}',\mathsf{O},\mathsf{V},\mathsf{W})$ before step 6 is then

$$\alpha_{acc}|0\rangle_{\mathsf{O}'}U(|0^n\rangle_{\mathsf{O}}|\psi_{acc}\rangle) + \alpha_{rej}|1\rangle_{\mathsf{O}'}U(|1^n\rangle_{\mathsf{O}}|\psi_{rej}\rangle).$$

In step 6, the leader subtracts O' from O_ℓ, yielding the state

$$\alpha_{acc}|0\rangle_{\mathsf{O}'}|\phi_{acc}\rangle + \alpha_{rej}|1\rangle_{\mathsf{O}'}|\phi_{rej}\rangle,$$

where $|\phi_{acc}\rangle$ and $|\phi_{rej}\rangle$ are some states in $(\mathsf{O},\mathsf{V},\mathsf{W})$. The leader applies the operation T_c and accepts with probability $\|\alpha_{acc}\sqrt{c}|\phi_{acc}\rangle + \alpha_{rej}\sqrt{1-c}|\phi_{rej}\rangle\|^2$. For the completeness, we have $\alpha_{acc} = \sqrt{c}$ and $\alpha_{rej} = \sqrt{1-c}$. If the honest prover performs the operation that makes $|\phi_{acc}\rangle = |\phi_{rej}\rangle$, the acceptance probability is 1. For the soundness, observe that the acceptance probability is bounded by

$$(\alpha_{acc}\sqrt{c} + \alpha_{rej}\sqrt{1-c})^2 + \varepsilon \leq 1 - (c - \alpha_{acc}^2)^2 + \varepsilon \leq 1 - \delta^2 + \varepsilon.$$

where ε is the soundness error of the spanning tree construction. ◀

Then we can use the dQIP^{com} variant of Theorem 9 and the parallel repetition of [11] repeatedly to reduce the number of turns to 5.

▶ **Corollary 23.** *Any $\mathsf{dQIP}^{com}[k](f(n))$ protocol that uses $(g(n)+f(n)\deg(u))$-qubit private register at node u can be transformed to a $\mathsf{dQIP}^{com}[5](f(n)\Delta+g(n))$ protocol with perfect completeness where Δ is the maximum degree of the network.*

[5] The malicious prover can make the state e.g., $\alpha_{acc}(\sqrt{\beta}|0^n\rangle + \sqrt{1-\beta}|1^n\rangle)_{\mathsf{O}}|\psi_{acc}\rangle + \alpha_{rej}|1^n\rangle_{\mathsf{O}}|\psi_{rej}\rangle$, but it only leads lower acceptance probability.

$(k+4)$-turn dQIP^{com} protocol with perfect completeness.

1. Run the original protocol except outputting accept or reject. Construct a spanning tree with the root ℓ.
2. Each node u prepares 1-qubit register O_u, perform the CNOT gate on O_u controlled on the first qubit of its private register which corresponds to the output, then sends O_u to the prover.
3. The prover sends the register O_u to each node u, then u rejects if O_u is in $|0\rangle$ and its output qubit is $|1\rangle$. The network checks if all of qubits provided by the prover are the same. If not, the network rejects the protocol.
4. Let ℓ be the leader node which is also the root of a spanning tree. The leader ℓ prepares one-qubit register O' in the state $|0\rangle$ and increments O' iff O_ℓ is $|1\rangle$.
5. Each node u sends the registers V_u, O_u and $\mathsf{W}_{u,v}$ for all $(u,v) \in E$ to the prover.
6. The prover sends O_ℓ to ℓ, and ℓ subtracts O_ℓ from O' (i.e., flips O' iff the content of O_ℓ is $|1\rangle$). The leader ℓ applies T_c to O_ℓ where T_c is given by

$$T_c(|0\rangle) = \sqrt{c}\,|0\rangle - \sqrt{1-c}\,|1\rangle$$
$$T_c(|1\rangle) = \sqrt{1-c}\,|0\rangle + \sqrt{c}\,|1\rangle\,.$$

   Then the leader measures O_ℓ and accepts iff the outcome is $|0\rangle$.

Figure 4 $(k+4)$-turn dQIP^{com} protocol with perfect completeness.

Reconfiguration of Digraph Homomorphisms

Benjamin Lévêque
Laboratoire G-SCOP, Grenoble INP, Université Grenoble-Alpes, France

Moritz Mühlenthaler
Laboratoire G-SCOP, Grenoble INP, Université Grenoble-Alpes, France

Thomas Suzan
Laboratoire G-SCOP, Grenoble INP, Université Grenoble-Alpes, France

Abstract

For a fixed graph H, the H-Recoloring problem asks whether, given two homomorphisms from a graph G to H, one homomorphism can be transformed into the other by changing the image of a single vertex in each step and maintaining a homomorphism to H throughout. The most general algorithmic result for H-Recoloring so far has been proposed by Wrochna in 2014, who introduced a topological approach to obtain a polynomial-time algorithm for any undirected loopless square-free graph H. We show that the topological approach can be used to recover essentially all previous algorithmic results for H-Recoloring and that it is applicable also in the more general setting of digraph homomorphisms. In particular, we show that H-Recoloring admits a polynomial-time algorithm i) if H is a loopless digraph that does not contain a 4-cycle of algebraic girth 0 and ii) if H is a reflexive digraph that contains no triangle of algebraic girth 1 and no 4-cycle of algebraic girth 0.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Digraph Homomorphisms, Combinatorial Reconfiguration

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.43

Related Version *Full Version*: `https://arxiv.org/abs/2205.09210` [14]

1 Introduction

Reconfiguration problems have been introduced formally by Ito et al. in [11] and their complexity has been studied systematically since. Applications can be found in statistical physics, combinatorial games, and uniform sampling of objects such as colorings and matchings. The general setting is the following: Given two feasible solutions of an instance of a combinatorial problem, the goal is to decide whether one can be transformed into the other in a step-by-step manner, visiting only feasible configurations during the transformation. Related questions of interest are whether any two feasible solutions admit a transformation, and if there is a transformation of at most a certain length between two given solutions. We refer the reader to the surveys of Nishimura [15] and van den Heuvel [16] for a discussion of results and applications in this area.

A digraph homomorphism maps the vertex set of a digraph G to the vertex set of a digraph H such that each arc of G is mapped to an arc of H. The classical digraph homomorphism problem $\mathrm{CSP}(H)$ asks whether there is a digraph homomorphism from a given digraph G to a fixed "template" digraph H. Besides the fact that $\mathrm{CSP}(H)$ generalizes graph coloring, one of the motivations for studying its complexity is that it is polynomially equivalent to the seemingly richer constraint satisfaction problem $\mathrm{CSP}(\mathcal{H})$, where the template $\mathcal{H}$ can be any fixed finite relational structure [8]. The complexity of $\mathrm{CSP}(H)$ (and hence $\mathrm{CSP}(\mathcal{H})$) is well understood in the sense that for any digraph H, the problem $\mathrm{CSP}(H)$ is known to be either polynomial-time solvable or NP-complete, a result that has been proved recently by Bulatov [5] and by Zhuk [18], settling in the affirmative a long-standing conjecture by


© Benjamin Lévêque, Moritz Mühlenthaler, and Thomas Suzan;
licensed under Creative Commons License CC-BY 4.0

Feder and Vardi [8]. Motivated by these recent developments we study the complexity of the natural reconfiguration variant H-Recoloring associated with CSP(H), which is the following question: Given two digraph homomorphisms α and β from a G to H, is there a step-by-step transformation between α and β that changes the image of one vertex of G at a time and maintains a digraph homomorphism from G to H throughout?

The complexity of H-Recoloring is known for several special cases, most notably cases when H is from some class of undirected loopless graphs. Despite several positive [4, 6, 13, 17] and negative [1, 4, 12] results, a complete classification of the complexity of H-Recoloring even for undirected graphs H is not known. By replacing each edge of an undirected graph by two directed edges of opposite orientation, we see that digraphs homomorphisms are strictly more general than homomorphisms of undirected graphs in this context. For digraphs H, we are aware of only two results for H-Recoloring, which consider the case where H is a transitive tournament [7] and where H is some orientation of a reflexive digraph cycle [2] (a graph is *reflexive* if it has a loop on each vertex).

The *algebraic girth* of the orientation of a cycle is the absolute value of the number of forward arcs minus the number of backward arcs. In particular, a *4-cycle of algebraic girth 0* is either one of the two graphs shown in Figure 1a. We extend the topological approach developed by Wrochna [17] in the context of undirected graphs to digraphs and obtain a polynomial-time algorithm for CSP(H) for any graph H that contains no 4-cycle of algebraic girth 0 as a subgraph.

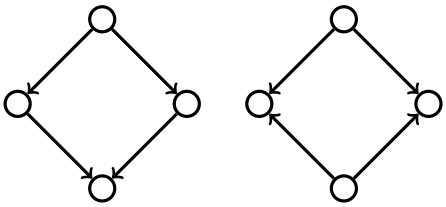
▶ **Theorem 1.** *Let H be a loopless digraph that contains no 4-cycle of algebraic girth 0 as a subgraph. Then H-Recoloring admits a polynomial-time algorithm.*

Theorem 1 generalizes the polynomial-time algorithm for CSP(H) given in [17] for undirected graphs H without a cycle on four vertices as subgraph, which in turn is a generalization of [6], where H is a complete graph on three vertices. The triangle of algebraic girth 1 is shown in Figure 1b. For reflexive digraphs we obtain the following result.

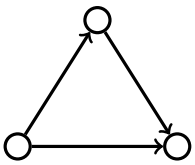
▶ **Theorem 2.** *Let H be a reflexive digraph that contains neither a triangle of algebraic girth 1 nor a 4-cycle of algebraic girth 0 as a subgraph. Then H-Recoloring admits a polynomial-time algorithm.*

Theorem 2 generalizes the algorithmic results from [2], where H is assumed to be a reflexive digraph cycle and from [13], where H is a triangle-free reflexive undirected graph. We remark that the algorithms of theorems 1 and 2 produce certificates for both Yes and No instances and that their running time is polynomial in the size of G and the size of H. The remaining known algorithmic results in [4, 7], while not implied directly by theorems 1 and 2, can be obtained in a straight-forward manner using the topological approach, see sections B.2 and B.3. In the light of our results, to our knowledge, all previous algorithmic results for H-Recoloring can be obtained by the topological approach. Therefore it seems natural to ask whether there are digraphs H such that the topological approach for H-Recoloring does not work, but there is nevertheless a polynomial-time algorithm. This question seems to be a stepping stone to a complete classification of the complexity of H-Recoloring.

One intriguing property of reconfiguration problems is that they can be easy even if the underlying decision problem is hard and vice versa. To illustrate this, consider the classical 3-Coloring problem, which can be formulated as follows: Given an undirected graph G, decide if there is a homomorphism from G to the complete graph on three vertices. While 3-Coloring is NP-complete, Cereceda et al. showed [6] that, given two such homomorphisms ("3-colorings"), there is a polynomial-time algorithm that decides whether there is a step-by-step transformation between them. Theorem 1 generalizes the result of Cereceda et al.



(a) The two non-isomorphic orientations of a 4-cycle of algebraic girth zero.



(b) The orientation of a 3-cycle of algebraic girth 1.

Figure 1 Orientations of a 4-cycle and a 3-cycle that appear in theorems 1 and 2.

and provides new examples with similar behavior. For instance, it is known that there are orientations H of a tree such that CSP(H) is NP-complete, but Theorem 1 implies that for any orientation H of any tree, the problem H-Recoloring admits a polynomial-time algorithm. The situation is different for reflexive graphs: Here, deciding if a given graph G admits a homomorphism to a fixed reflexive graph H is trivial, since a homomorphism may send all vertices of G to the same looped vertex of H. However, deciding if two given homomorphisms to a reflexive graph H admit a step-by-step transformation turns out to be non-trivial even for restricted graphs H (see [3, 13] and the proof of Theorem 2) and is PSPACE-complete in general [17].

1.1 Our results and their relation to Wrochna's algorithm

We show that the topological approach introduced by Wrochna [17] for reconfiguring homomorphisms of undirected graphs can be extended to the digraph homomorphisms. An undirected graph is *square-free* if it does not contain a cycle on four vertices as a subgraph. For a homomorphism $G \to H$ of directed or undirected graphs, we refer to the image of a vertex of G as its *color*. Two graph homomorphisms $\alpha, \beta : G \to H$ admit a step-by-step transformation if there is a sequence $\sigma_1, \sigma_2, \ldots, \sigma_\ell$ of homomorphisms $G \to H$, such that $\alpha = \sigma_1$, $\beta = \sigma_\ell$, and any two consecutive homomorphisms σ_i, σ_{i+1} differ with respect to the color of exactly one vertex. Such a sequence is called *H-recoloring sequence* (from α to β). One key observation of Wrochna [17] is that if H is an undirected square-free graph then, whenever the color of a vertex changes during a step-by-step transformation then all of its neighbors must have the same color[1]. This so-called *monochromatic neighborhood property* sets the stage for a topological approach to the H-Recoloring problem. Here, the graphs are considered to be continuous objects, obtained by gluing edges represented by unit intervals to their respective end-points, and graph homomorphism are continuous maps between the corresponding topological spaces. The monochromatic neighborhood property implies that if there is an H-recoloring sequence between two homomorphisms then they are homotopy-equivalent. This permits to represent H-recoloring sequences satisfying the monochromatic neighborhood property (up to re-ordering) by walks in H that correspond to the color changes of a single vertex of G. Wrochna [17] provides a characterization of all such walks for square-free graphs H, which leads to a polynomial-time algorithm for the corresponding H-Recoloring problem.

[1] It is easy to verify that if a vertex v of an undirected graph G has two neighbors with distinct colors then a color change of v implies that H contains a square.

At first sight, the crucial premise of Wrochna's algorithm seems to be the square-freeness of H. But in fact, the algorithm finds for any undirected graph H, square-free or not, walks that represent all H-recoloring sequences satisfying the monochromatic neighborhood property (possibly there is no such walk). In [17], Wrochna remarks the following.

▶ Remark 3 ([17]). We note that none of the proofs in this paper used any structural properties of H. If we consider H-Recoloring for any graph H, but only allow recoloring a vertex if all of its neighbors have one common color (in other words, a reconfiguration step is allowed only when the homotopy class of the mapping does not change), the same results will follow.

This remark implies immediately that his algorithm also works for undirected graphs H with loops allowed, whenever H does not contain C_4, K_3 with one loop added and K_2 with both loops added.

For a loopless digraph H, a structural property that enforces the monochromatic neighborhood property of any H-recoloring sequence is that H does not contain a 4-cycle of algebraic girth 0 (see Figure 1a). Following the discussion of Wrochna's algorithm above, we may apply it to the corresponding undirected graph $\bar{H}$ and it will return a description of all walks that represent $\bar{H}$-recoloring sequences satisfying the monochromatic neighborhood property. To obtain Theorem 1, it remains to determine whether or not one of the walks corresponds to an H-recoloring sequence that is compatible with the orientation of the arcs of H. For this purpose we introduce the so-called zigzag condition and show that the walks in H that represent H-recoloring sequences satisfying the monochromatic neighborhood property and the zigzag condition are precisely those that are compatible with the orientation of H (Theorem 16). Finally, to prove Theorem 1, we show that it can be checked in polynomial time whether any of the walks found by Wrochna's algorithm satisfies the zigzag condition. Using ideas from [17], a polynomial-time algorithm for finding *shortest* H-recoloring sequences can be obtained (see Section B.1).

In order to prove Theorem 2, the topological approach needs to be adapted to the setting of reflexive graphs. For this purpose, we introduce the *push-or-pull* property, which is similar to the monochromatic neighborhood property in a topological sense. We say that an H-recoloring sequence satisfies the *push-or-pull property* if, whenever a vertex of a graph G changes its color, say from a to b, then all its neighbors in G have either color a or b. This property ensures that if we consider the graphs G and H as topological spaces and homomorphisms $G \to H$ as continuous mappings between them (as described above), then all H-colorings of a given H-recoloring sequence satisfying the push-or-pull property are homotopy-equivalent. See Figure 2 for an example of two homomorphisms to a reflexive triangle that are not homotopy-equivalent: the first wraps around the triangle while the second does not.

It is not hard to see that for any triangle-free reflexive undirected graph H, any H-recoloring sequence satisfies the push-or-pull property. Similar to the loopless case we characterize those walks in H that correspond to H-recoloring sequences satisfying the push-or-pull property (see Theorem 20). From this characterization we obtain a polynomial-time algorithm for H-Recoloring for any undirected reflexive graph H of girth at least 5 (Corollary 24 in [14]). Recently, Lee et al. [13] have obtained the same result using other methods.

An advantage of the topological approach is that it allows for a generalization to digraphs H: we use the characterization of walks in Theorem 20 for the undirected graph $\bar{H}$ (almost) as a black box and then check whether any of the corresponding $\bar{H}$-recoloring sequences is compatible with the orientation of the arcs of H. Depending on which case of

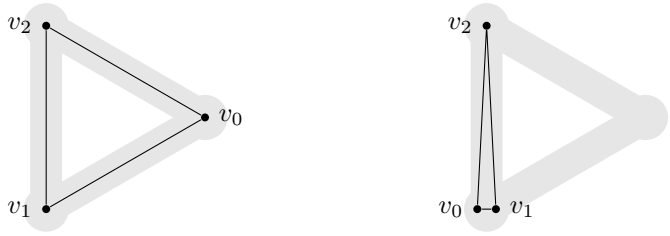


Figure 2 Two homomorphisms $G \to H$ that differ in the color of one vertex but which are not homotopy-equivalent. The graph G (in black) is a triangle and the graph H (in gray) is a triangle with a loop on each vertex.

Theorem 20 applies, different levels of sophistication are required, but in any case there is a polynomial-time algorithm. Notice that the push-or-pull property implies that when a vertex changes its color, say from a to b, then a and b are adjacent in H. Using ideas from [13], we show that we can get rid of this restriction in the case that H contains no 4-cycle of algebraic girth 0 which is the final ingredient of Theorem 2. The last step illustrates the connection between H-Recoloring and the so-called Hom-graph, which is discussed in Section 3 in [14].

1.2 Related work

The complexity of H-Recoloring for undirected graphs H has been studied systematically, in particular since the work of Cereceda et al. [6], who showed that if $H = K_3$, a complete graph on three vertices, then H-Recoloring admits a polynomial-time algorithm, despite CSP(K_3) ("3-Coloring") being NP-complete. Wrochna [17] generalized this result, showing that H-Recoloring admits a polynomial-time algorithm if H is loopless and square-free. Brewster et al. [4] gave a complexity classification of H-Recoloring for circular cliques $C_{p,q}$. Note that their polynomial-time algorithm for $2 \leq p/q < 4$ includes graphs H that are not square-free. Recently, Lee et al. [13] adapted Wrochna's algorithm to the case that H is reflexive and has girth at least 5. On the negative side, it is known that H-Recoloring is PSPACE-complete if H is a clique on at least four vertices [1], a circular clique $C_{p,q}$ where $p/q \geq 4$ [4], a wheel on a k-cycle, where $k \geq 3$ and $k \neq 4$ [12], or a quadrangulation of the 2-sphere with certain properties [12].

We are aware of two results for H-Recoloring for digraphs H. The first one is by Brewster et al. [2], who showed that H-Recoloring admits a polynomial-time algorithm if H is a reflexive digraph cycle that does not contain a 4-cycle of algebraic girth 0. In spirit this algorithm uses the topological approach similar to that of Wrochna (but more involved) that reduces the task of finding H-recoloring sequences to finding vertex walks in H. Secondly, Dochterman and Singh [7] study the Hom-complex for digraphs G and H and show that it is connected (in the topological sense) if H is the transitive tournament T_n on n vertices. From this they conclude that any instance of T_n-Recoloring is a Yes-instance and give a polynomial-time algorithm that finds a T_n-recoloring sequence. The algorithm is simple and does not need any topological tools. It boils down to the fact that a homomorphism of an acyclic digraph into a tournament corresponds to a linear extension of a partial order. To reconfigure one linear extension into another, we may greedily take the last element where the two linear extensions disagree and assign to the vertex with the smaller image the larger image (w.r.t. the total order). The same result can be obtained using the topological approach (see Section B.3).

Further results are known for $\mathcal{H}$-Recoloring for a relational structure $\mathcal{H}$ on a Boolean domain. This problem corresponds to the reconfiguration of satisfying assignments of Boolean formulas. In [9], Gopalan et al. provide a complexity dichotomy for this problem

by characterizing the relations for which Boolean satisfiability reconfiguration admits a polynomial-time algorithm and showing that the problem is PSPACE-complete otherwise. Another popular theme with a different flavor is the reconfiguration of subgraphs, which may be considered to be homomorphisms (injective or not) from a fixed graph H to a given graph G. In this context, Ito et al. showed that reconfiguring directed paths is PSPACE-complete. Often however the graph H is not fixed, but any subgraph of a certain "shape", e.g., any tree, would be acceptable. We refer the readers to the survey of Nishimura [15] for an overview of known results in this direction, in particular those on the reconfiguration of independent sets, as well as [10] for results on spanning and induced subgraphs. A general introduction to reconfiguration problems that also discusses their relation to combinatorial games and puzzles can be found in [16].

1.3 Organization

Section 2 contains notation and basic definitions that are needed to prove Theorems 1 and 2. Section 3 gives an overview of the proof of Theorem 1, which implies a polynomial-time algorithm for H-Recoloring if H is a loopless digraph that contains no 4-cycle of algebraic girth 0. This section also serves as a blueprint for the more involved proof of Theorem 2, which is sketched in Section 4. Due to space limitations the full proofs are deferred to the appendix.

2 Preliminaries

A directed graph (digraph) is a pair $(V(G), A(G))$ where $V(G)$ is a finite set of vertices and $A(G) \subseteq V(G) \times V(G)$ are *arcs*. We write $u \to v$ when $uv \in A(G)$. We say that a digraph G is *symmetric* if $vu \in A(G)$ whenever $uv \in A(G)$. A digraph G is *reflexive* if $uu \in A(G)$ for each vertex $u \in V(G)$. We interpret a symmetric digraph as *undirected graph* and think of two edges $\{uv, vu\}$ as undirected edge, which we also write as uv since it should be clear from the context whether we refer to a directed or undirected edge. We write $E(G)$ for the set of undirected edges of a symmetric graph G. For any digraph G, we canonically associate to G an undirected graph $\bar{G}$ where $V(\bar{G}) = V(G)$ and $uv \in E(\bar{G})$ if $u \to v$ or $v \to u$. Let G be a digraph. The *in-neighborhood* (resp., *out-neighborhood*) of a vertex $v \in V(G)$ is given by $N^-_G(v) := \{w \in V(G) \mid w \to v\}$ (resp., $N^+_G(v) := \{w \in V(G) \mid v \to w\}$). If G is symmetric (undirected), the neighborhood $N_G(v)$ of a vertex $v \in V(G)$ is the set of vertices adjacent to v in G, that is, $N_G(v) := \{w \in V(G) \mid vw \in E(G)\}$. Let G and H be digraphs. A homomorphism $\phi : G \to H$ or (H-coloring of G) is a map $V(G) \to V(H)$ that preserves arcs, that is, for each $u \to v$, we have $\phi(u) \to \phi(v)$. Similarly, for undirected graphs G and H, a homomorphism $\phi\colon G \to H$ is a map $V(G) \to V(H)$ that preserves edges (but not necessarily non-edges). A homomorphism $\alpha\colon G \to H$ also induces a homomorphism $\bar{\alpha}\colon \bar{G} \to \bar{H}$.

A sequence $W = (v_1v_2)(v_2v_3)\dots(v_{n-1}v_n)$ of consecutive edges of an undirected graph G is a *walk*. The reverse walk W^{-1} of a walk $W = (v_1v_2)(v_2v_3)\dots(v_{n-1}v_n)$ is the walk $W^{-1} = (v_nv_{n-1})\dots(v_2\,v_1)$. The length $|W|$ of W the number of edges of W. A *cycle* C is a closed walk, i.e., a walk such that $v_1 = v_n$ on $n-1$ distinct vertices. A walk in digraph G is a walk in $\bar{G}$. The *algebraic girth* of a cycle C in a digraph G is the absolute value of the number of forward arcs minus the number of backward arcs. We say that a graph or a digraph G is *connected* if for any two vertices $u, v \in V(G)$ there is a walk from u to v in G. A walk $W = (v_1v_2)\dots(v_{n-1}v_n)$ in a digraph is *directed* if $v_i \to v_{i+1}$ for all $1 \le i \le n-1$. The walk W is *symmetric* if both W and W^{-1} are directed. We denote the empty walk by ε.

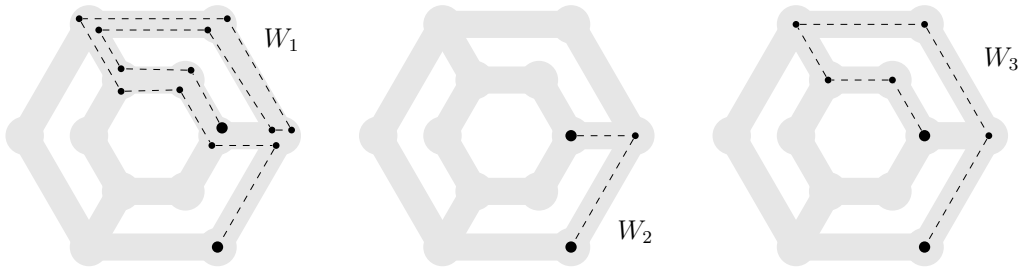


Figure 3 Consider the gray graph H and the three walks W_1, W_2 and W_3 in H (their successive vertices are drawn in black). The walk W_1 reduces to W_2. The walk W_3 is already reduced.

Fundamental groupoid

Let H be an undirected or directed graph. Given a walk $W = (v_1\, v_2)(v_2\, v_3)\dots(v_{n-1}\, v_n)$ in H, we call *reduction* the two following operations (see Figure 3):

- The operation of deleting $(v_i\, v_{i+1})(v_{i+1}\, v_{i+2})$ from W if $v_i = v_{i+2}$ and $1 \le i \le n-2$
- The operation of deleting $(v_i\, v_{i+1})$ from W if $v_i = v_{i+1}$ and $1 \le i \le n-1$. Note that this operation requires a loop on v_i, so it applies in particular if H is reflexive.

We say that W is *reduced* if none of the two operations above is applicable. That is, for $1 \le i \le n-2$, we have $v_{i+2} \neq v_i$ and for $1 \le i \le n-1$, we have $v_{i+1} \neq v_i$. We can *reduce* a walk W by iteratively applying reductions on it; we can easily see that by doing so we obtain a unique reduced walk. By considering two walks to be equivalent if they reduce to the same walk, we obtain an equivalence relation $\sim$ on the walks in H. The fundamental groupoid $\pi(H)$ is the set of all equivalence classes of walks in H under $\sim$. Its groupoid operation is the concatenation $\cdot$ of walks and its neutral element is the empty walk ε. For any walk $W = (v_1\, v_2)\dots(v_{n-1}\, v_n)$, the inverse of (the class of) W in $\pi(H)$ is (the class of) the reversed walk $W^{-1} = (v_n\, v_{n-1})\dots(v_2\, v_1)$ since both WW^{-1} and $W^{-1}W$ reduce to ϵ. In the next sections of this paper, we will write $W_1 = W_2$ *in* $\pi(H)$ if $W_1 \sim W_2$, that is, W_1 and W_1 reduce to the same walk.

Cyclic reduction

We say that closed walk $C = (v_1 v_2)\dots(v_{n-1} v_1)$ is *cyclically reduced* if it is reduced and additionally $v_2 \neq v_{n-1}$. We can *cyclically reduce* any reduced closed walk C by iteratively deleting from it both its first and last edges while one is the inverse of the other. This operation leads to a unique decomposition $C = A^{-1}C_0A$ where A is the sequence of deleted edges of C and C_0 is cyclically reduced.

H-recoloring

Let G and H be digraphs. Recall that two digraph homomorphisms $\alpha, \beta : G \to H$ admit an *H-recoloring sequence* if for some ℓ there are digraph homomorphisms $\sigma_1, \sigma_2, \dots, \sigma_\ell : G \to H$, such that $\alpha = \sigma_1$, $\beta = \sigma_\ell$ and for $1 \le i < \ell$, the two homomorphisms σ_i and σ_{i+1} differ with respect to the color of a single vertex of G. The problem H-Recoloring asks whether, given a graph G and two digraph homomorphisms $\alpha, \beta : G \to H$, the homomorphisms α and β admit an H-recoloring sequence. An H-recoloring sequence satisfies the *monochromatic neighborhood property* if, whenever a vertex v of G changes its color then all neighbors of v in G have the same color.

In this entire paper we assume that G and H are directed or undirected, (weakly) connected graphs, with at least two vertices

One can see that assuming the connectivity of G and H imposes no restrictions, since if G is not connected, we may consider the recoloring of each connected component of G separately. If H is not connected, observe that any connected component of G maps to a connected component of H.

3 Loopless digraphs

In this section we prove Theorem 1. To do so, we extend the polynomial-time algorithm from [17] for H-Recoloring for the case that H is symmetric and square-free to the case where H is any loopless digraph that contains no 4-cycle of algebraic girth 0. We assume in the following that G and H are simple weakly connected digraphs with at least two vertices. Observe that since H is a subgraph of the symmetric graph $\bar{H}$, any H-recoloring sequence is also a $\bar{H}$-recoloring sequence. Furthermore, if H contains no 4-cycle of algebraic girth 0 then any H-recoloring sequence satisfies the monochromatic neighborhood property. To see this, consider a step of any H-recoloring sequence where the color of a vertex $u \in V(G)$ changes from a to b ($a, b \in V(H)$). Let v be any neighbor of u and let h be the current color of v. If a color different from h appears in the neighborhood of u then H contains a cycle of algebraic girth 0, that is, one of the two orientations of the 4-cycle shown in Figure 1a. To prove Theorem 1, the main idea is to run Wrochna's algorithm on the symmetric graphs $\bar{G}$ and $\bar{H}$ to obtain a description of those $\bar{H}$-recoloring sequences that satisfy the monochromatic neighborhood property. We then check whether one of these sequences is compatible with the orientation of the edges of H.

3.1 Realizable walks in $\bar{H}$

We recall several results and definitions from [17] that will be useful later on, stating them in a slightly different manner for the sake of better integration in the context of digraph homomorphisms. Largely the same proofs apply however, as pointed out in Remark 3.

Let $S = \sigma_0, \ldots, \sigma_\ell$ be a $\bar{H}$-recoloring sequence satisfying the monochromatic neighborhood property and let v be a vertex of G. For each $0 \leq i < \ell$, let $S_i(v)$ be given by

$$S_i(v) = \begin{cases} \epsilon & \text{if } \sigma_i(v) = \sigma_{i+1}(v), \text{ and} \\ (\sigma_i(v)\, h)(h\, \sigma_{i+1}(v)) & \text{otherwise,} \end{cases}$$

where h is the unique color of the neighbors of v with respect to $\sigma_i(v)$ and $\sigma_{i+1}(v)$. We associate with S and v the walk $S(v) := S_0(v)S_1(v)\cdots S_\ell(v)$ in H. Suppose that $S(v) = (a_1\, a_2)(a_2\, a_3)\cdots(a_{n-2}\, a_{n-1})(a_{n-1}\, a_n)$. Then according to S the vertex v changes its color from a_1 to a_3 while its neighbors have color a_2, then it changes color from a_3 to a_5 while its neighbors all have color a_4 (so all the neighbors must change their color from a_2 to a_4 before), and so on until v changes its color from a_{n-2} to a_n while its neighbors all have color a_{n-1}.

Let us pick any vertex $q \in V(G)$ and let Q be a reduced walk from $\alpha(q)$ to $\beta(q)$. Furthermore, let $(W_v)_{v \in V(G)}$ be a set of walks from q to each $v \in V(G)$. We say that Q and $(W_v)_v$ *generate* the walks $S_v := \alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v) \in \pi(H)$. We will say that Q is $\bar{H}$*-realizable* for α, β, q if there is a $\bar{H}$-recoloring sequence S satisfying the monochromatic neighborhood property such that $Q = S(q)$. Hence, by Lemma 4, if Q is $\bar{H}$-realizable then S_v

does not depend on W_v. We say that Q is *H-realizable* for α, β, q if there is a H-recoloring sequence S satisfying the monochromatic neighborhood property such that $Q = S(q)$. Clearly, if Q is H-realizable then it is $\bar{H}$-realizable[2].

According to Remark 3, we obtain the next results by following the proofs in [17] nearly word by word.

▶ **Lemma 4** ((∗) see [17, Lemma 4.1]). *Let $S = \sigma_0, \ldots, \sigma_\ell$ be an $\bar{H}$-recoloring sequence from $\alpha = \sigma_0$ to $\beta = \sigma_\ell$ satisfying the monochromatic neighborhood property and let W be any walk in G connecting two vertices u and v. Then $S(v) = \alpha(W)^{-1} \cdot S(u) \cdot \beta(W)$ in $\pi(H)$. In particular, for any set $(W_v)_{v \in V(G)}$ of walks from q to all vertices $v \in V(G)$, if a walk Q from $\alpha(q)$ to $\beta(q)$ is $\bar{H}$-realizable then the other vertex walks in any associated $\bar{H}$-recoloring sequence are the walks $\alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v) \in \pi(H)$.*

The next theorem gives a description of all $\bar{H}$-realizable walks.

▶ **Theorem 5** (see [17, Theorem 8.1]). *Let $\alpha, \beta \colon G \to H$ and $q \in V(G)$. Let $\bar{\Pi}$ be the set of all reduced walks that are $\bar{H}$-realizable for α, β, q. Then one of the following holds:*

1. *$\bar{\Pi} = \emptyset$.*
2. *$\bar{\Pi} = \{Q\}$ for some $Q \in \pi(H)$.*
3. *$\bar{\Pi} = \{R^n P \mid n \in \mathbb{Z}\}$, for some $R, P \in \pi(H)$.*
4. *$\bar{\Pi}$ contains all reduced walks of even length from $\alpha(q)$ to $\beta(q)$.*

Furthermore, there is an algorithm that determines in time $O(|V(G)| \cdot |E(G)| + |E(H)|)$ which case holds and outputs Q or R, P in cases 2 3 such that $|Q|, |R|, |P|$ are bounded by the total running time $O(|V(G)| \cdot |E(G)| + |E(H)|)$.

The following lemma tells us that from any $\bar{H}$-realizable walk we can obtain in polynomial time an $\bar{H}$-recoloring sequence. We will later generalize it to digraphs, which allows us to construct in polynomial time an H-recoloring sequence from a given H-realizable walk (see Lemma 8).

▶ **Lemma 6** (see [17, Theorem 6.1]). *Given an $\bar{H}$-realizable walk Q we can construct an associated $\bar{H}$-recoloring sequence in time $O(|V(G)|^2 + |V(G)| \cdot |Q|)$.*

3.2 Orientation compatibility

We now characterize $\bar{H}$-recoloring sequences that both satisfy the monochromatic neighborhood property and are compatible with the orientation of H. To this end we give a simple condition, the *zigzag condition*, on the reduced walks of the vertices of G obtained from a given $\bar{H}$-recoloring sequence. We show that it suffices to check the zigzag condition for each vertex of G in order to determine whether a given $\bar{H}$-recoloring sequence is also an H-recoloring sequence. From this we obtain a polynomial-time algorithm that, given two H-colorings $\alpha, \beta : G \to H$ and a vertex v of G, finds a walk from $\alpha(v)$ to $\beta(v)$ in H that is compatible with the orientation of H or reports correctly that no such walk exists. For the remainder of this section let us fix two digraph homomorphisms $\alpha, \beta \colon G \to H$.

[2] This definition generalizes the definition of realizability from [17] such that $\bar{H}$-realizability becomes a necessary condition for H-realizability. When $\bar{H}$ is square-free our definition coincides with the one from [17].

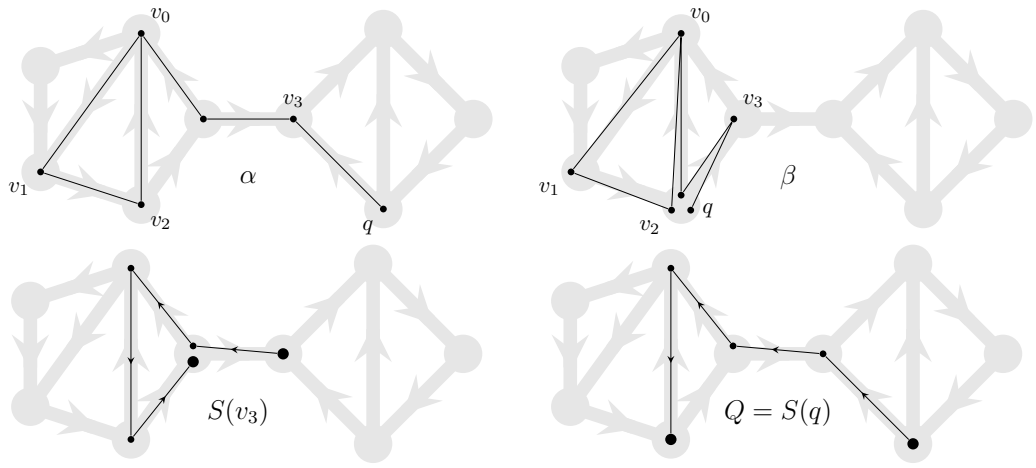


Figure 4 Homomorphisms α and β map a graph G on 6 vertices to the gray graph H. The walk $Q = S(q)$ satisfies the zigzag condition and generates walks that also satisfy the zigzag condition like $S(v_3)$, so it is orientation compatible. We will see soon that the vertices of the triangle $v_0\, v_1\, v_2$ must have symmetric vertex walks, so they cannot change colors for the lack of a symmetric edge in H.

3.2.1 The zigzag condition

Let $v \in V(G)$ and $S_v = (a_1a_2)\dots(a_{n-1}a_n)$ be a walk of even length from $\alpha(v)$ to $\beta(v)$ (think of S_v as the walk of v induced by some $\bar{H}$-recoloring sequence S that satisfies the monochromatic neighborhood property; in particular, S and S_v could have been obtained by combining Theorem 5 and Lemma 6). We say that S_v satisfies the *zigzag condition* if $a_1 \leftarrow a_2 \rightarrow a_3 \leftarrow \dots \leftarrow a_{n-1} \rightarrow a_n$ is a walk in H whenever $N_G^-(v) \neq \emptyset$ (then we also say that S_v zigzags correctly with respect to its *in*-neighborhood) and $a_1 \rightarrow a_2 \leftarrow a_3 \rightarrow \dots \rightarrow a_{n-1} \leftarrow a_n$ is a walk in H whenever $N_G^+(v) \neq \emptyset$. Notice that at least one of $N_G^-(v)$ and $N_G^+(v)$ is non-empty by our assumptions on G. Also, observe that if S_v is not reduced and satisfies the zigzag condition then it will still satisfy that condition after reduction.

Based on Lemma 4 we can state a definition of orientation compatibility for walks: given $\alpha, \beta\colon G \to H$, $q \in V(G)$ and a set of walks (W_v) from q to v. We say that a reduced walk of even length $Q \in \pi(H)$ from $\alpha(q)$ to $\beta(q)$ is *orientation compatible* for the set $(W_v)_{v\in V(G)}$ if for each vertex $v \in V(G)$ and any walk W_v from q to v, the walk $\alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v) \in \pi(H)$ satisfies the zigzag condition. See Figure 4 for illustrations of the zigzag condition and of orientation compatibility. Observe that by Lemma 4, if Q is $\bar{H}$-realizable, then the orientation compatibility of Q does not depend of the choice of the walks $(W_v)_{v\in V(G)}$.

▶ **Lemma 7.** *Let $q \in V(G)$. Let Q be an $\bar{H}$-realizable walk for α, β, q. Then Q is H-realizable if and only if Q is orientation compatible.*

Proof. Let Q be an $\bar{H}$-realizable walk. First assume that Q is H-realizable. Then let S be a H-recoloring sequence from α to β such that $Q = S(q)$. Let $v \in V(G)$ and W_v a walk from q to v in G. Then $\alpha(W_v)^{-1}Q\beta(W_v) = S(v)$ and we can write $S(v)$ as a reduced walk $(a_1a_2)(a_2a_3)\dots(a_{n-1}a_n)$. If $N_G^-(v) \neq \emptyset$ then there is an arc $w \rightarrow v$ for some $w \in V(G)$. At each color change of v from a_i to a_{i+2}, the vertex w must have color a_{i+1} because S satisfies the monochromatic neighborhood property. Since S is a H-recoloring sequence, its homomorphisms all preserve the arc $w \rightarrow v$. Hence $a_1 \leftarrow a_2 \rightarrow \dots \leftarrow a_{n-1} \rightarrow a_n$ is a path in H. Similarly, if $N_G^+(v) \neq \emptyset$ then we have the same path with all arcs reversed. As this holds for each vertex $v \in V(G)$, Q is orientation compatible.

Conversely, if Q is orientation compatible, then let S be the $\bar{H}$-recoloring sequence constructed via Lemma 6. Consider any step σ_i, σ_{i+1} of S, say when a vertex v changes color from a to b while its neighbors have color c. This color change is recorded in $S(u)$ which satisfies the zigzag condition so if σ_i induces a homomorphism $G \to H$, then σ_{i+1} too. By induction, each homomorphism of S induces a homomorphism $G \to H$ so we eventually have a H-recoloring sequence. ◀

The proof of Lemma 7 implies the following generalization of Lemma 6.

▶ **Lemma 8** (∗). *Given an H-realizable walk Q, we can construct an associated H-recoloring sequence in time $O(|V(G)|^2 + |V(G)| \cdot |Q|)$.*

Furthermore, the zigzag condition can be exploited in order to decide efficiently whether a given walk from $\alpha(q)$ to $\beta(q)$ is H-realizable. To show this, we need the following lemma, which is contained in Wrochna's proof of Lemma 6.

▶ **Lemma 9** ([17]). *Given any walk Q from $\alpha(q)$ to $\beta(q)$, we can decide in time $O(|E(G)| \cdot (|Q| + |V(G)|))$ if Q is $\bar{H}$-realizable.*

▶ **Lemma 10** (∗). *There is a polynomial-time algorithm that, given a walk Q from $\alpha(q)$ to $\beta(q)$, decides in time $O(|E(G)| \cdot (|Q| + |V(G)|))$ if Q is H-realizable for α, β, q.*

3.2.2 Characterizing orientation-compatible walks

The following three technical observations (see Definition 11) on the zigzag condition help us to find all orientation compatible walks (for any set of walks $(W_v)_{v \in V(G)}$). More precisely:

1. For any arc $u \to v$, if S_u satisfies the zigzag condition then S_v satisfies the part of the zigzag condition for the *in*-neighborhood (See Lemma 12).
2. It turns out that a walk Q from $\alpha(q)$ to $\beta(q)$ is H-realizable if and only if it satisfies the zigzag condition and generates symmetric walks on a certain set of vertices of G (See Lemma 13).
3. By checking possible reductions for the walks on that set of vertices (see Lemma 15), we obtain the lemma, which provides a description of all orientation compatible walks.

▶ **Definition 11.** *Let Q be an even walk from $\alpha(q)$ to $\beta(q)$. Furthermore, let $v \in V(G)$ and let W_v be a walk from q to v. Suppose that $S_v := \alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v) = (a_1\, a_2) \dots (a_{n-1}\, a_n)$ is the reduced walk generated on v by Q and W_v. We say that v is of type* IN *if $N_G^-(v) \neq \emptyset$ and it is of type* OUT *if $N_G^+(v) \neq \emptyset$. Furthermore, we say that S_v is* IN-compatible *(resp.,* OUT-compatible*) if v is of type* IN *and additionally $a_1 \leftarrow a_2 \to a_3 \leftarrow \dots \leftarrow a_{n-1} \to a_n$ is a path in H (resp., v is of type* OUT *and additionally $a_1 \to a_2 \leftarrow a_3 \to \dots \to a_{n-1} \leftarrow a_n$ is a path of H). Finally, if v is of type* IN *and of type* OUT*, we say that v is of type* SYM *and that S_v is* SYM*-compatible if it is* IN*-compatible and* OUT*-compatible, this means in particular that S_v has only symmetric edges.*

By Lemma 7, we have that an even walk Q is orientation compatible for the set $(W_v)_{v \in V(G)}$ if and only if for every vertex v, if v is of type IN, then S_v is IN-compatible and if v is of type OUT, then S_v is OUT-compatible.

▶ **Lemma 12.** *For any arc $u \to v$ of G, the walk S_u is* OUT*-compatible if and only if S_v is* IN*-compatible.*

Proof. Note that u is of type OUT and v is of type IN. By the monochromatic neighborhood property, if S_u is OUT-compatible then since α and β are homomorphisms, $(\alpha(v)\,\alpha(u))S_u(\beta(u)\,\beta(v))$, is IN-compatible. As S_v is precisely this walk after reduction, it is then IN-compatible. Similarly, if S_v is IN-compatible, then S_u is OUT-compatible. ◀

▶ **Lemma 13** (∗).

1. *If G has no vertex of type SYM and Q satisfies the zigzag condition then S_v satisfies the zigzag condition for all $v \in V(G)$.*
2. *Let $\{v_1, \ldots, v_k\} \subseteq V(G)$ be the subset of vertices of G of type SYM. If S_{v_i} satisfies the zigzag condition for $1 \leq i \leq k$ then S_v satisfies the zigzag condition for all $v \in V(G)$.*

▶ **Lemma 14** (∗). *Let (W_v) be a set of walks from q_0 to all vertices $v \in V(G)$. There is some vertex $q \in V(G)$ such that the set of orientation compatible walks for q and the set $(W_q^{-1}W_v)_{v\in V(G)}$ is one of the followings:*

1. *$\emptyset$.*
2. *$\{Q\}$ for some reduced walk Q of even length $|Q| = O(|V(G)|)$.*
3. *The set of all reduced walks of even length from $\alpha(q)$ to $\beta(q)$ that satisfy the zigzag condition.*

Furthermore, we can determine in time $O(|V(G)| \cdot |E(G)|)$ which case holds, and output Q in Case 2.

▶ **Lemma 15.** *Let $V' \subset V(G)$ be any nonempty set of vertices of G. Let $q \in V'$ and for each $v \in V'$, let W_v be a walk from q to v of length $|W_v| \leq |V(G)|$. Then the set of walks from $\alpha(q)$ to $\beta(q)$ that generate symmetric vertex walks with the set $(W_v)_{v\in V'}$ on V' is one of the following:*

1. *$\emptyset$.*
2. *$\{Q\}$ for some symmetric walk Q of length at most $2|V(G)|$.*
3. *All symmetric walks from $\alpha(q)$ to $\beta(q)$.*

Furthermore, we can decide in time $O(|V(G)| \cdot |E(G)|)$ which case holds and output Q in Case 2.

Proof. Assume that Q and $(W_v)_{v\in V(G)}$ generate symmetric vertex walks. Then, in particular, Q is symmetric. Note that $\alpha(W_v)$ and $\beta(W_v)$ may have non symmetric edges, but since Q generates only symmetric walks and S_v is the reduced walk obtained from $\alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v)$, we have that the non-symmetric edges of $\alpha(W_v)$ and of $\beta(W_v)$ must be the same and appear in the same order. Suppose that $e_1, e_2, \ldots, e_p$ are the non-symmetric edges of $\alpha(W_v)$ and $\beta(W_v)$. Then we can write $\alpha(W_v) = A_1e_1A_2 \ldots A_pe_pA_{p+1}$ and $\beta(W_v) = B_1e_1B_2 \ldots B_pe_pB_{p+1}$, where $A_1, \ldots, A_{p+1}$ and $B_1, \ldots, B_{p+1}$ are symmetric. Note that if $\alpha(W_v)$ and $\beta(W_v)$ are symmetric then $\alpha(W_v) = A_1$ and $\beta(W_v) = B_1$. Since all non-symmetric edges cancel in S_v, we obtain

$$A_p^{-1} \cdots A_2^{-1}e_1A_1^{-1}QB_1e_1B_2 \cdots B_p = \varepsilon \ .$$

So in particular, $Q = A_1B_1^{-1}$ (so $|Q| \leq 2|V(G)|$) and $A_p^{-1} \ldots A_2^{-1}B_2 \ldots B_p = \varepsilon$.

It remains to show that there is an algorithm that decides in time $O(|V(G)| \cdot |E(G)|)$ which case holds and outputs Q in Case 2. The algorithm repeats the following for each vertex $v \in V'$:

- Compute $\alpha(W_v)$ and $\beta(W_v)$ and search for non symmetric edges. If all edges in $\alpha(W_v)$ and $\beta(W_v)$ are symmetric, continue with the next vertex. Else do the next steps.

- Check that $\alpha(W_v)$ and $\beta(W_v)$ have the same non-symmetric edges appearing in the same order. If not, there is no symmetric walk Q that generates a symmetric walk S_v and we may conclude that Case 1 holds. Otherwise, let e_1 and e_2 be respectively the first and the last non-symmetric edge of $\alpha(W_v)$ and $\beta(W_v)$. Decompose $\alpha(W_v) = A_1e_1A_2e_2A_3$ and $\beta(W_v) = B_1e_1B_2e_2B_3$.
- If in some previous iteration a reduced walk Q has been fixed, check that $Q = A_1B_1^{-1}$. If not, then report Case 1. If no walk Q has been fixed in an earlier iteration, fix Q to be the reduced walk $A_1B_1^{-1} \in \pi(H)$.
- Finally, check that $A_p^{-1} \dots A_2^{-1}B_2 \dots B_p = \varepsilon$. If this equality doesn't hold, report Case 1.

Then, report case 2 and Q if some walk Q has been fixed. Otherwise, report case 3.

Each step runs in time $O(|E(G)|)$ and will repeat at most $|V(G)|$ times, so this algorithm runs in time $O(|V(G)| \cdot |E(G)|)$. ◀

We are now able to obtain a description of all H-realizable walks.

▶ **Theorem 16.** *Let $\alpha, \beta \colon G \to H$. We can find in time $O(|V(G)|)$ a vertex $q \in V(G)$ such that the set Π_q of H-realizable walks from $\alpha(q)$ to $\beta(q)$ is one of the following:*

1. $\Pi_q = \emptyset$.
2. $\Pi_q = \{Q\}$ *for some* $Q \in \pi(H)$.
3. $\Pi_q = \{R^nP \mid n \in \mathbb{Z}\}$ *for some* $R, P \in \pi(H)$.
4. Π_q *contains all reduced walks of even length from $\alpha(q)$ to $\beta(q)$ that satisfy the zigzag condition.*

Moreover, we can determine in time $O(|V(G)| \cdot |E(G)| + |E(H)|)$ which case holds and output Q or R, P in cases 2 or 3. In case 4, we can find such a walk in time $O(|E(G)|)$ (or certify there is none).

Theorem 16 will be proved in the next subsection. Combining Lemma 8 and Lemma 10 with Theorem 16, we immediately obtain Theorem 1.

3.2.3 Proof of Theorem 16

In order to prove Theorem 16 we need the following technical lemma.

▶ **Lemma 17.** *Let R_0 be cyclically reduced walk and let P be a reduced walk starting at the base point of R_0. We can find in time $O(|R_0| + |P|)$ an integer n_0 such that none of R_0 and R_0^{-1} entirely cancels with $R^{n_0}P$.*

Recall that for $\alpha, \beta \colon G \to H$ and $q \in V(G)$, we denote by Π_q the set of all walks from $\alpha(q)$ to $\beta(q)$ that are H-realizable. We are now ready to prove Theorem 16.

Proof of Theorem 16. Fix any $q_0 \in V(G)$ and use breadth first search to compute shortest walks W_v from q_0 to v for all $v \in V(G)$ in time $O(|V(G)| \cdot |E(G)|)$. We invoke Lemma 14 for q_0 and $(W_v)_{v \in V}$ to obtain in time $O(|V(G)| \cdot |E(G)|)$ a vertex $q \in V(G)$ and a description of the set of orientation compatible walks for q and $(W_v)_{v \in V(G)}$. We distinguish the possible outcomes:

Case 1 There is no orientation-compatible walk. Then report that there is no H-realizable walk.

Case 2 There is a unique orientation-compatible reduced walk Q of even length. By Lemma 10, we can decide in time $O(|E(G)| \cdot (|Q| + |V(G)|) = O(|V(G)| \cdot |E(G)|)$ if $|Q|$ is H-realizable as $|Q| = O(|V(G)|)$.

Case 3 All reduced walks of even length from $\alpha(q)$ to $\beta(q)$ that satisfy the zigzag condition and are orientation compatible. Invoke Theorem 5 to get in time $O(|V(G)| \cdot |E(G)| + |E(H)|)$ a description of the set $\bar{\Pi}_q$ of all $\bar{H}$-realizable walks from $\alpha(q)$ to $\beta(q)$. We again distinguish the four possible outcomes.

1. $\bar{\Pi}_q = \emptyset$. There is no $\bar{H}$-realizable walk for α, β, q, so there is no H-realizable walk.
2. $\bar{\Pi}_q = \{Q\}$. There is a unique reduced walk Q that is $\bar{H}$-realizable. Then we can check in time $O(|Q|) = O(|E(G)| \cdot |V(G)| + |E(H)|)$ whether it satisfies the zigzag condition and hence is orientation compatible.
3. $\bar{\Pi}_q = \{R^n P \mid n \in \mathbb{Z}\}$ with $R, P \in \pi(H)$ and R closed and of even length. If R does not satisfy the zigzag condition then the following claim allows us to conclude.

   ▷ Claim 1. Suppose that R does not satisfy the zigzag condition. Then at most one of the $\bar{H}$-realizable walks $R^n P$, $n \in \mathbb{Z}$, satisfies the zigzag condition. Furthermore, we can find such a walk in time $O(|V(G)| \cdot |E(G)| + |E(H)|)$ or conclude there is none.

   Proof. Decompose $R = AR_0A^{-1}$ with all walks minimal and R_0 cyclically reduced. Apply Lemma 17 with the walks R_0 and $A^{-1}P$ and obtain in time $O(|R_0| + |P|) = O(|V(G)| \cdot |E(G)| + |E(H)|)$ an integer $n_0 \in \mathbb{Z}$ such that none of R_0^{-1} and R_0 entirely cancels with $R_0^{n_0} A^{-1} P$.
   If A does not satisfy the zigzag condition, then none of $R^n P$ do for $n \neq n_0$, so only $R^{n_0} P$ can possibly satisfy the zigzag condition, which can be checked in time $O(|R^{n_0} P|) = O(|V(G)| \cdot |E(G)| + |E(H)|)$.
   Otherwise, if A satisfies the zigzag condition, then so do the edges of A^{-1}, so R_0 does not satisfy the zigzag condition (since R does not). For $n > n_0 + 1$, $R^n P \in \pi(H)$ contains an entire R_0 that does not reduce, so it cannot satisfy the zigzag condition. Similarly if $n < n_0 - 1$, then $R^n P$ does not satisfy the zigzag condition since it contains an entire R_0^{-1}. Eventually we only need to test $R^{n_0-1}P$, $R^{n_0}P$ and $R^{n_0+1}P$, which can be done in time $O(|P|) = O(|V(G)| \cdot |E(G)| + |E(H)|)$. Observe that each edge of R_0 belongs to precisely two of those three walks, so as it is the case for the edges of R_0 that do not fit the zigzag condition, we deduce that at most one of $R^{n_0-1}P$, $R^{n_0}P$ and $R^{n_0+1}P$ satisfies the zigzag condition. ◁

   On the other hand, if R satisfies the zigzag condition then P satisfies the zigzag condition if and only all walks $R^n P$ do. To see this, notice that "badly oriented" edges of P must reduce with "badly oriented" edges of R^n, but there is none in R^n since it is orientation-compatible. So we can again distinguish between Case 1 and Case 3 in time $O(|E(G)| \cdot |V(G)| + |E(H)|)$ and report the result.
4. $\bar{\Pi}_q$ contains all reduced walks of even length from $\alpha(q)$ to $\beta(q)$. We report Case 4. Using breadth first search in the tensor product $G \times K_2$ (we construct the graph $G \times K_2$ by creating for each vertex v_i of G two copies u_i and w_i; two vertices u_i and w_j of $G \times K_2$ are adjacent if and only if v_i and v_j are adjacent in G), we can find such a walk in time $O(|E(G)|)$ or conclude there is none. ◀

4 Reflexive graphs

In this section we sketch the proof of Theorem 2. First, we assume that the graphs G and H are undirected and reflexive. The following property can be thought of as an analogue of the monochromatic neighborhood property for reflexive graphs.

▶ **Definition 18.** *Let $\alpha, \beta : G \to H$ and let S be a H-recoloring sequence from α to β. Then S has the* push-or-pull *property if whenever a vertex $u \in V(G)$ changes its color from a to b then any neighbor $v \in V(G)$ of u has color a or b.*

Intuitively, whenever a vertex u of G changes its color, for any neighbor v of u, we have that u is either "pushed" away from the color of v or "pulled" towards the color v. Notice that if H is triangle-free then any H-recoloring sequence satisfies the push-or-pull property: if a neighbor v of u has a color different from a and b then H contains a triangle.

Let $\alpha, \beta : G \to H$ and let S be an H-recoloring sequence from α to β such that S satisfies the push-or-pull property. We associate to each vertex $u \in V(G)$ the vertex walk $S(u)$ in H corresponding to the successive colors of u according to S. Given a vertex q and a walk Q from $\alpha(q)$ to $\beta(q)$, we say that Q is *H-realizable* for α, β, q if there is an H-recoloring sequence S from α to β such that $Q = S(q)$ and S satisfies the push-or-pull property. By the next lemma, we have that for any vertex $v \in V(G)$, the corresponding walk $S(v)$ generates the walk $S(u)$ of any other vertex by conjugation. Notice that for irreflexive graphs H, the same holds for H-recoloring sequences that satisfy the monochromatic neighborhood property ([17, Lemma 4.1], Lemma 4).

▶ **Lemma 19** (∗). *Let S be a H-recoloring sequence from α to β satisfying the push-or-pull property. Then for any $u, v \in V(G)$ and any u-v walk W, we have $S(v) = \alpha(W)^{-1} S(u) \beta(W)$ in $\pi(H)$.*

Lemma 19 allows us to characterize H-realizable walks based on an algorithm that finds vertices of G whose color cannot change. We end up with the following result which immediately implies that if H is reflexive and triangle-free then H-Recoloring admits a polynomial-time algorithm on reflexive instances, under the condition that if the color of a vertex changes then the old and new colors are neighbors in H (see [13, Theorem 1.1]).

▶ **Theorem 20** (∗). *Let G and H be reflexive undirected graphs and let $\alpha, \beta \colon G \to H$ and $q \in V(G)$. Let $\bar{\Pi}$ be the set of all walks that are H-realizable for α, β, q (in particular, the corresponding H-recoloring sequences satisfy the push-or-pull property). Then one of the following holds:*

1. *$\bar{\Pi} = \emptyset$.*
2. *$\bar{\Pi} = \{Q\}$ for some $Q \in \pi(H)$.*
3. *$\bar{\Pi} = \{R^n P \mid n \in \mathbb{Z}\}$, for some $R, P \in \pi(H)$.*
4. *$\bar{\Pi}$ contains all reduced walks from $\alpha(q)$ to $\beta(q)$.*

Furthermore, we can determine in time $O(|V(G)| \cdot |E(G)| + |E(H)|)$ which case holds and output Q or R, P in cases 2 and 3 such that $|Q|, |R|, |P|$ are bounded by the total running time $O(|V(G)| \cdot |E(G)| + |E(H)|)$. Case 4 happens when $\alpha(C) = \beta(C) = \varepsilon$ in $\pi(H)$ for all closed walks C in G.

In order to obtain Theorem 2, we run the algorithm of Theorem 20 on $\bar{H}$ and check for each of the four cases whether there is a corresponding $\bar{H}$-recoloring sequence that is compatible with the orientation of the arcs of H. In Case 1 there is no H-recoloring sequence since there is no $\bar{H}$-recoloring sequence. To deal with Case 2, we use a greedy-type algorithm ("move-forward algorithm") that first constructs from the walk Q the vertex walks of all other vertices of G using Lemma 19 and then either finds a H-recoloring sequence moving vertices to their next color step-by-step or detects an obstruction in the form of a cyclic dependency of color changes. We essentially reduce Case 3 to Case 2, by showing that it suffices to check the H-realizability of the walks $\{R^n P \mid n \in \mathbb{Z}\}$ only for a polynomial number of values of n. For each value n of interest, we run the move-forward algorithm for the walk $R^n P$. In Case 4,

we first compute the set V' of vertices of G that belong to some directed closed walk. We show that the H-realizable walks are exactly those which generate symmetric walks on V'. We use a characterization of such walks in order to conclude whether there is a H-recoloring sequence satisfying the push-or-pull property or not.

Notice that Theorem 20 requires both graphs G and H to be reflexive. The final step in the proof of Theorem 2 is to observe that if H contains no 4-cycle of algebraic girth 0 then we can remove this requirement for the graph G.

5 Conclusion

We showed that H-Recoloring admits a polynomial-time algorithm whenever i) H is a loopless digraph without a 4-cycle of algebraic girth 0 and ii) H is a reflexive digraph containing neither a triangle of algebraic girth 1 nor a 4-cycle of algebraic girth 0. For this purpose we make use of the topological approach developed by Wrochna [17]. Additionally, we showed that all known polynomial-time algorithms for H-Recoloring can be obtained using this approach. That is, in all cases, whenever there is a H-recoloring sequence then in particular all H-colorings of in such a sequence are homotopy-equivalent. This leads to the interesting question, whether homotopy-equivalence is *exactly* the condition that makes H-Recoloring tractable.

References

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

2 Richard C Brewster, Jae-Baek Lee, and Mark H Siggers. Recoloring reflexive digraphs. *Discrete Mathematics*, 341(6):1708–1721, 2018.

3 Richard C Brewster, Jae-Baek Lee, and Mark H Siggers. Reconfiguration of homomorphisms to reflexive digraph cycles. *Discrete Mathematics*, 344(8):112441, 2021. doi:10.1016/j.disc.2021.112441.

4 Richard C Brewster, Sean McGuinness, Benjamin Moore, and Jonathan A Noel. A dichotomy theorem for circular colouring reconfiguration. *Theor. Comput. Sci.*, 639:1–13, 2016. doi:10.1016/j.tcs.2016.05.015.

5 Andrei A Bulatov. A dichotomy theorem for nonuniform CSPs. In *IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 319–330. IEEE, 2017.

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

7 Anton Dochtermann and Anurag Singh. Homomorphism complexes, reconfiguration, and homotopy for directed graphs, 2021. doi:10.48550/arXiv.2108.10948.

8 Tomás Feder and Moshe Y Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998. doi:10.1137/S0097539794266766.

9 Parikshit Gopalan, Phokion G Kolaitis, Elitza N Maneva, and Christos H Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6):2330–2355, 2009. doi:10.1137/07070440X.

10 Tesshu Hanaka, Takehiro Ito, Haruka Mizuta, Benjamin Moore, Naomi Nishimura, Vijay Subramanya, Akira Suzuki, and Krishna Vaidyanathan. Reconfiguring spanning and induced subgraphs. *Theor. Comput. Sci.*, 806:553–566, 2020. doi:10.1016/j.tcs.2019.09.018.

11 Takehiro Ito, Erik D Demaine, Nicholas JA Harvey, Christos H Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

12 Jae-Baek Lee, Jonathan A Noel, and Mark H Siggers. Reconfiguring graph homomorphisms on the sphere. *Eur. J. Comb.*, 86:103086, 2020. doi:10.1016/j.ejc.2020.103086.

13 Jae-Baek Lee, Jonathan A Noel, and Mark H Siggers. Recolouring homomorphisms to triangle-free reflexive graphs. *Journal of Algebraic Combinatorics*, pages 1–21, 2022.

14 Benjamin Lévêque, Moritz Mühlenthaler, and Thomas Suzan. Reconfiguration of digraph homomorphisms. *CoRR*, abs/2205.09210, 2022. doi:10.48550/arXiv.2205.09210.

15 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

16 Jan van den Heuvel. The complexity of change. In Simon R Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409. London Mathematical Society Lectures Note Series, 2013.

17 Marcin Wrochna. Homomorphism reconfiguration via homotopy. *SIAM J. Discret. Math.*, 34(1):328–350, 2020. doi:10.1137/17M1122578.

18 Dmitriy Zhuk. A proof of the CSP dichotomy conjecture. *Journal of the ACM*, 67(5):1–78, 2020.

A Proofs omitted from Section 3

Proof of Lemma 4. We use induction on the length ℓ of S. Let $\ell = 1$ and suppose $\sigma_0 \neq \sigma_1$, so a vertex $w \in V(G)$ is recolored from $\sigma_0(w) = a$ to $\sigma_1(w) = b$ and all neighbors of w have color h. By definition, we have $S(w) = (a\,h)(h\,b)$ and $S(v) = \varepsilon$ for $v \in V(G) \setminus \{w\}$. If $W = \varepsilon$ then $\sigma_0(W) = \sigma_1(W) = \varepsilon$ and $S(u) = S(v)$ since $u = v$, so we are done. If W has length one then, without loss of generality, $W = u \to v$. We consider three cases:

- $u \neq w$ and $v \neq w$. Then $S(u) = S(v) = \varepsilon$ and $\sigma_0(W) = \sigma_1(W)$.
- $u \neq w$ and $v = w$. Then $S(u) = \varepsilon$, $S(v) = (ah)(hb)$. Since S satisfies the monochromatic neighborhood property, all neighbors of v, including u, have color h, so $\sigma_0(W) = (h, a)$, $\sigma_1(W) = (h, b)$.
- $u = w$ and $v \neq w$. Then $S(u) = (ah)(hb)$ and $S(v) = \varepsilon$. Again using the fact that S satisfies the monochromatic neighborhood property, we have $\sigma_0(W) = (ah)$ and $\sigma_1(W) = (bh)$.

In each case we have $S(v) = \sigma_0(W)^{-1} S(u) \sigma_1(W)$. If W has length at least two then we may split W inductively into $W = W_1 W_2$ such that W_2 is of length one, so W_1 is a walk from u to z and W_2 is a walk from z to v. Then we have

$$\begin{aligned}\sigma_1(W) &= \sigma_1(W_1)\sigma_i(W_2)\\ &= \sigma_1(W_1)S(z)^{-1}\sigma_0(W_2)S(v)\\ &= S(u)^{-1}\sigma_0(W)S(v)\ .\end{aligned}$$

If the sequence S has length more than one we use the same idea and split S inductively into $S = S_1 S_2$ such that S_2 has length one. Then for each $v \in V(G)$ we have $S(v) = S_1(v)S_2(v)$ and

$$\begin{aligned}S(v) &= S_1(v)S_2(v)\\ &= S_1(v)\sigma_{\ell-1}(W)^{-1}S_2(u)\sigma_\ell(W)\\ &= \sigma_0(W)^{-1}S(u)\sigma_\ell(W)\ ,\end{aligned}$$

which concludes the proof. ◀

Proof of Lemma 10. Use Lemma 9 to decide in time $O(|E(G)| \cdot (|Q| + |V(G)|))$ whether Q is $\bar{H}$-realizable for α, β, q. If not then Q is not H-realizable for α, β, q. Otherwise, for each vertex $v \in V(G)$, use breadth first search to find a shortest walk W_v from q to v in G in time

$O(|E(G)|)$, then compute $S(v) := \alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v)$, reduce the resulting walk and check the zigzag condition in time $O(|Q| + |V(G)|)$. By Lemma 7, the zigzag condition is satisfied for each vertex $v \in V(G)$ if and only if Q is H-realizable for α, β, q. In total, for each vertex $v \in V(G)$ the computations can be performed in time $O(|Q| + |E(G)|)$. ◀

Proof of Lemma 13. We prove the first statement. Assume there is no vertex of type SYM and that Q satisfies the zigzag condition. Without loss of generality, we may assume that q is of type IN, so Q is IN-compatible. Let v be any other vertex of G and P a path from q to v in G. Since there is no vertex of type SYM we deduce that P is alternating between vertices of type IN and vertices of type OUT. By Lemma 12 S_w satisfies the zigzag condition for each vertex w in P. In particular, S_w does.

It remains to prove the second statement. Let $X = \{v_1, \ldots, v_k\} \subseteq V(G)$ be the vertices of G of type SYM and suppose that S_{v_i} satisfies the zigzag condition for $1 \leq i \leq k$. Let $v \in V(G)$ be any vertex that is not of type SYM and let P be a shortest path from X to v. Again, P is alternating between vertices of type IN and vertices of type OUT and hence for each vertex w of P, we obtain that S_w satisfies the zigzag condition by Lemma 12. ◀

Proof of Lemma 14. Let $V' \subseteq V(G)$ be the set of vertices of type SYM. Notice that V' can be computed in time $O(|E(G)|)$. Suppose first, suppose that $V' = \emptyset$. Then, by statement A the orientation-compatible walks are precisely those of even length that satisfy the zigzag-condition. We can therefore indicate Case 3 with $q := q_0$. Now suppose that $V' \neq \emptyset$. Let $q \in V'$ and apply Lemma 15 to determine in time $O(|V(G)| \cdot |E(G)|)$ all walks from $\alpha(q)$ to $\beta(q)$ that generate symmetric walks with the set $(W_v)_{v \in V'}$ on all vertices of V'. Invoke the second statement of Lemma 13 to deduce that:

1. In Case 1 of Lemma 15 we report Case 1, i.e., there is no orientation-compatible walk.
2. In Case 2 of Lemma 15 we report Case 2 and output Q if Q has even length and Case 1 otherwise.
3. In Case 3 of Lemma 15 we report Case 3.

The total runtime is dominated by the algorithm of Lemma 15, hence $O(|V(G)| \cdot |E(G)|)$ as claimed. ◀

Proof of Lemma 17. Start with $n = 0$. Check if R_0 entirely reduces with $R_0^n P$ and if so then replace n by $n+1$. Similarly, if R_0^{-1} reduces with $R_0^n P$, then replace n by $n-1$. Repeat until none of R_0^{-1} or R_0 entirely reduces with $R_0^n P$. Each step is done in $|R_0|$ and reduces $|R_0^n P|$ by $|R_0|$, so we deduce that this process terminate in time $O(|R_0| + |P|)$. Also observe that $|n_0|$ is polynomial in $|V(G)|$ and $|V(H)|$. ◀

B Additional results

B.1 Shortest H-recoloring

Let H be a fixed digraph. Given a digraph G and two homomorphism $\alpha, \beta : G \to H$, the problem Shortest H-Recoloring asks for the length of a shortest H-recoloring sequence from α to β (if there is none, the shortest length is ∞). The topological approach, in particular the description of H-realizable walks, is also useful for finding shortest H-recoloring sequences. Again, we may follow Wrochna's arguments from [17] to obtain a polynomial-time algorithm for Shortest H-Recoloring under the conditions of of Theorem 1.

▶ **Theorem 21.** *Let H be a loopless digraph that contains no 4-cycle of algebraic girth 0 as a subgraph. Then Shortest H-Recoloring admits a polynomial-time algorithm.*

We observe that for any H-recoloring sequence, the corresponding walk $S(v)$ of a vertex v (see Section 3.1) is reduced.

▶ **Lemma 22** ([17, see Corollary 6.2]). *Let H be a loopless digraph that contains no 4-cycle of algebraic girth 0 as a subgraph. Let $\alpha, \beta : G \to H$ be graph homomorphisms and $S := \sigma_0, \ldots, \sigma_\ell$ a shortest H-recoloring sequence from $\alpha = \sigma_0$ to $\beta = \sigma_\ell$. Then for any vertex $v \in V(G)$, the walk $S(v)$ is reduced.*

Proof. Observe that the length $|S|$ of the H-recoloring sequence S is $|S| = \frac{1}{2}\sum_{v\in V(G)} |S(v)|$. Fix any base vertex $q \in V(G)$ to notice that $Q = S(q)$ is H-realizable. Reducing for $v \in V(G)$ all walks $S(v)$ to $S_r(v)$, Lemma 8 constructs an associated H-recoloring sequence of length $\sum_{v\in V(G)} |S_r(v)|$. Since $S_r(v)$ is the reduction of $S(v)$, we have $|S_r(v)| \leq |S(v)|$. Thus if $|S|$ is minimal then $|S(v)| = |S_r(v)|$ for each $v \in V(G)$ and hence the walks $S(v)$ are reduced. ◀

Following the proof of [17, see Theorem 8.1] nearly word by word and additionally taking care the orientation compatibility yields Theorem 21.

Proof of Theorem 21. By Lemma 22, it suffices to choose a walk $Q \in \Pi'$ from the description in Theorem 16 that minimizes

$$\sum_{v\in V(G)} |S(v)| = \sum_{v\in V(G)} |\alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v)| \ , \tag{1}$$

where W_v is an arbitrary chosen walk from q to v. In cases 1 and 2 of Theorem 16 this is trivial. Recall that in Case 3 we have $Q = R^n \cdot P$ for any $n \in \mathbb{N}$. It is easy to see that $|n| \leq 2|V(G)| + |P|$ in a shortest sequence, since repeating R will eventually increase all summands of (1). It thus suffices to compute (1) for all these choices of n.

In Case 4, consider an H-realizable walk Q, i.e., any reduced walk of even length from $\alpha(q)$ to $\beta(q)$ that satisfies the zigzag condition. Let P_1 be the longest common prefix of Q and $\alpha(W_v) \in \pi(H)$, choosing $v \in V(G)$ to maximize its length. That is, P_1 is longest such that all of P_1 will reduce with $\alpha(W_v)^{-1}$ in some summand of (1). Similarly, let P_2 be the longest common suffix of Q and some $\beta(W_v)^{-1} \in \pi(H)$. Either P_1 and P_2 overlap, or $Q = P_1 Q' P_2$, for some $Q' \in \pi(H)$. In the latter case, since P_1 and P_2 are longest, no element of Q' will reduce in any summand of (1), the sum can be written as

$$\sum_{v\in V(G)} |\alpha(W_v)^{-1} \cdot Q \cdot \beta(W_v)| = \sum_{v\in V(G)} \left(|\alpha(W_v)^{-1} \cdot P_1| + |Q'| + |P_2 \cdot \beta(W_v)|\right).$$

Where we take $\alpha(W_v)^{-1}$ and $\beta(W_v)$ after reduction in $\pi(H)$. Also observe that both P_1 and P_2 must zigzag properly according to the type of q (see Definition 11), i.e., if q is of type IN and $P_1 = (a_1 a_2) \ldots (a_{n-1} a_n)$, then $a_1 \leftarrow a_2 \rightarrow a_3 \ldots$ up to a_n (so the orientation of the last arc depends on parity). Similarly if $P_2 = (b_n b_{n-1}) \ldots (b_2 b_1)$, then $\ldots b_3 \leftarrow b_2 \rightarrow b_1$. Thus we can guess P_1 by enumerating all prefixes of all $\alpha(W_v)$ that zigzag properly according to the type of q, similarly guess P_2 and guess how much they overlap. In case they do not overlap, the sum is minimized by taking Q' to be an arbitrary shortest path of appropriate parity, and that zigzags correctly according to the type of q, from the tail of P_1 to the head of P_2 in H. Enumerating all possibilities for (the length of) P_1, P_2 and the overlap can be done in polynomial time, and a shortest path of given parity that zigzags correctly in H can be found by duplicating every vertex, i.e., finding a shortest path that zigzags correctly in the tensor product $H \times K_2$. ◀

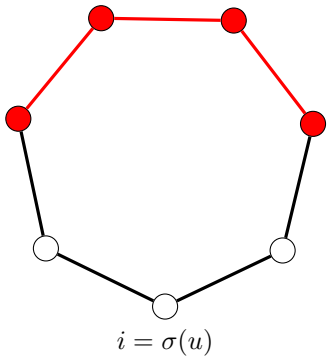


Figure 5 C_p for the circular clique $G_{p,p'}$ with $p = 7$ and $p' = 2$. If $(uv) \in E(G)$ and $\sigma(u) = i$ is the vertex in the bottom, then $\sigma(v)$ must belong to the path in red.

B.2 H-recoloring for circular cliques

Given two integers p, p' such that $p/p' \geq 2$, the circular clique $G_{p,p'}$ has as vertex set the set $\mathbb{Z}_p$ of integers modulo p and has edge set $\{(ij) \mid i - j \bmod p \leq p'\}$. The p-cycle C_p is a graph on the vertex set $\mathbb{Z}_p$ and edge set $\{(i(i+1)) \mid i \in \mathbb{Z}_p\}$. We sketch a proof of the following result by Brewster at al. from [4] using the topological approach of Wrochna.

▶ **Theorem 23** (See [4]). *Let p, p' be fixed positive integers with $2 \leq p/p' < 4$. Let $H := G_{p,p'}$ be the circular clique of parameters p, p'. Then H-Recoloring admits a polynomial-time algorithm.*

By the definition of homomorphisms $G \to G_{p,p'}$ we have for each neighbor u of a vertex $v \in V(G)$, the color of v must belong to a path $P_u(v)$ on C_p which depends on the color of u (see Figure 5). The constraint $p/p' < 4$ is necessary to ensure that the intersection $P(v) = \bigcap_{u \in N(v)} P_u(v)$ is a path on C_p, which is the set of all colors to which the color of v can change. Thus, we can associate vertex walks $S(v)$ in C_p to an H-recoloring sequence S by associating to each color change of v the (unique) walk in $P(v)$ between the two consecutive colors. Also, we convert walks W in $G_{p,p'}$ to walks $\widetilde{W}$ in C_p as follows: replace each edge (ab) in W by the walk $(a\,(a+1)) \dots ((b-1)\,b)$ in C_p and concatenate to obtain $\widetilde{W}$. We can prove the following lemma.

▶ **Lemma 24.** *Let S be a $G_{p,p'}$-recoloring sequence from α to β, then for any $u, v \in V(G)$ and any uv walk W, we have $S(v) = \widetilde{\alpha(W)}^{-1} S(u) \widetilde{\beta(W)}$ in $\pi(C_p)$.*

This lemma implies in particular that vertex walks must be topologically valid: for any vertex $q \in V(G)$ and any closed walk C from q to q, $\widetilde{\beta(C)} = S(q)^{-1} \widetilde{\alpha(C)} S(q)$ in $\pi(C_p)$. In particular, the walk of any vertex $q \in V(G)$ generates all the others. Again, we can define that a closed walk $C = (v_0\, v_1) \dots (v_{n-1}\, v_n)$ from $q = v_0$ to $q = v_n$ is α-tight if $\alpha(v_{i+1}) = \alpha(v_i) + p' \bmod p$ for all $i \in \{0, \dots, n-1\}$. We say that a walk Q from $\alpha(q)$ to $\beta(q)$ in C_p is realizable if there is a $G_{p,p'}$-recoloring sequence from α to β such that $Q = S(q)$ in $\pi(C_p)$. We obtain again a similar construction theorem the as in the previous sections.

▶ **Theorem 25.** *Let $\alpha, \beta : G \to G_{p,p'}$ be two $G_{p,p'}$-colorings of a graph G. Furthermore, let $q \in V(G)$ be any vertex and let Q be a reduced walk from $\alpha(q)$ to $\beta(q)$. Then Q is realizable for α, β, q if and only if*

1. *Q is topologically valid for α, β, q.*
2. *For each α-tight closed walk in G and any vertex v on this walk, for any walk W from v to q, we have $Q = \widetilde{\alpha(W)}^{-1}\widetilde{\beta(W)}$ in $\pi(H)$.*

Furthermore, there is an algorithm that, given a reduced walk Q, constructs a H-recoloring sequence S from α to β or certifies that Q cannot satisfy one of the previous conditions. This algorithm runs in time $O(|V(G)|^2 + |V(G)| \cdot |Q|)$. The H-recoloring sequence S is such that $S(q) = Q$.

The proof of Theorem 25 is essentially the same that the proof of Theorem 29 in [14] or of Theorem 6.1 in [17]. The only difference is that we will provide recoloring sequences that are longer since the color changes will follow edges in C_p. Using Theorem 5 from [14] (and that an α-tight closed walk can be found in polynomial time if there is one), we directly obtain a description of realizable walks:

▶ **Theorem 26.** *Let $\alpha, \beta \colon G \to H = G_{p,p'}$ and $q \in V(G)$. Let Π be the set of all reduced walks that are realizable for α, β, q. One of the following holds:*

1. *$\Pi = \emptyset$.*
2. *$\Pi = \{Q\}$ for some $Q \in \pi(C_p)$.*
3. *$\Pi = \{R^n P \mid n \in \mathbb{Z}\}$, for some $R, P \in \pi(C_p)$.*

Furthermore, there is an algorithm that determines in time $O(|V(G)| \cdot |E(G)|)$ which case holds and outputs Q or R, P in cases 2, 3 such that $|Q|, |R|, |P|$ are bounded by the time $O(|V(G)| \cdot |E(G)|)$.

Since here $\pi(C_p) \simeq \mathbb{Z}$, it turns out that the case where all walks from $\alpha(q)$ to $\beta(q)$ in C_p are realizable happens in Case 3 when R turns a single time around C_p. So Theorem 23 directly follows. Notice that this result is already proved in [4] together with a dichotomy Theorem for H-recoloring where H-is a circular clique. Our goal here was to show how the topological approach can be exploited even without the monochromatic neighborhood property or the push-or-pull property.

B.3 H-recoloring for transitive tournaments

The transitive tournament T_n is an acyclic orientation of the complete graph on n vertices. Dochtermann [7] showed that when H is a transitive tournament then H-Recoloring is always Yes and a corresponding H-recoloring sequence can be found in polynomial-time. However, we can easily recover this result using the following construction: Say $\{1, \dots, n\}$ are the vertices of T_n, with $i \to j$ if and only if $i < j$. Let P_n be the undirected path on the same vertices with only the edges $\{i, i+1\}$, $1 \leq i < n$. So for any instance (G, α, β) of H-Recoloring, for each $v \in V(G)$, there is a unique reduce walk S_v in P_n from $\alpha(v)$ to $\beta(v)$ which can be realized (as a vertex walk) by the following special case of the move-forward algorithm (see Lemma 30 in [14]): for each vertex v of G, try to move v onto its next color in S_v and repeat until we reach β. Suppose by contradiction that at any step σ, there is a cycle of obstruction $C = (u_0u_1)\dots(u_nu_0)$ where for all i, u_i prevents its neighbor u_i from moving. Suppose without loss of generality that S_{u_0} is increasing in P_n. So $\sigma(u_1) > \sigma(u_0)$, hence $u_0 \to u_1$ and eventually S_{u_1} since it must remain above $\sigma(u_0)$. So by induction, all walks S_{u_0} are increasing and we have a directed cycle $u_0 \to u_1 \to \dots \to u_n \to u_0$ in G, which is impossible.

The homotopy group $\pi(P_n)$ being trivial, it is not necessary to define topological validity in this case.

An $\mathcal{O}(3.82^k)$ Time $\mathcal{FPT}$ Algorithm for Convex Flip Distance

Haohong Li
Department of Computer Science, Lafayette College, Easton, PA, USA

Ge Xia
Department of Computer Science, Lafayette College, Easton, PA, USA

Abstract

Let $\mathcal{P}$ be a convex polygon in the plane, and let $\mathcal{T}$ be a triangulation of $\mathcal{P}$. An edge e in $\mathcal{T}$ is called a diagonal if it is shared by two triangles in $\mathcal{T}$. A *flip* of a diagonal e is the operation of removing e and adding the opposite diagonal of the resulting quadrilateral to obtain a new triangulation of $\mathcal{P}$ from $\mathcal{T}$. The *flip distance* between two triangulations of $\mathcal{P}$ is the minimum number of flips needed to transform one triangulation into the other. The Convex Flip Distance problem asks if the flip distance between two given triangulations of $\mathcal{P}$ is at most k, for some given parameter $k \in \mathbb{N}$.

We present an $\mathcal{FPT}$ algorithm for the Convex Flip Distance problem that runs in time $\mathcal{O}(3.82^k)$ and uses polynomial space, where k is the number of flips. This algorithm significantly improves the previous best $\mathcal{FPT}$ algorithms for the problem.

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

Keywords and phrases Flip distance, Rotation distance, Triangulations, Exact algorithms, Parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.44

Related Version *Full Version*: https://arxiv.org/abs/2209.13134

1 Introduction

Let $\mathcal{P}$ be a convex polygon in the plane. A *triangulation* of $\mathcal{P}$ adds edges between non-adjacent points in $\mathcal{P}$ to produce a planar graph $\mathcal{T}$ such that all faces except the outer face in $\mathcal{T}$ are triangles. The edges in the triangulation $\mathcal{T}$ not on the convex hall are called *diagonals*. The number of diagonals in $\mathcal{T}$ is denoted by $\phi(\mathcal{T})$.

A *flip* of a diagonal e in $\mathcal{T}$ removes e and adds the opposite diagonal of the resulting quadrilateral, thus transforming $\mathcal{T}$ to another triangulation $\mathcal{T}'$ of $\mathcal{P}$. Given two triangulations $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$ of $\mathcal{P}$, the *flip distance* between them, denoted as $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final})$, is the minimum number of flips required to transform $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ (or equivalently from $\mathcal{T}_{final}$ to $\mathcal{T}_{init}$). The Convex Flip Distance problem is formally defined as:

Convex Flip Distance
Given: Two triangulation $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$ of a convex polygon $\mathcal{P}$ in the plane.
Parameter: k.
Question: Is $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final})$ at most k?

The number of ways to triangulate a convex $(n+2)$-gon is C_n, the n-th Catalan number. Consequently, there exists an isomorphism between triangulations of a convex polygon and a plethora of counting problems, such as binary trees, Dyck paths, etc [28]. In particular, there exists a bijection between the set of triangulations of a convex $(n+2)$-gon and the set of full binary trees with n internal nodes. Furthermore, flipping an edge in a convex polygon triangulation corresponds to rotating a node in a binary tree, which is an essential operation for maintaining balanced binary search trees. The *rotation distance* between two binary trees


© Haohong Li and Ge Xia;
licensed under Creative Commons License CC-BY 4.0

with the same number of nodes is the minimum number of rotations needed to transform one binary tree into the other. Thus, finding the flip distance between two triangulations of a convex $(n+2)$-gon is equivalent to finding the rotation distance between two binary trees with n internal nodes.

Determining the computational complexity of Convex Flip Distance (equivalently the rotation distance problem between binary trees) has been an important open problem. Despite extensive research on this problem [9, 3, 6, 22, 24, 27, 19, 12, 8, 7, 21, 5], it is still unknown whether it is NP-hard or not.

In 1982, Culik and Wood [9] first studied the rotation distance problem between binary trees and proved an upper bound of $2n-2$, where n is the number of internal nodes of the trees. In 1988, Sleator, Tarjan, and Thurston [27] improved the upper bound to $2n-6$, which is proven to be tight for $n \geq 11$ by Pournin [26].

In 1998, Li and Zhang gave two polynomial-time approximation algorithms [19] for computing the flip distance of convex polygon triangulations. One algorithm has an approximation ratio of $2-\frac{2}{4(d-1)(d+6)+1}$ given that each vertex is an endpoint of at most d diagonals. Another algorithm has an approximation ratio of 1.97 provided that both triangulations do not contain internal triangles. Cleary and St. John [8] gave a linear-time 2-approximation algorithm for rotation distance in 2009.

In 2009, Cleary and St. John [7] gave a kernel of size $5k$ for Convex Flip Distance and presented an $\mathcal{O}^*((5k)^k)$-time fixed-parameter tractable ($\mathcal{FPT}$) algorithm based on this kernel (the $\mathcal{O}^*$ notation suppresses polynomial factors in the input size). The upper bound on the kernel size of Convex Flip Distance was subsequently improved to $2k$ by Lucas [21], who also gave an $\mathcal{O}^*(k^k)$-time $\mathcal{FPT}$ algorithm for this problem. Very recently, Celvo and Kelk improved the kernel size to $(1+\epsilon)k$ for any $\epsilon > 0$ in 2021 [5], although their result does not lead to an improved approximation algorithm over [19] and [8].

The generalized version of the Convex Flip Distance problem, referred to as the General Flip Distance problem, is the flip distance between triangulations of a point set in general positions in the plane. The General Flip Distance problem is also a fundamental and challenging problem, and has also been extensively studied [18, 1, 2, 4, 7, 13, 14, 20, 25, 17, 11].

In 1972, Lawson [18] gave an $\mathcal{O}(n^2)$ upper bound on General Flip Distance, where n is the number of points in $\mathcal{S}$ [18]. The complexity of the General Flip Distance problem was resolved in 2012 by Lubiw and Pathak [20] who showed the problem to be $\mathcal{NP}$-complete. Simultaneously, and independently, the problem was shown to be $\mathcal{APX}$-hard by Pilz [25]. In 2015, Aichholzer Mulzer, and Pilz [2] proved that the flip distance problem is $\mathcal{NP}$-complete for triangulations of a simple (but not convex) polygon.

Very recently, Kanj, Sedgwick, and Xia [16] presented the first $\mathcal{FPT}$ algorithm for General Flip Distance that runs in $\mathcal{O}(n+kc^k)$, where $c \leq 2 \cdot 14^{11}$. Their approach defines a dependency relation for a sequence of flips: some flips required some other flips to be performed first. They proved that any topological sort of the directed acyclic graph (DAG) modeling this dependency relation yields the equivalent end result. Their algorithm simulates a "non-deterministic walk" that tries the possible flips to find a topological sort of the DAG representing an optimal solution. Refining this approach, Feng, Li, Meng, and Wang improved the $\mathcal{FPT}$ algorithm to run in $\mathcal{O}(n+k \cdot 32^k)$ [11], which currently stands as the best $\mathcal{FPT}$ algorithm for both General Flip Distance and Convex Flip Distance. No more efficient polynomial space algorithm was known for Convex Flip Distance despite its structural properties. We note in passing here that although a straightforward algorithm based on breadth-first search (BFS) can find the flip distance between triangulations of a convex polygon in time $\mathcal{O}^*(C_n) = \mathcal{O}^*(4^n)$, where C_n is the n^{th} Catalan number, it requires exponential space.

In this paper, we present an $\mathcal{FPT}$ algorithm for the Convex Flip Distance problem that runs in time $\mathcal{O}(3.82^k)$ and uses polynomial space, where k is the number of flips. Instead of performing a "non-deterministic walk" as in [16, 11], our algorithm computes a topological sort of the DAG representing an optimal solution by repeatedly finding and removing its source nodes. This approach allows us to take advantage of the structural properties of Convex Flip Distance and design a simple yet significantly more efficient algorithm.

Our algorithm is closely related to algorithms in [16] and [11]. All three algorithms rely on the same structure results from [16] which state that (1) the flips in an optimal solution have a dependency relation that can be modeled as a DAG, and (2) any topological sort of this DAG yields an optimal solution. Therefore, three algorithms address the same problem: how to find a topological sort of this unknown DAG. The algorithm of Kanj, Sedgwick, and Xia [16] simulates a "non-deterministic walk". For any diagonal e, the algorithm will either flip e (when this flip is a source in the DAG) or move on to one of e's neighbors (when a neighbor of e must be flipped before e can be flipped). Through a sequence of such "flip/move"-type local actions, a topological sort of the DAG (if exists) is found in time $\mathcal{O}(n + kc^k)$, where $c \leq 2 \cdot 14^{11}$. The algorithm of Feng, Li, Meng, and Wang [11] refines the "non-deterministic walk" approach by reducing the number of the action types and streamlining the backtracking in the walk, resulting in an improved algorithm that runs in time $\mathcal{O}(n + k \cdot 32^k)$. Different from the previous "walk"-based approach, our algorithm performs a topological sort of the DAG by repeatedly finding and removing source nodes in the DAG, similar to Kahn's algorithm [15]. After the current source nodes of the DAG are removed, the new source nodes can be found among the neighbors of the flipped diagonals. Based on this more efficient approach and by exploiting the structural properties of Convex Flip Distance, we significantly improve the running time of our algorithm to $\mathcal{O}(3.82^k)$.

2 Preliminaries

2.1 Flips, triangulations, and flip distance

For any flip f, we use the notation $f^{\leftarrow}$ to denote the underlying diagonal e on which f is performed, and the notation $f^{\rightarrow}$ to denote the new diagonal $\overline{e}$ added when f is performed. For any two diagonals e_1 and e_2 in $\mathcal{T}$, we say they are *neighbors* if they appear in the same triangle and say they are *independent* if they are not neighbors. Note that two independent diagonals in $\mathcal{T}$ can share an endpoint, as long as they are not in the same triangle.

Let $\mathcal{T}$ and $\mathcal{T}'$ be two triangulations of $\mathcal{P}$. We refer to $(\mathcal{T}, \mathcal{T}')$ as *a pair of triangulations* of $\mathcal{P}$. Denote by $\mathrm{C}(\mathcal{T}, \mathcal{T}')$ the number of common diagonals shared by $\mathcal{T}$ and $\mathcal{T}'$. We say a sequence of flips $F = \langle f_1, \ldots, f_r \rangle$ *transforms* a triangulation $\mathcal{T}$ to $\mathcal{T}'$, denoted as $\mathcal{T} \xrightarrow{F} \mathcal{T}'$, if there exist triangulations $\mathcal{T}_0, \ldots, \mathcal{T}_r$ such that $\mathcal{T}_0 = \mathcal{T}$, $\mathcal{T}_r = \mathcal{T}'$, and performing flip f_i in $\mathcal{T}_{i-1}$ results in $\mathcal{T}_i$, for $i = 1, \ldots, r$. Such a transformation $\mathcal{T} \xrightarrow{F} \mathcal{T}'$ is referred to as a *path* from $\mathcal{T}$ to $\mathcal{T}'$ following F. The *length* of F (equivalently, the length of the path $\mathcal{T} \xrightarrow{F} \mathcal{T}'$), denoted $|F|$, is the number of flips in it. The flip distance $\mathrm{Dist}(\mathcal{T}, \mathcal{T}')$ is the length of the shortest path between T and T'. A sequence of flips F such that $\mathcal{T} \xrightarrow{F} \mathcal{T}'$ is a shortest path is called an *optimal* (or *minimum*) *solution* of the pair $(\mathcal{T}, \mathcal{T}')$.

Let f_i and f_j be two flips in $F = \langle f_1, \ldots, f_r \rangle$ such that $1 \leq i < j \leq r$. The flip f_j is said to be *adjacent* to the flip f_i, denoted $f_i \to f_j$, if $f_i^{\rightarrow}$ is a neighbor of $f_j^{\leftarrow}$ in $\mathcal{T}_{j-1}$. This adjacency relation defines a partial order among flips in F: if $f_i \to f_j$ then f_i must precede f_j because $f_i^{\rightarrow}$ is a neighbor of $f_j^{\leftarrow}$ at the moment when f_j is performed. Therefore, the adjacency relation on the flips in F can be naturally represented by a directed acyclic graph (DAG), denoted $\mathcal{D}_F$, where the nodes of $\mathcal{D}_F$ are the flips in F, and its arcs represent the (directed) adjacencies in F.

Recall that a topological sort of a DAG is *any* ordering of its nodes that satisfies: For any directed arc (u, v) in the DAG, u appears before v in the ordering. There could be many different topological sorts of $\mathcal{D}_F$, but the following lemma by Kanj, Sedgwick, and Xia [16] asserts that all of them yield the same outcome:

▶ **Lemma 1** ([16]). *Let $\mathcal{T}_0$ be a triangulation and let $F = \langle f_1, \ldots, f_r \rangle$ be a sequence of flips such that $\mathcal{T}_0 \xrightarrow{F} \mathcal{T}_r$. Let $\pi(F)$ be a permutation of the flips in F such that $\pi(F)$ is a topological sort of $\mathcal{D}_F$. Then $\pi(F)$ is a valid sequence of flips such that $\mathcal{T}_0 \xrightarrow{\pi(F)} \mathcal{T}_r$. Furthermore, the DAG $\mathcal{D}_{\pi(F)}$, defined based on the sequence $\pi(F)$, is the same directed graph as $\mathcal{D}_F$.*

▶ **Definition 2.** *Let $(\mathcal{T}_{init}, \mathcal{T}_{final})$ be a pair of triangulations. Let e be a diagonal in $\mathcal{T}_{init}$. We say e is a* free-diagonal with respect to $\mathcal{T}_{final}$ *if flipping e creates a new diagonal $\overline{e}$ that is in $\mathcal{T}_{final}$. When the context is clear, we simply refer to e as a free-diagonal.*

▶ **Lemma 3.** *If e_1 and e_2 are two free-diagonals in $\mathcal{T}_{init}$, then e_1 and e_2 are independent.*

Proof. Let $\overline{e}_1$ and $\overline{e}_2$ be the edges created by flipping e_1 and e_2 in $\mathcal{T}_{init}$, respectively. By Definition 2, both $\overline{e}_1$ and $\overline{e}_2$ are in $\mathcal{T}_{final}$. If e_1 and e_2 are neighbors, then $\overline{e}_1$ and $\overline{e}_2$ intersect each other, contradicting the fact that both $\overline{e}_1$ and $\overline{e}_2$ are in $\mathcal{T}_{final}$. ◀

The following lemma by Sleator, Tarjan and Thurston [27] shows that free-diagonals can be safely flipped and common diagonals will never be flipped in any shortest path.

▶ **Lemma 4** ([27]). *Let $(\mathcal{T}_{init}, \mathcal{T}_{final})$ be a pair of triangulations. (a) If $\mathcal{T}_{init}$ contains a free-diagonal e, then there exists a shortest path from $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ where e is flipped first. (b) If $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$ share a diagonal e in common, then every shortest path from $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ never flips e.*

In a sequence of flips, a previously non-free diagonal can become a free-diagonal only when one of its neighbors is flipped.

▶ **Lemma 5.** *Let $F = \langle f_1, \ldots, f_r \rangle$ be a sequence of flips such that $\mathcal{T}_{init} = \mathcal{T}_0 \xrightarrow{F} \mathcal{T}_r = \mathcal{T}_{final}$ is a shortest path. Let e be a common diagonal of $\mathcal{T}_{i-1}$ and $\mathcal{T}_i$, for $1 \leq i \leq r$. If e is not a free-diagonal in $\mathcal{T}_{i-1}$ and is a free-diagonal in $\mathcal{T}_i$, then $f_i^{\rightarrow}$ is a neighbor of e.*

Proof. Suppose that $f_i^{\rightarrow}$ is not a neighbor of e. Then Q_e, the quadrilateral associated with e, remains the same in $\mathcal{T}_i$ as in $\mathcal{T}_{i-1}$. Therefore, flipping e in $\mathcal{T}_{i-1}$ creates the same diagonal as flipping e in $\mathcal{T}_i$, a contradiction to the fact that e is not a free-diagonal in $\mathcal{T}_{i-1}$ and is a free-diagonal in $\mathcal{T}_i$. ◀

▶ **Definition 6.** *A pair of triangulations $(\mathcal{T}_{init}, \mathcal{T}_{final})$ of a convex polygon $\mathcal{P}$ is called* trivial *if* $\mathrm{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) = \phi(\mathcal{T}_{init}) - \mathrm{C}(\mathcal{T}_{init}, \mathcal{T}_{final})$.

▶ **Lemma 7.** *If a pair of triangulations $(\mathcal{T}_{init}, \mathcal{T}_{final})$ is trivial, then every flip in an optimal solution is a flip of a free-diagonal. Furthermore, it takes linear time to decide if a pair $(\mathcal{T}_{init}, \mathcal{T}_{final})$ is trivial.*

Proof. If $\mathrm{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) = \phi(\mathcal{T}_{init}) - \mathrm{C}(\mathcal{T}_{init}, \mathcal{T}_{final})$, then every flip in an optimal solution F of $(\mathcal{T}_{init}, \mathcal{T}_{final})$ must create an additional common diagonal between $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$, which means that every flip in F is performed on a free-diagonal.

To decide if a pair $(\mathcal{T}_{init}, \mathcal{T}_{final})$ is trivial, first find all initial free-diagonals in $\mathcal{T}_{init}$ and add them to a queue. Then flip the free-diagonals in the queue. By Lemma 5, new free-diagonals must be neighbors of previous flips and hence can be found and added to

the queue as the previous free-diagonals are flipped. The pair $(\mathcal{T}_{init}, \mathcal{T}_{final})$ is trivial if and only if there are always free-diagonals in the queue to be flipped until $\mathcal{T}_{init}$ is transformed to $\mathcal{T}_{final}$. Since the flip distance between $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$ is at most $2n-4$, where n is the number of diagonals in $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$ [26], this process takes linear time. ◀

▶ **Definition 8.** *Let $(\mathcal{T}_{init}, \mathcal{T}_{final})$ be a pair of triangulations. Let I be a set of diagonals in $\mathcal{T}_{init}$. We say I is a* safe-set of diagonals with respect to $\mathcal{T}_{final}$*, or simply* safe-set*, if*

1. *the diagonals in I are pair-wise independent, and*
2. *for any permutation $\pi(I)$ of I, there is a shortest path from $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ such that the diagonals in I are flipped first, in the same order as $\pi(I)$.*

The set of diagonals corresponding to the source nodes of $\mathcal{D}_F$ is a safe-set.

▶ **Lemma 9.** *Let $F = \langle f_1, \ldots, f_r \rangle$ be a sequence of flips such that $\mathcal{T}_{init} = \mathcal{T}_0 \xrightarrow{F} \mathcal{T}_r = \mathcal{T}_{final}$ is a shortest path. Let $SC = \{f_{sc_1}, \ldots, f_{sc_l}\}$ be the set of source nodes in $\mathcal{D}_F$. The set of diagonals $I = \{f^{\leftarrow}_{sc_1}, \ldots, f^{\leftarrow}_{sc_l}\}$ is a safe-set in $\mathcal{T}_{init}$.*

Proof. Let $f_{sc_i}, f_{sc_j} \in SC$, $i \neq j$, be two source nodes in $\mathcal{D}_F$. By Lemma 1, we may assume that f_{sc_i} is the first flip in F. If $f^{\leftarrow}_{sc_i}$ and $f^{\leftarrow}_{sc_j}$ share a triangle in $\mathcal{T}_{init}$, after flipping f_{sc_i}, $f^{\rightarrow}_{sc_i}$ and $f^{\leftarrow}_{sc_j}$ share a triangle in $\mathcal{T}_1$, and hence by Lemma 1, there is a directed path from f_{sc_i} to f_{sc_j} in $\mathcal{D}_F$, contradicting to the fact that f_{sc_i} is a source in $\mathcal{D}_F$. Therefore, the diagonals in I are independent.

For an arbitrary permutation of $\pi(I)$, there is a topological sort of $\mathcal{D}_F$ that begins with the flips in SC according to the order of $\pi(I)$. By Lemma 1, there is a shortest path between $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$ that flips the diagonal in I first, in the order of $\pi(I)$. Therefore, I is a safe-set. ◀

2.2 Counting matchings in binary trees

The following lemma will be useful in the analysis of the running time of our algorithm.

▶ **Lemma 10.** *Let T_u be a binary tree rooted at a node u with n nodes and $n-1$ edges. A matching in T_u is a subset of edges in T_u that do not share any endpoint (we consider an empty set to be a matching). Let $\mathcal{E}_u$ be the set of matchings in T_u. Let $\mathcal{E}^-_u$ be the set of matchings in T_u that exclude the edge(s) incident on u. Then $|\mathcal{E}^-_u| \leq F_n$ and $|\mathcal{E}_u| \leq F_{n+1}$, where F_n is the n-th Fibonacci number.*

Proof. Let $\mathcal{E}^+_u$ be the set of matchings in T_u that include exactly one edge incident on u. Then $|\mathcal{E}_u| = |\mathcal{E}^+_u| + |\mathcal{E}^-_u|$. The lemma is proven by induction on n.

When $n = 1$, T_u is a leaf. Thus $\mathcal{E}^+_u = \emptyset$ and $\mathcal{E}^-_u = \{\emptyset\}$. Therefore, $|\mathcal{E}^+_u| = 0 \leq F_0$, $|\mathcal{E}^-_u| = 1 \leq F_1$, and $|\mathcal{E}_u| = |\mathcal{E}^+_u| + |\mathcal{E}^-_u| = 1 \leq F_2$.

When $n = 2$, T_u has a single edge that connects u to a single leaf child v. Thus, $\mathcal{E}^+_u = \{uv\}$ and $\mathcal{E}^-_u = \{\emptyset\}$. Therefore, $|\mathcal{E}^+_u| = 1 \leq F_1$, $|\mathcal{E}^-_u| = 1 \leq F_2$, and $|\mathcal{E}_u| = |\mathcal{E}^+_u| + |\mathcal{E}^-_u| = 2 \leq F_3$.

Now suppose that $n \geq 3$. We distinguish 2 cases.

(a) u has only one child v. In this case, T_v, the subtree rooted at v, has $n-1$ nodes and $n-2$ edges. Every matching in $\mathcal{E}^+_u$ includes uv and hence excludes edges in T_v incident on v. Thus $|\mathcal{E}^+_u| = |\mathcal{E}^-_v| \leq F_{n-1}$ by the induction hypothesis. Every matching in $\mathcal{E}^-_u$ excludes uv and hence is also a matching in T_v. Therefore, $|\mathcal{E}^-_u| = |\mathcal{E}_v| \leq F_{(n-1)+1} = F_n$ by the induction hypothesis. Finally, $|\mathcal{E}_u| = |\mathcal{E}^+_u| + |\mathcal{E}^-_u| \leq F_{n-1} + F_n = F_{n+1}$. The statement is true.

(b) u has two children v, w. Let the number of nodes in T_v and T_w be n_1 and n_2, respectively. Then $n = n_1 + n_2 + 1$. We consider $\mathcal{E}^-_u$ and $\mathcal{E}^+_u$ separately:

(i) The matchings in $\mathcal{E}_u^-$ exclude both uv and uw. Therefore, $|\mathcal{E}_u^-| = |\mathcal{E}_v| \cdot |\mathcal{E}_w| \leq F_{n_1+1}F_{n_2+1} \leq F_{n_1+n_2+1} \leq F_n$ by the induction hypothesis and the Honsberger's Identity of Fibonacci numbers.

(ii) The matchings in $\mathcal{E}_u^+$ include exactly one of the edges uv and uw. The number of matchings that include uv and exclude uw is $|\mathcal{E}_v^-| \cdot |\mathcal{E}_w| \leq F_{n_1}F_{n_2+1}$ by the induction hypothesis. The number of matchings that include uw and exclude uv is $|\mathcal{E}_v| \cdot |\mathcal{E}_w^-| \leq F_{n_1+1}F_{n_2}$ by the induction hypothesis. Therefore, $|\mathcal{E}_u^+| \leq F_{n_1}F_{n_2+1} + F_{n_1+1}F_{n_2}$.

Finally, $|\mathcal{E}_u| = |\mathcal{E}_u^-| + |\mathcal{E}_u^+| \leq F_{n_1+1}F_{n_2+1} + F_{n_1}F_{n_2+1} + F_{n_1+1}F_{n_2} = F_{n_1+2}F_{n_2+1} + F_{n_1+1}F_{n_2} = F_{n_1+n_2+2} = F_{n+1}$, where the second last equality is the Honsberger's Identity of Fibonacci numbers.

This completes the proof. ◀

2.3 Parameterized complexity

A *parameterized problem* is a set of instances of the form (x, k), where x is the input instance and $k \in \mathbb{N}$ is the *parameter*. A parameterized problem is *fixed-parameter tractable* ($\mathcal{FPT}$) if there is an algorithm that solves the problem in time $f(k)|x|^c$, where f is a computable function and $c > 0$ is a constant. We refer to [10, 23] for more information about parameterized complexity.

3 The algorithm and its analysis

3.1 The basic ideas

Given an instance $(\mathcal{T}_{init}, \mathcal{T}_{final}, k)$ of Convex Flip Distance, our algorithm decides whether there is a sequence of flips $F = \langle f_1, \ldots, f_r \rangle$, $r \leq k$ that transforms $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$.

By Lemma 4, common diagonals will never be flipped and free diagonals can be safely flipped. Thus, we can assume that $(\mathcal{T}_{init}, \mathcal{T}_{final})$ does not have any common diagonals, i.e., $\mathrm{C}(\mathcal{T}_{init}, \mathcal{T}_{final}) = 0$, and that there are no free-diagonals in the initial triangulation $\mathcal{T}_{init}$. Therefore, we can assume $n \leq k \leq 2n - 4$, where $n = \phi(\mathcal{T}_{init})$.

In Preliminaries, we showed that we can represent a minimum solution F to an instance $(\mathcal{T}_{init}, \mathcal{T}_{final})$ using a DAG $\mathcal{D}_F$, and that any topological sort of $\mathcal{D}_F$ is a minimum solution (Lemma 1). Therefore, to solve an instance of Convex Flip Distance, it suffices to compute a topological sort of $\mathcal{D}_F$.

Intuitively, our algorithm computes a topological sort of $\mathcal{D}_F$ by repeatedly finding and removing its source nodes, similar to Kahn's algorithm [15]. The algorithm uses a branch-and-bound approach to find and flip the source nodes in $\mathcal{D}_F$.

This seemingly simple approach faces two technical challenges. The design of our algorithm revolves around addressing them.

First, how to find the source nodes of the unknown $\mathcal{D}_F$ without trying all possible subsets of the diagonals? The set I of initial source nodes of $\mathcal{D}_F$ must be a subset of independent diagonals in $\mathcal{T}_{init}$. Our algorithm enumerates all subsets of independent diagonals in $\mathcal{T}_{init}$, whose number is at most F_{n+1}, the $(n+1)$-th Fibonacci number (Lemma 14). Afterward, any new source node f_j of $\mathcal{D}_F$ must be adjacent to a previous source node f_i. This means that the underlying diagonal of f_j is a neighbor of the new diagonal created by f_i. Therefore, when flipping a source node, our algorithm adds all neighbors of the new diagonal to a candidate pool S from which new source nodes are chosen. In fact, the neighbors of the new diagonal are added as *pairs* in S because at most one diagonal in each pair may be chosen as a new source node. The next set of new source nodes of $\mathcal{D}_F$ will be chosen by branching on the edge-pairs in S. This method significantly reduces the search space of the source nodes.

Second, how to flip free-diagonals without increasing the branching factor of the algorithm? By Lemma 4, any free-diagonals can be safely flipped. If there is a free-diagonal e in $\mathcal{T}_{init}$, our algorithm flips it to create a new diagonal $\overline{e}$. Since $\overline{e}$ is a common diagonal and hence will never be flipped again, the instance $(\mathcal{T}_{init}, \mathcal{T}_{final})$ is then partitioned along $\overline{e}$ into two smaller instances. The candidate pool S is also partitioned accordingly and passed on to the two smaller instances. This method, through careful analysis, allows the free-diagonals to be flipped "for free" essentially.

3.2 The algorithm

FlipDist$(\mathcal{T}_{init}, \mathcal{T}_{final}, k)$
Input: Two triangulations $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$, a parameter k
Precondition: $C(\mathcal{T}_{init}, \mathcal{T}_{final}) = 0$ and there is no free-diagonals in $\mathcal{T}_{init}$
Output: *True* if $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) \leq k$; *False* otherwise.
0. If $\phi(\mathcal{T}_{init}) > k$, return *False*.
1. Iterate through all subsets I of independent diagonals in $\mathcal{T}$, as follows:
  1.1. For each diagonal $e \in \mathcal{T}$, if none of the neighbors of e is in I, branch on two choices: (1) include e in I; (2) do not include e in I.
  1.2. At the end of the branching, if I is non-empty, do:
  If **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ returns *True*, then return *True*.
2. Return *False*.

Figure 1 The function **FlipDist**.

FlipDist-I$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$
Input: Two triangulations $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$, a parameter k, and a set of independent diagonals I.
Precondition: $C(\mathcal{T}_{init}, \mathcal{T}_{final}) = 0$ and there is no free-diagonals in $\mathcal{T}_{init}$
Output: *True* if $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) \leq k$; *False* otherwise.
0. If $\phi(\mathcal{T}_{init}) > k - |I|$, return *False*; If $\phi(\mathcal{T}_{init}) = 0$ and $k \geq 0$, return *True*.
1. Create an empty set S. For each edge e in I, do:
  1.1. Flip e to create a new edge $\overline{e}$.
  1.2. Let $\Delta_1 = \{e_1, e_1', \overline{e}\}$ and $\Delta_2 = \{e_2, e_2', \overline{e}\}$ be the triangles on either side of $\overline{e}$. Add the pairs (e_1, e_1') and (e_2, e_2') to the set S.
2. Return **FlipDist-S**$(\overline{\mathcal{T}}_{init}, \mathcal{T}_{final}, k - |I|, S)$, where $\overline{\mathcal{T}}_{init}$ is the triangulation resulting from $\mathcal{T}_{init}$ after all edges in I are flipped and $S = \{(e_1, e_1'), \ldots, (e_l, e_l')\}$ is the set of edge-pairs created in Step 1.

Figure 2 The function **FlipDist-I**.

The algorithm's main function **FlipDist** (Fig. 1) iterates through all subsets of independent diagonals in $\mathcal{T}$. For each non-empty subset I of independent diagonals that represents the set of initial source nodes of a DAG $\mathcal{D}_F$, two mutually recursive functions **FlipDist-I** (Fig. 2) and **FlipDist-S** (Fig. 3) are invoked to repeatedly remove the source nodes and find the set of new source nodes in $\mathcal{D}_F$. Along the way, whenever a free-diagonal is flipped to create a common diagonal, the instance is partitioned into smaller isolated instances.

FlipDist-S($\mathcal{T}_{init}$, $\mathcal{T}_{final}$, k, S)

Input: Two triangulations $\mathcal{T}_{init}$ and $\mathcal{T}_{final}$, a parameter k, and a set of edge-pairs S.
Precondition: $C(\mathcal{T}_{init}, \mathcal{T}_{final}) = 0$
Output: *True* if $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) \leq k$; *False* otherwise.

0. If $\phi(\mathcal{T}_{init}) > k$, return *False*; If $\phi(\mathcal{T}_{init}) = 0$ and $k \geq 0$, return *True*.

1. If there is a free-diagonal e in $\mathcal{T}_{init}$, do:

1.1. Remove any edge-pair containing e from S. Flip the free-diagonal e to create a new diagonal $\overline{e} \in \mathcal{T}_{final}$. Let $\overline{\mathcal{T}}_{init}$ be the triangulation after flipping e.

1.2. Let $\Delta_1 = \{e_1, e_1', \overline{e}\}$ and $\Delta_2 = \{e_2, e_2', \overline{e}\}$ be the triangles on either side of $\overline{e}$ in $\overline{\mathcal{T}}_{init}$. Add the edge-pairs (e_1, e_1') and (e_2, e_2') to S.

1.3. Partition the triangulations $\overline{\mathcal{T}}_{init}$ and $\mathcal{T}_{final}$ along $\overline{e}$ into $\{\overline{\mathcal{T}}^1_{init}, \overline{\mathcal{T}}^2_{init}\}$ and $\{\mathcal{T}^1_{final}, \mathcal{T}^2_{final}\}$, respectively. Partition the edge-pairs S into $\{S_1, S_2\}$ accordingly so that S_1 is in $\overline{\mathcal{T}}^1_{init}$ and S_2 is in $\overline{\mathcal{T}}^2_{init}$. Let $n_1 = \phi(\overline{\mathcal{T}}^1_{init})$ and $n_2 = \phi(\overline{\mathcal{T}}^2_{init})$.

1.4. If the pair $(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final})$ is trivial, set $k_1 = n_1$.

1.5. Else, find the smallest k_1 between $n_1 + 1$ and $\min(k - 1 - n_2,\ 2n_1)$ such that **FlipDist-S**$(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final}, k_1, S_1)$ returns *True*; If such a k_1 does not exist, return *False*.

1.6. If the pair $(\overline{\mathcal{T}}^2_{init}, \mathcal{T}^2_{final})$ is trivial, return *True*.

1.7. Return **FlipDist-S**$(\overline{\mathcal{T}}^2_{init},\ \mathcal{T}^2_{final},\ k - 1 - k_1,\ S_2)$.

2. Iterate through all non-empty subsets I of independent diagonals in $\bigcup S$, as follows:

2.1. For each edge pair $(e_i, e_i') \in S$, branch on up to three choices of their membership in I: (1) include neither e_i nor e_i'; (2) include e_i but not e_i'; and (3) include e_i' but not e_i. Skip a choice if it tries to include a non-diagonal edge or a diagonal with a neighbor already included in I.

2.2. At the end of branching, if I is non-empty, do:
If **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ returns *True*, then return *True*.

3. Return *False*.

Figure 3 The function **FlipDist-S**.

Specifically, **FlipDist-I**($\mathcal{T}_{init}$, $\mathcal{T}_{final}$, k, I) performs flips on all diagonals in I. Suppose that a diagonal $e \in I$ is flipped to create a new edge $\overline{e}$. There are two triangles $\Delta_1 = \{e_1, e_1', \overline{e}\}$ and $\Delta_2 = \{e_2, e_2', \overline{e}\}$ on opposite sides of $\overline{e}$. The edges e_1, e_1', e_2, e_2' are candidates of the new source nodes in $\mathcal{D}_F$. Since (e_1, e_1') are neighbors and so are (e_2, e_2'), at most one edge can be chosen from each pair as a new source node. Therefore, the pairs (e_1, e_1') and (e_2, e_2') are added to a candidate pool S. After all diagonals in I are flipped, the current triangulations are $\overline{\mathcal{T}}_{init}$ and $\mathcal{T}_{final}$, the current parameter is $k - |I|$ since $|I|$ flips are already performed, and the candidate poll is S, all of which are passed to **FlipDist-S** as parameters.

FlipDist-S($\mathcal{T}_{init}$, $\mathcal{T}_{final}$, k, S) first flips free-diagonals (if any) in $\mathcal{T}_{init}$. When a free-diagonal e is flipped, a common diagonal $\overline{e}$ is created, and by Lemma 4, the instance can be safely partitioned along $\overline{e}$ into two smaller isolated sub-instances, which can be solved recursively in a divide-and-conquer approach. In order to determine the parameters of the sub-instances, it is necessary to find the smallest parameter k_1 such that the first sub-instance returns *True*. The parameter of the second sub-instance is then set to be $k - k_1$. If there are no free-diagonals, **FlipDist-S** branches on the pairs of edges in S to form the safe-set I for the next round. For each edge-pair $(e_i, e_i') \in S$ in S, I may include neither, only e_i, or only e_i', forming a three-way branching.

3.3 Analysis of the algorithm

In the following, we will prove the correctness of the algorithm and analyze its running time. We start by proving the following invariant for both **FlipDist-I** and **FlipDist-S**:

▷ Claim 11. For every new diagonal created in **FlipDist-I** and **FlipDist-S**, its neighbors are contained in $\bigcup S$, which is the union of the edge-pairs in S.

Proof. In **FlipDist-I**, after each diagonal $e \in I$ is flipped, the neighbors of the new diagonal $\overline{e}$ are included in the edge-pairs in S. Thus in **FlipDist-I**, $\bigcup S$ contains all neighbors of the new diagonals in $\overline{\mathcal{T}}_{init}$.

FlipDist-S$(\mathcal{T}^i_{init}, \mathcal{T}^i_{final}, k, I)$ starts by flipping free-diagonals (if any) in $\mathcal{T}_{init}$. If there is a free-diagonal e in $\mathcal{T}_{init}$, then by Lemma 4, e can be safely flipped to create a new diagonal $\overline{e} \in \mathcal{T}_{final}$. Before e is flipped to create a new diagonal $\overline{e}$, any pair (e, e') containing e is removed from S. Let $\overline{\mathcal{T}}_{init}$ be the triangulation after flipping e. Let $\Delta_1 = \{e_1, e'_1, \overline{e}\}$ and $\Delta_2 = \{e'_2, e'_2, \overline{e}\}$ be the triangles on either side of $\overline{e}$ in $\overline{\mathcal{T}}_{init}$. Two edge-pairs (e_1, e'_1) and (e_2, e'_2) are added to S. Note that if (e, e') is a pair in S before the free-diagonal e is flipped, then e' must be contained in one of two new pairs added to S after e is flipped. Therefore, after flipping e, $\bigcup S$ still contains all neighbors of the new diagonals in $\overline{\mathcal{T}}_{init}$, and hence the claim is true. ◁

▶ **Lemma 12.** *Let $(\mathcal{T}_{init}, \mathcal{T}_{final})$ be a pair of triangulations that do not share any common edge. Let I be a set of independent diagonals in $\mathcal{T}_{init}$. Let S be a set of edge-pairs in $\mathcal{T}_{init}$ and $\bigcup S$ be the union of the edge-pairs in S.*

(a) *If there is a minimum solution F of $(\mathcal{T}_{init}, \mathcal{T}_{final})$ such that I is the set of underlying diagonals of the source nodes of $\mathcal{D}_F$, then* **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ *returns* True *if and only if F has at most k flips.*

(b) *If there is a minimum solution F of $(\mathcal{T}_{init}, \mathcal{T}_{final})$ such that the set of underlying diagonals of the source nodes of $\mathcal{D}_F$ is a subset of $\bigcup S$, then* **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ *returns* True *if and only if F has at most k flips.*

Proof. The proof is by mutual induction on k. For the base case when $k = 0$, both functions will return true if and only if $\mathcal{T}_{init} = \mathcal{T}_{final}$. The statements are true.

Based on the inductive hypothesis, the inductive step is proven in two parts.

(a) First consider **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$. By Lemma 9, I is a safe-set and hence for any permutation $\pi(I)$ of I, there is a shortest path from $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ such that the diagonals in I are flipped first according to the order of $\pi(I)$. Therefore, $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) \leq k$ if and only of $\text{Dist}(\overline{\mathcal{T}}_{init}, \mathcal{T}_{final}) \leq k - |I|$, where $\overline{\mathcal{T}}_{init}$ is the resulting triangulation after the diagonals in I are flipped. After the set of diagonals I corresponding to the source nodes of $\mathcal{D}_F$ are flipped and removed from $\mathcal{D}_F$, the set of diagonals corresponds to the new source nodes in $\mathcal{D}_F$ must be neighbors of the new diagonals in $\overline{\mathcal{T}}_{init}$ and hence by Claim 11 is a subset of $\bigcup S$. Therefore, when **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ calls **FlipDist-S**$(\overline{\mathcal{T}}_{init}, \mathcal{T}_{final}, k - |I|, S)$, by the inductive hypothesis, it returns *True* if and only if $\text{Dist}(\overline{\mathcal{T}}_{init}, \mathcal{T}_{final}) \leq k - |I|$, as required.

(b) Now consider **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ in two cases:

  (i) If $\mathcal{T}_{init}$ has a free-diagonal e, then by Lemma 4, there is a shortest path from $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ such that e is flipped first to create a new diagonal $\overline{e}$ that is shared by $\overline{\mathcal{T}}_{init}$ and $\mathcal{T}_{final}$. Again by Lemma 4, no shortest paths from $\overline{\mathcal{T}}_{init}$ to $\mathcal{T}_{final}$ will flip $\overline{e}$. Therefore, the triangulations $\overline{\mathcal{T}}_{init}$ and $\mathcal{T}_{final}$ can be safely partitioned along $\overline{e}$ into $\{\overline{\mathcal{T}}^1_{init}, \overline{\mathcal{T}}^2_{init}\}$ and $\{\mathcal{T}^1_{final}, \mathcal{T}^2_{final}\}$, respectively. By Claim 11, after e is flipped, the set S still contains all neighbors of the new diagonals, which is partitioned

accordingly to S_1 and S_2. There is a path of length at most k from $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$ if and only if there is a path of length k_1 from $\mathcal{T}^1_{init}$ to $\mathcal{T}^1_{final}$ and a path of length k_2 from $\mathcal{T}^2_{init}$ to $\mathcal{T}^2_{final}$ such that $k_1 + k_2 = k - 1$. It is easy to see that S_1 and S_2 satisfy the condition of Part (b) for $\overline{\mathcal{T}}^1_{init}$ and $\overline{\mathcal{T}}^2_{init}$, respectively. By the inductive hypothesis, **FlipDist-S**$(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final}, k_1, S_1)$ returns *True* if and only if there is a path of length k_1 from $\mathcal{T}^1_{init}$ to $\mathcal{T}^1_{final}$, and **FlipDist-S**$(\overline{\mathcal{T}}^2_{init},\ \mathcal{T}^2_{final},\ k-1-k_1,\ S_2)$ returns *True* if and only if there is a path of length $k - 1 - k_1$ from $\overline{\mathcal{T}}^2_{init}$ to $\mathcal{T}^2_{final}$. Therefore, **FlipDist-S**$(\overline{\mathcal{T}}_{init}, \mathcal{T}_{final}, k - |I|, S)$ returns *True* if and only if there is a path of length at most k from $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$, as required.

(ii) If $\mathcal{T}_{init}$ has no free-diagonals, then **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ enumerates all subsets I of independent diagonals and calls **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$. For each edge-pair $(e_1, e_2) \in S$, the algorithm branches on up to three possible choices: 1) include neither e_1 nor e_2, 2) include e_1 but not e_2, and 3) include e_2 but not e_1. Since e_1 and e_2 are neighbors, they cannot both belong to I. **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ returns *True* if and only if there is an enumerated independent subset I of $\bigcup S$ such that **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ returns *True*. If $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) \leq k$, then by Claim 11, the set of the underlying diagonals of the source nodes of $\mathcal{D}_F$ is a subset of $\bigcup S$ and hence, will be enumerated. Consequently, by Part (a) of the inductive step proven above, **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ returns *True*. Conversely, if there is an enumerated set I such that **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ returns *True*, then by Part (a) of the inductive step, there is a sequence of at most k flips that transforms $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$. Therefore, **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ returns *True* if and only if $\text{Dist}(\mathcal{T}_{init}, \mathcal{T}_{final}) \leq k$.

This completes the proof. ◀

The following lemma bounds the number of leaves in the search trees of **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ and **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$.

▶ **Lemma 13.** *Let $n = \phi(\mathcal{T}_{init})$. Let $L_I(n, k)$ be the number of leaves in the search tree of* **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$. *Let $L_S(n, k, s)$ be the number of leaves in the search tree of* **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$, *where $s = |S|$. Then $L_I(n, k) \leq 3^{2(k-n)}$ and $L_S(n, k, s) \leq 3^{s+2(k-n)}$.*

Proof. The proof is by mutual induction on k. For the base case when $k = 0$, we have $n = 0$ and both search trees have only 1 leaf. The statements are true.

Based on the inductive hypothesis, the inductive step is proven in two parts.

(a) First consider **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$, which flips $|I|$ edges and create a set S of $2|I|$ edge-pairs. It then calls **FlipDist-S**$(\overline{\mathcal{T}}_{init}, \mathcal{T}_{final}, k - |I|, S)$ where $\phi(\overline{\mathcal{T}}_{init}) = \phi(\mathcal{T}_{init}) = n$. By the inductive hypothesis, the number of leaves in **FlipDist-S**$(\overline{\mathcal{T}}_{init}, \mathcal{T}_{final}, k - |I|, S)$ is at most $3^{|S|+2(k-|I|-n)} = 3^{2|I|+2(k-|I|-n)} = 3^{2(k-n)}$. Since there is no branching in **FlipDist-I**, we have $L_I(n, k) \leq 3^{2(k-n)}$ as required.

(b) Now consider **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ in two cases:

(i) If there is no free-diagonal in $\mathcal{T}_{init}$, then **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ branches into up to 3 subtrees for each pair in S, corresponding to three possible choices that include at most one edge in the pair. Therefore, at most $3^{|S|}$ subtrees are created after all edge-pairs in S are branched on and each subtree is a call to **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$. By Part (a) of the inductive step proven above, **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ has at most $3^{2(k-n)}$ leaves. Therefore, the total number of leaves in the search tree of **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ is at most $3^{|S|}3^{2(k-n)} = 3^{s+2(k-n)}$, as required.

(ii) Suppose that a free-diagonal e is flipped in Step 1 of **FlipDist-S** to create a new diagonal $\overline{e} \in \mathcal{T}_{final}$. The algorithm partitions the triangulations $\overline{\mathcal{T}}_{init}$ and $\mathcal{T}_{final}$ along $\overline{e}$ into $\{\overline{\mathcal{T}}^1_{init}, \overline{\mathcal{T}}^2_{init}\}$ and $\{\mathcal{T}^1_{final}, \mathcal{T}^2_{final}\}$, respectively. This creates two smaller instances $(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final})$ and $(\overline{\mathcal{T}}^2_{init},\ \mathcal{T}^2_{final})$. We may assume both are non-trivial because, by Lemma 7, trivial instances are solvable in linear time. Let $n_1 = \phi(\overline{\mathcal{T}}^1_{init})$. The algorithm calls **FlipDist-S**$(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final}, \hat{k}_1, S_1)$ for $\hat{k}_1 = n_1 + 1, \ldots, \min(k-1-n_2,\ 2n_1)$, until $\hat{k}_1 = k_1$, where $k_1 = \text{Dist}(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final})$. If no such k_1 is found, then the search tree terminates and it is easy to verify that the statement is true. Since $\hat{k}_1 < k$, by the inductive hypothesis, **FlipDist-S**$(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final}, \hat{k}_1, S_1)$ has at most $3^{|S_1|+2(\hat{k}_1-n_1)}$ leaves. This means for $\hat{k}_1 = n_1+1, \ldots, k_1$, the subtrees of **FlipDist-S**$(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final}, \hat{k}_i, S_1)$ has at most $3^{|S_1|+2}$, $3^{|S_1|+4}$, …, $3^{|S_1|+2(k_1-n_1)}$ leaves, respectively. Observing that this is a geometric sequence with a common ratio of 9, their sum is at most $\frac{9}{8} \cdot 3^{|S_1|+2(k_1-n_1)}$.

Finally, the algorithm calls **FlipDist-S**$(\overline{\mathcal{T}}^2_{init},\ \mathcal{T}^2_{final},\ k-1-k_1,\ S_2)$, which by the inductive hypothesis has at most $3^{|S_2|+2(k-1-k_1-n_2)}$ leaves.

Thus, in total, the number of leaves in the search tree of **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ is at most $\frac{9}{8} \cdot 3^{|S_1|+2(k_1-n_1)} + 3^{|S_2|+2(k-1-k_1-n_2)}$. Let $x = 3^{|S_1|+2(k_1-n_1)}$ and $y = 3^{|S_2|+2(k-1-k_1-n_2)}$. Since both instances $(\overline{\mathcal{T}}^1_{init}, \mathcal{T}^1_{final})$ and $(\overline{\mathcal{T}}^2_{init},\ \mathcal{T}^2_{final})$ are non-trivial, we have $k_1 - n_1 \geq 1$ and $k-1-k_1-n_2 \geq 1$, and hence $x, y \geq 9$. Therefore, the number of leaves in the search tree of **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ is at most

$$\begin{aligned}
\frac{9}{8} \cdot 3^{|S_1|+2(k_1-n_1)} + 3^{|S_2|+2(k-1-k_1-n_2)} &= \frac{9}{8}x + y && (1)\\
&= (\frac{27}{8y} + \frac{3}{x}) \cdot \frac{xy}{3} && (2)\\
&\leq (\frac{27}{8 \cdot 9} + \frac{3}{9}) \cdot \frac{xy}{3} && (3)\\
&\leq \frac{xy}{3} && (4)\\
&= \frac{3^{|S_1|+2(k_1-n_1)} \cdot 3^{|S_2|+2(k-1-k_1-n_2)}}{3} && (5)\\
&= 3^{|S_1|+|S_2|-1+2(k-1-n_1-n_2)}. && (6)
\end{aligned}$$

We have $|S_1| + |S_2| \leq |S| + 1$ because when a free-diagonal e is flipped, at least one edge-pair in S that contains e is removed and two new edge-pairs are added to S. We also have $n_1 + n_2 = n - 1$ because after the $\overline{\mathcal{T}}_{init}$ is partitioned along the diagonal $\overline{e}$, the total number of diagonals is reduced by 1.

Therefore, the total number of leaves in the search tree of **FlipDist-S**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, S)$ is $L_S(n,k,s) \leq 3^{|S_1|+|S_2|-1+2(k-1-n_1-n_2)} \leq 3^{|S|}3^{2(k-n)} = 3^{s+2(k-n)}$, as required.

This completes the proof. ◀

When the algorithm initially starts, the set of source nodes may be any subsets I of independent diagonals in $\mathcal{T}_{init}$. We will prove in Lemma 14 that there are at most F_{n+1} such subsets, where F_n is the n-th Fibonacci number, and they can be enumerated using polynomial space and in time $\mathcal{O}(n)$ each.

▶ **Lemma 14.** *Let $\mathcal{T}$ be a triangulation of a convex polygon and $\phi(\mathcal{T}) = n$. The number of subsets of independent diagonals in $\mathcal{T}$ is at most F_{n+1}, the $(n+1)$-th Fibonacci number. Furthermore, all such subsets can be iterated in time $\mathcal{O}(n)$ each using polynomial space.*

Proof. First, observe that the set of subsets of independent diagonals in $\mathcal{T}$ is in bijection with the set of matchings in the binary tree T corresponding to $\mathcal{T}$. Each triangle in $\mathcal{T}$ corresponds to a node in T and each diagonal shared by two triangles in $\mathcal{T}$ corresponds to an edge between the two nodes in T representing the two triangles. A set of independent diagonals in $\mathcal{T}$ corresponds to a subset of edges in T that do not share any endpoint, referred to as a *matching* (we consider an empty set to be a matching). By Lemma 10, the number of matchings in T is at most F_{n+1} and hence the number of subsets of independent diagonals in $\mathcal{T}$ is at most F_{n+1}.

To iterate all such subsets I, consider all diagonals in an arbitrarily fixed order. For each diagonal e, if any of its neighbors has already been included in I, do not include e; otherwise, branch on e to include or exclude e in I. When all diagonals have been branched on, I is a subset of independent diagonals in $\mathcal{T}$. Since each leaf of this branching tree corresponds to a unique subset of independent diagonals in $\mathcal{T}$ and the depth of the branching tree is n, the running time is $\mathcal{O}(n)$ for each subset. The iteration uses polynomial space because only a subset of diagonals in $\mathcal{T}$ is maintained at each step of the branching. ◀

Finally, we have the correctness and complexity of our algorithm.

▶ **Theorem 15.** *The algorithm* **FlipDist**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k)$ *runs in time $\mathcal{O}(3.82^k)$ using polynomial space and returns* True *if and only if there is a sequence of at most k flips that transforms $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$.*

Proof. By Lemma 14, **FlipDist**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k)$ branches into at most F_{n+1} subsets of independent diagonals. For each such subset I, the function **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ is called, which has at most $3^{2(k-n)}$ leaves by Lemma 13. Therefore, the search tree of **FlipDist**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k)$ has at most $F_{n+1}3^{2(k-n)}$ leaves, which means there are at most $F_{n+1}3^{2(k-n)}$ root to leaf paths in the search tree.

Along each root-to-leaf path, the algorithm does the following: (a) enumerates the initial independent set I in time $\mathcal{O}(n)$ when charged to each path; (b) performs at most k flips that take time $\mathcal{O}(1)$ each; (c) performs at most n partitions that take time $\mathcal{O}(n)$ each; (d) perform at most k rounds of branching, where each round takes time $\mathcal{O}(k)$ when charged to each end-branch. Therefore, the time spent on each root-to-leaf path is $O(k^2 + n^2) = O(n^2)$. Since $\mathcal{O}(n^2 F_{n+1}) = \mathcal{O}(1.618^n)$, the overall running time is $\mathcal{O}(n^2 F_{n+1} 3^{2(k-n)}) = \mathcal{O}(1.618^n 3^{2(k-n)}) = \mathcal{O}\left(9^k \cdot \left(\frac{1.618}{9}\right)^n\right)$.

Since $k/2 \leq n \leq k$, the overall running time $\mathcal{O}\left(9^k \cdot \left(\frac{1.618}{9}\right)^n\right)$ is maximized when $n = k/2$. Therefore, the total running time of the algorithm is $\mathcal{O}\left(9^k \cdot \left(\frac{1.618}{9}\right)^n\right) = \mathcal{O}\left(9^k \cdot \left(\frac{1.618}{9}\right)^{k/2}\right) = \mathcal{O}(3.82^k)$.

FlipDist$(\mathcal{T}_{init}, \mathcal{T}_{final}, k)$ enumerates all subsets I of independent diagonals as the initial set of source nodes and calls **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$. By Lemma 12, **FlipDist-I**$(\mathcal{T}_{init}, \mathcal{T}_{final}, k, I)$ returns *True* if and only if there is a sequence of at most k flips that transforms $\mathcal{T}_{init}$ to $\mathcal{T}_{final}$.

Finally, the algorithm uses polynomial space because every step of it, including the iteration of the initial independent sets (Lemma 14), uses polynomial space.

This proves the correctness and complexity of our algorithm. ◀

4 Concluding remarks

Both Convex Flip Distance and General Flip Distance are important problems. The current paper presents a simple $\mathcal{FPT}$ algorithm for Convex Flip Distance that runs in time $\mathcal{O}(3.82^k)$ and uses polynomial space, significantly improving the previous best $\mathcal{FPT}$ algorithm the problem, which runs in time $\mathcal{O}(n+k\cdot 32^k)$ and is the same $\mathcal{FPT}$ algorithm for General Flip Distance [11].

Our algorithm takes advantage of the structural properties of Convex Flip Distance regarding the common diagonals and the free-diagonals. However, the general approach of our algorithm, namely finding a topological sort of the DAG $\mathcal{D}_F$ by repeatedly removing the source nodes, seems applicable to General Flip Distance. It remains to be seen if can be used to derive an improved algorithm for General Flip Distance.

The recent progress on both Convex Flip Distance and General Flip Distance, including [16, 11] and this work, all rely on the DAG that models dependency relation among the flips. More research along this line will likely produce further improved algorithms. However, deciding whether the Convex Flip Distance problem is $\mathcal{NP}$-hard remains a challenging open problem and may require new insights into the structural properties of the problem.

References

1. O. Aichholzer, F. Hurtado, and M. Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry: Theory and Applications*, 29(2):135–145, 2004.
2. O. Aichholzer, W. Mulzer, and A. Pilz. Flip distance between triangulations of a simple polygon is NP-complete. *Discrete & Computational Geometry*, 54(2):368–389, 2015.
3. A. Bonnin and J.-M. Pallo. A shortest path metric on unlabeled binary trees. *Pattern Recognition Letters*, 13(6):411–415, 1992.
4. P. Bose and F. Hurtado. Flips in planar graphs. *Computational Geometry: Theory and Applications*, 42(1):60–80, 2009.
5. M. B. Calvo and S. Kelk. An improved kernel for the flip distance problem on simple convex polygons. In Meng He and Don Sheehy, editors, *Proceedings of the 33rd Canadian Conference on Computational Geometry, CCCG 2021*, pages 195–199, 2021.
6. Y.-J. Chen, J.-M. Chang, and Y.-L. Wang. An efficient algorithm for estimating rotation distance between two binary trees. *International Journal of Computer Mathematics*, 82:1095–1106, 2005.
7. S. Cleary and K. St. John. Rotation distance is fixed-parameter tractable. *Information Processing Letters*, 109(16):918–922, 2009.
8. S. Cleary and K. St. John. A linear-time approximation algorithm for rotation distance. *J. Graph Algorithms Appl.*, 14:385–390, 2010.
9. K. Culik and D. Wood. A note on some tree similarity measures. *Information Processing Letters*, 15(1):39–42, 1982.
10. R. Downey and M. Fellows. *Parameterized Complexity*. Springer, New York, 1999.
11. Q. Feng, S. Li, X. Meng, and J. Wang. An improved fpt algorithm for the flip distance problem. *Information and Computation*, 281, 2021.
12. S. Fordham and S. Cleary. Minimal length elements of thompson's groups f(p). *Geometriae Dedicata*, 141:163–180, 2007.
13. S. Hanke, T. Ottmann, and S. Schuierer. The edge-flipping distance of triangulations. *Journal of Universal Computer Science*, 2(8):570–579, 1996.
14. F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.

15 A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, November 1962.

16 I. Kanj, E. Sedgwick, and G. Xia. Computing the flip distance between triangulations. *Discret. Comput. Geom.*, 58(2):313–344, 2017.

17 I. Kanj and G. Xia. Flip Distance Is in FPT Time $O(n + k \cdot c^k)$. In *proceedings of STACS*, volume 30 of *LIPIcs*, pages 500–512, 2015.

18 C. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

19 M. Li and L. Zhang. Better approximation of diagonal-flip transformation and rotation transformation. In *Proceedings of the 4th Annual International Conference on Computing and Combinatorics*, COCOON '98, pages 85–94, 1998.

20 A. Lubiw and V. Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry: Theory and Applications*, 49:17–23, 2015.

21 J. Lucas. An improved kernel size for rotation distance in binary trees. *Information Processing Letters*, 110(12):481–484, 2010.

22 F. Luccio and L. Pagli. On the upper bound on the rotation distance of binary trees. *Information Processing Letters*, 31(2):57–60, 1989.

23 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, USA, 2006.

24 J. Pallo. On the rotation distance in the lattice of binary trees. *Information Processing Letters*, 25(6):369–373, 1987.

25 A. Pilz. Flip distance between triangulations of a planar point set is APX-hard. *Computational Geometry: Theory and Applications*, 47(5):589–604, 2014.

26 L. Pournin. The diameter of associahedra. *Advances in Mathematics*, 259:13–42, 2014.

27 D. Sleator, R. Tarjan, and W. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *Journal of the American Mathematical Society*, 1:647–681, 1988.

28 R. P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, 1999.

Tight Bounds for Repeated Balls-Into-Bins

Dimitrios Los
Department of Computer Science & Technology, University of Cambridge, UK

Thomas Sauerwald
Department of Computer Science & Technology, University of Cambridge, UK

Abstract

We study the *repeated balls-into-bins* process introduced by Becchetti, Clementi, Natale, Pasquale and Posta [3]. This process starts with m balls arbitrarily distributed across n bins. At each round $t = 1, 2, \ldots$, one ball is selected from each non-empty bin, and then placed it into a bin chosen independently and uniformly at random. We prove the following results:

- For any $n \leqslant m \leqslant \text{poly}(n)$, we prove a lower bound of $\Omega(m/n \cdot \log n)$ on the maximum load. For the special case $m = n$, this matches the upper bound of $\mathcal{O}(\log n)$, as shown in [3]. It also provides a positive answer to the conjecture in [3] that for $m = n$ the maximum load is $\omega(\log n / \log\log n)$ at least once in a polynomially large time interval. For $m \in [\omega(n), n \log n]$, our new lower bound disproves the conjecture in [3] that the maximum load remains $\mathcal{O}(\log n)$.
- For any $n \leqslant m \leqslant \text{poly}(n)$, we prove an upper bound of $\mathcal{O}(m/n \cdot \log n)$ on the maximum load for all steps of a polynomially large time interval. This matches our lower bound up to multiplicative constants.
- For any $m \geqslant n$, our analysis also implies an $\mathcal{O}(m^2/n)$ waiting time to reach a configuration with a $\mathcal{O}(m/n \cdot \log m)$ maximum load, even for worst-case initial distributions.
- For $m \geqslant n$, we show that every ball visits every bin in $\mathcal{O}(m \log m)$ rounds. For $m = n$, this improves the previous upper bound of $\mathcal{O}(n \log^2 n)$ in [3]. We also prove that the upper bound is tight up to multiplicative constants for any $n \leqslant m \leqslant \text{poly}(n)$.

2012 ACM Subject Classification Mathematics of computing → Probability and statistics; Mathematics of computing → Discrete mathematics; Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Design and analysis of algorithms

Keywords and phrases Repeated balls-into-bins, self-stabilizing systems, balanced allocations, potential functions, random walks

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.45

Related Version *Full Version*: `https://arxiv.org/abs/2203.12400` [25]
Brief Announcement (SPAA'22): `https://doi.org/10.1145/3490148.3538561` [24]

Supplementary Material *Software*: `https://github.com/Dim131/RBB`
archived at `swh:1:dir:106d475ab210c4248cebfdee217a0a7f40fbfcab`

1 Introduction

We consider the allocation processes involving m balls (jobs or data items) to n bins (servers or memory cells), by allowing each ball to choose from a set of randomly chosen bins. The goal is to allocate (or re-allocate) balls efficiently, while also keeping the load distribution balanced. The balls-into-bins framework has found numerous applications in hashing, load balancing, routing (we refer to the surveys [27] and [32] for more details).

A classical sequential allocation algorithm is the d-Choice process introduced by Azar, Broder, Karlin and Upfal [1] and Karp, Richard, Luby, and Meyer auf der Heide [20], where for each ball to be allocated, we sample $d \geqslant 1$ bins uniformly and then place the ball in the least loaded of the d sampled bins. It is well-known that for the One-Choice process ($d = 1$), the maximum load is w.h.p.[1] $\Theta(\log n / \log\log n)$ for $m = n$ and $m/n + \Theta\big(\sqrt{m/n \cdot \log n}\big)$

[1] In general, with high probability refers to probability of at least $1 - n^{-c}$ for some constant $c > 0$.

© Dimitrios Los and Thomas Sauerwald;
licensed under Creative Commons License CC-BY 4.0

for $m = \Omega(n \log n)$. In particular, this gap between maximum and average load grows significantly as $m/n \to \infty$, which is called the *heavily loaded case*. For $d = 2$, [1] proved that the maximum load is only $m/n + \log_2 \log n + \mathcal{O}(1)$ for $m = n$. This result was generalized by Berenbrink, Czumaj, Steger and Vöcking [6] who proved that the same guarantee also holds for $m \geqslant n$, in other words, even as $m/n \to \infty$, the difference between the maximum and average load remains a slowly growing function in n that is independent of m. This improvement of Two-Choice over One-Choice has been widely known as the "power of two choices".

In this work, we investigate the *repeated balls-into-bins (RBB)* process, introduced by Becchetti, Clementi, Natale, Pasquale and Posta [3]. In this process, there are m balls initially allocated arbitrarily across n bins. In each round, one ball is removed from each non-empty bin and then each of these balls is allocated to one bin sampled uniformly at random (see Figure 1). This setting differs from the classical balls into bins setting in that the number of balls is fixed and the amount of balls we re-allocate in each round varies from 1 to n. Unlike Two-Choice (or d-Choice), this re-allocation is performed without inspecting the load of any bin or taking additional samples.

Becchetti et al. [3] proved that for $m = n$, starting from an arbitrary configuration, w.h.p. after $\mathcal{O}(n)$ rounds, the process reaches a maximum load of $\mathcal{O}(\log n)$ and remains in such a configuration for $\text{poly}(n)$ rounds. Thus, the RBB process is a natural instance of a self-stabilizing system, and falls into a long line of research on random-walk based algorithms for stabilization and consensus [4, 14, 17, 18, 29]. More recently, Cancrini and Posta [11] proved that the mixing time is $\mathcal{O}(L)$ where L is the maximum load at the initial configuration.

Our Results. In this work, we settle two conjectures stated in [3] and prove tight bounds for the more general case with $n \leqslant m \leqslant \text{poly}(n)$.

Becchetti et al. [3] conjectured that the $\mathcal{O}(\log n)$ upper bound holds for all $m = \mathcal{O}(n \log n)$. They also conjectured that for $m = n$, the maximum load is $\omega(\log n / \log \log n)$. We resolve both conjectures, proving an $\Omega(\frac{m}{n} \cdot \log n)$ lower bound on the maximum load w.h.p. in any interval of length $\Omega(m^2/n^2 \cdot \log^4 n)$ and for any $n \leqslant m \leqslant \text{poly}(n)$ (Lemma 3.3). This disproves the first conjecture, but confirms the second one, showing that for $m = n$, the maximum load is w.h.p. $\Theta(\log n)$.

For the case $m \geqslant n$, we also prove that starting from an arbitrary configuration after $\mathcal{O}(m^2/n)$ rounds, w.h.p. we reach a configuration with a maximum load of $\mathcal{O}(\frac{m}{n} \cdot \log m)$ (Section 4.2). For $n \leqslant m \leqslant \text{poly}(n)$, we show that the process stabilizes in such a configuration there for at least m^2 rounds (Theorem 4.11).

Becchetti et al. [3] also studied the *cover time* (or *traversal time*) of a ball, which is the time required to visit all n bins. For $m = n$, they proved an $\mathcal{O}(n \log^2 n)$ bound on the traversal time. For any $m \geqslant n$, we improve this to $\mathcal{O}(n \log m)$, and also show that it is tight up to constant factors for any $m = \text{poly}(n)$ (Section 5).

Intuition and Techniques. For the upper bound we use an exponential potential Φ with smoothing parameter $\Theta(n/m)$. Provided that Φ is $\text{poly}(m)$, we immediately obtain the $\mathcal{O}(m/n \cdot \log m)$ bound on the maximum load. Our analysis exploits that after only $\mathcal{O}((m/n)^2)$ rounds, sufficiently many bins will become empty, which in turn will reduce the number of balls being re-allocated. This then helps to reduce the load of any non-empty bin, since these are guaranteed to lose one ball per round, but only receive in expectation less than one ball in total from the other non-empty bins. As we will prove, the actual equilibrium will have

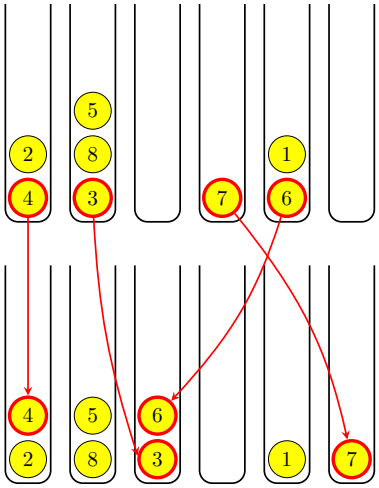


Figure 1 Illustration of one round of RBB with $m = 8$ balls and $n = 6$ bins. The balls highlighted in red are re-allocated to bins chosen randomly among $\{1, 2, \ldots, 6\}$.

most bins being empty roughly every $\mathcal{O}(m/n)$ rounds. To establish this, we employ some martingale and drift-arguments to first prove that any bin which starts at load $\mathcal{O}(m/n)$, becomes empty after $\mathcal{O}((m/n)^2)$ rounds with constant probability > 0. Secondly, we prove that if this happens to a fixed bin, the empty load state will be revisited $\Omega(m/n)$ times during the next $\mathcal{O}((m/n)^2)$ rounds. In some sense, this is a generalization of the approach in [3], where they also bounded the fraction of empty bins for the case $m = n$.

A kind of reversed argument is used for the lower bound. Here, the goal is to prove that each bin is only empty every $\mathcal{O}(m/n)$ rounds on average. This shows that the RBB process can be approximated by a One-Choice process where at least an $1 - \mathcal{O}(n/m)$ fraction of the balls are allocated. For $t = \Omega(m^2/n^2 \cdot \log n)$, this yields a maximum load of $\Omega(m/n \cdot \log n)$. To prove that bins are not empty "too often", we establish a link between a quadratic potential and the number of empty bins, similar to that in [26, Lemma 6.2]. This connection essentially implies that whenever the fraction of empty bins is $\omega(n/m)$, then the quadratic potential decreases. By aggregating sufficiently over many rounds, we can conclude that, on average, the number of empty bins cannot be too large.

Further Related Work. Cancrini and Posta investigated the behavior of the RBB process for a large number of rounds, and established "propagation of chaos" [10], meaning that under some conditions on the initial load distribution, the load of the bins become eventually independent. In [10], the authors prove results for the RBB process considered here, while [12] considered more general re-allocation rules. Another variant of the RBB setting was studied in [8], where in each round one ball is deleted from each bin and an expected λn new balls arrive and are distributed in parallel to the bins. In contrast to the RBB model, this means that the number of balls in the system is not fixed.

The RBB is an instance of a discrete time closed Jackson network [19, 21]. However, in RBB, updates are happening synchronously and in parallel, while in most queuing models updates occur asynchronously based on independent point processes. As also pointed out in [10, 12], this leads to a non-reversible Markov Chain, which seems to make the computation of the stationary distribution intractable. Furthermore, formal methods have been used to prove guarantees for RBB with $m = n$ [2]. The RBB setting has also been applied to analyze protocols in short packet communications [33].

Czumaj, Riley and Scheideler [15] studied a similar re-allocation process where in each round one random ball is allocated to a random of d bin choices. These are also related to randomized rerouting protocols studied in [7, 9]. In another parallel allocation processes, Berenbrink, Czumaj, Englert, Friedetzky and Nagel [5] proved an $\mathcal{O}(\log n)$ gap for the Two-Choice process where balls are allocated in batches of n balls and was recently improved to $\mathcal{O}(\frac{\log n}{\log\log n})$ in [23].

Organization. In Section 2 we introduce some standard balls-into-bins notations and define the processes. In Section 3, we prove our lower bound on the maximum load. In Section 4, we prove an upper bound on maximum load and also analyze the time until such configuration is reached and preserved (convergence time). In Section 5, we analyze the traversal time. In Section 6, we present some empirical results on the RBB process. We conclude the paper with a summary and a few open problems in Section 7.

2 Notation and Definitions

We consider a set of n bins labeled $[n] := \{1, 2, \ldots, n\}$. By x^t we denote the n-dimensional *load vector* after t rounds, and x^0 is the initial load vector. In our processes, no balls are added or removed, and the existing m balls are only re-allocated; hence, $\sum_{i=1}^n x_i^t = m$ for all $t \geqslant 0$.

By $F^t := |\{i \in [n] \colon x_i^t = 0\}|$ we denote the number of *empty (free) bins* and by $f^t := \frac{1}{n} \cdot F^t$ the fraction of empty bins. Similarly $\kappa^t := n - F^t$ is the number of *non-empty bins*. Since it will be important to track the number of empty bins over a time interval, we also define $F_{t_0}^{t_1}$ as the total number of pairs of empty bins and rounds in the entire interval $[t_0, t_1]$, i.e.,

$$F_{t_0}^{t_1} := \sum_{t=t_0}^{t_1} F^t.$$

RBB (Repeated Balls-into-Bins Process):
Iteration: At each round $t = 1, 2, \ldots$
- For each of the $\kappa^t = n - F^t$ non-empty bins, take one ball and re-allocate it to a bin chosen independently and uniformly at random among $[n]$.

More specifically, in each round we choose κ^t bins $z_1^t, \ldots, z_{\kappa^t}^t \in [n]$ uniformly at random and the load vector at step $t+1$ is given by

$$x_i^{t+1} := x_i^t - \mathbf{1}_{x_i^t > 0} + \sum_{j=1}^{\kappa^t} \mathbf{1}_{z_j^t = i}, \quad \text{for each } i \in [n].$$

Hence, we can express the marginal load distribution of an arbitrary bin $i \in [n]$ at round $t \geqslant 0$ (i.e., having completed t iterations before), as

$$x_i^{t+1} = x_i^t - \mathbf{1}_{x_i^t > 0} + \mathsf{Bin}(\kappa^t, 1/n), \tag{2.1}$$

where with slight abuse of notation, we write $\mathsf{Bin}(\kappa^t, 1/n)$ as a placeholder for a random variable (independent of $\mathfrak{F}^t$, the entire history of the process up to round t) which has distribution $\mathsf{Bin}(\kappa^t, 1/n)$.

Similarly, assuming each bin acts as a FIFO queue on the incoming and departing balls, we can follow the trajectory of an arbitrary single ball. Only if the ball is at the front of its queue, it will be re-allocated to a bin chosen randomly from $[n]$ in the next round. A natural question is the so-called *cover time* (or *traversal time*), the expected time until every ball has been allocated to each bin [3]. This is related to the well-studied *cover time* of parallel random walks on graphs, but with the constraint that only one walk can leave each vertex (=bin) at a time.

3 Lower Bound on the Maximum Load for $n \leqslant m \leqslant \mathrm{poly}(n)$

In this section, we prove that w.h.p. the maximum load becomes $\Omega\left(\frac{m}{n} \cdot \log n\right)$ at least once in every $\mathcal{O}(\frac{m^2}{n^2} \cdot \log^4 n)$ rounds, for any $n \leqslant m \leqslant \mathrm{poly}(n)$. This matches the upper bound in Section 4 up to multiplicative constants and also settles two conjectures in [3].

On a high level, the lower bound follows by showing that in a long enough interval, w.h.p. a constant fraction of the rounds have an $\mathcal{O}(n/m)$ fraction of empty bins. Then, we couple the process with the One-Choice process, to show that the maximum load must be w.h.p. at least $\Omega(\frac{m}{n} \cdot \log n)$.

In order to bound the number of empty bins in an interval we make use of the *quadratic potential function*, defined as

$$\Upsilon^t := \sum_{i=1}^{n} (x_i^t)^2,$$

where x_i^t is the load of bin $i \in [n]$ at round t. We then prove the following relation between the expected change of Υ^t and the number of empty bins F^t in round t:

▶ **Lemma 3.1.** *Consider the RBB setting with any $m \geqslant 1$. Then, for any round $t \geqslant 0$,*

$$\mathbf{E}\left[\, \Upsilon^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] \leqslant \Upsilon^t - 2 \cdot \frac{m}{n} \cdot F^t + 2n.$$

Proof. Let us define the binomial random variable $Z \sim \mathrm{Bin}(\kappa^t, \frac{1}{n})$. For any bin $i \in [n]$ with load $x_i^t \geqslant 1$,

$$\begin{aligned}
\mathbf{E}\left[\, \Upsilon_i^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] &= \sum_{z=0}^{\kappa^t} (x_i^t + z - 1)^2 \cdot \binom{\kappa^t}{z} \cdot \frac{1}{n^z} \cdot \left(1 - \frac{1}{n}\right)^{\kappa^t - z} \\
&= (x_i^t)^2 \cdot \sum_{z=0}^{\kappa^t} \binom{\kappa^t}{z} \cdot \frac{1}{n^z} \cdot \left(1 - \frac{1}{n}\right)^{\kappa^t - z} \\
&\quad + 2 \cdot x_i^t \cdot \sum_{z=0}^{\kappa^t} (z-1) \cdot \binom{\kappa^t}{z} \cdot \frac{1}{n^z} \cdot \left(1 - \frac{1}{n}\right)^{\kappa^t - z} \\
&\quad + \sum_{z=0}^{\kappa^t} (z-1)^2 \cdot \binom{\kappa^t}{z} \cdot \frac{1}{n^z} \cdot \left(1 - \frac{1}{n}\right)^{\kappa^t - z} \\
&= (x_i^t)^2 \cdot \mathbf{E}[\, Z \,] + 2 \cdot x_i^t \cdot \mathbf{E}[\, Z - 1 \,] + \mathbf{E}\left[\, (Z-1)^2 \,\right] \\
&\overset{(a)}{=} (x_i^t)^2 + 2 \cdot x_i^t \cdot \left(\frac{\kappa^t}{n} - 1\right) + \kappa^t \cdot (\kappa^t - 1) \cdot \frac{1}{n^2} - \frac{\kappa^t}{n} + 1 \\
&\leqslant (x_i^t)^2 + 2 \cdot x_i^t \cdot \left(\frac{\kappa^t}{n} - 1\right) + 2,
\end{aligned}$$

having used in (a) that $\mathbf{E}\left[\, Z \,\right] = \frac{\kappa^t}{n}$ and $\mathbf{E}\left[\, Z^2 \,\right] = \kappa^t \cdot \frac{1}{n} \cdot (1 - \frac{1}{n}) + (\kappa^t)^2 \cdot (\frac{1}{n})^2$, and thus

$$\mathbf{E}\left[\, (Z-1)^2 \,\right] = \kappa^t \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right) + (\kappa^t)^2 \cdot \left(\frac{1}{n}\right)^2 - 2 \cdot \frac{\kappa^t}{n} + 1 = \kappa^t \cdot (\kappa^t - 1) \cdot \frac{1}{n^2} - \frac{\kappa^t}{n} + 1.$$

Similarly for an empty bin $i \in [n]$ with $x_i^t = 0$, the contribution is

$$\mathbf{E}\left[\, \Upsilon_i^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] = \sum_{z=0}^{\kappa^t} z^2 \cdot \binom{\kappa^t}{z} \cdot \frac{1}{n^z} \cdot \left(1 - \frac{1}{n}\right)^{\kappa^t - z} = \frac{\kappa^t}{n} + \frac{\kappa^t \cdot (\kappa^t - 1)}{n^2}.$$

Hence, by aggregating the contributions of the κ^t bins non-empty bins and the $n - \kappa^t$ empty bins we obtain

$$\begin{aligned}
\mathbf{E}\left[\, \Upsilon^{t+1} \mid \mathfrak{F}^t \,\right] &\leqslant \Upsilon^t + \sum_{i \in [n]\colon x_i^t \geqslant 1} \left(2 \cdot x_i^t \cdot \left(\frac{\kappa^t}{n} - 1\right) + 2\right) + \sum_{i \in [n]\colon x_i^t = 0} \left(\frac{\kappa^t}{n} + \frac{\kappa^t \cdot (\kappa^t - 1)}{n^2}\right) \\
&\leqslant \Upsilon^t + \left(\frac{\kappa^t}{n} - 1\right) \cdot 2 \cdot m + 2\kappa^t + (n - \kappa^t) \cdot 2 \\
&= \Upsilon^t - 2 \cdot \frac{m}{n} \cdot F^t + 2n,
\end{aligned}$$

where in the last inequality we used that $\kappa^t \leqslant n$. This concludes the proof. ◀

The key insight is that the quadratic potential drops in expectation as soon as the fraction of empty bins is of order $\Omega(n/m)$. This is crucial to upper bound the number of empty bins in an interval. This relation is similar to the ones used in [23, 26], where an interplay between the quadratic potential and the absolute value potential was used to show that the absolute value potential is small in a constant fraction of the rounds.

The next lemma shows that for any sufficiently long interval, either there is a maximum load that is $\Omega(m/n \cdot \log n)$ or the fraction of empty bins in the interval is $\mathcal{O}(n/m)$. Note that we indeed need the interval to be long enough as starting with the perfectly balanced load vector may require several rounds to reach a gap of $\Omega(\frac{m}{n} \log n)$ even for the One-Choice process.

▶ **Lemma 3.2.** *Consider the RBB process with any $n \leqslant m \leqslant n^k$ for some constant $k \geqslant 1$ and any $1 \leqslant \hat{c} \leqslant n$. Then, for any $t_0 \geqslant 0$ and $t_1 := t_0 + \hat{c} \cdot \left(\frac{m}{n} \cdot \log n\right)^2$,*

$$\mathbf{Pr}\left[\left\{ F_{t_0}^{t_1} < \frac{n^2}{4m} \cdot (t_1 - t_0 + 1) \right\} \cup \bigcup_{t \in [t_0, t_1]} \left\{ \max_{i \in [n]} x_i^t > \frac{m}{n} \cdot \log n \right\} \,\middle|\, \mathfrak{F}^{t_0} \right] \geqslant 1 - e^{-\frac{\hat{c}}{18}}.$$

Proof. Consider an arbitrary step t_0 and filtration $\mathfrak{F}^{t_0}$. We can further assume that $\{\max_{i \in [n]} x_i^{t_0} \leqslant \frac{m}{n} \cdot \log n\}$ holds, otherwise the conclusion trivially follows.

For any $t \geqslant t_0$, we define the sequence

$$Z^t := \Upsilon^t - 2 \cdot (t - t_0) \cdot n + 2 \cdot \frac{m}{n} \cdot F_{t_0}^{t-1},$$

where $F_{t_0}^{t_0 - 1} = 0$. This sequence forms a super-martingale since by Lemma 3.1,

$$\begin{aligned}
\mathbf{E}\left[\, Z^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] &= \mathbf{E}\left[\, \Upsilon^{t+1} - 2 \cdot (t - t_0 + 1) \cdot n + 2 \cdot \frac{m}{n} \cdot F_{t_0}^t \,\middle|\, \mathfrak{F}^t \,\right] \\
&= \mathbf{E}\left[\, \Upsilon^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] - 2 \cdot (t - t_0 + 1) \cdot n + 2 \cdot \frac{m}{n} \cdot F_{t_0}^t \\
&\leqslant \Upsilon^t + 2 \cdot n - 2 \cdot \frac{m}{n} \cdot F^t - 2 \cdot (t - t_0 + 1) \cdot n + 2 \cdot \frac{m}{n} \cdot F_{t_0}^t
\end{aligned}$$

$$= \Upsilon^t - 2 \cdot (t - t_0) \cdot n + 2 \cdot \frac{m}{n} \cdot F_{t_0}^{t-1}$$
$$= Z^t.$$

Further, let $\tau := \min\{t \geqslant t_0 \colon \max_{i \in [n]} x_i^t > \frac{m}{n} \cdot \log n\}$ and consider the stopped random variable

$$\widetilde{Z}^t := Z^{t \wedge \tau},$$

which is then also a super-martingale.

To prove concentration of $\widetilde{Z}^t$, we will now derive an upper bound on $\left|\widetilde{Z}^{t+1} - \widetilde{Z}^t\right|$ conditional on $\mathfrak{F}^t$.

Case 1: $t \geqslant \tau$**.** In this case, $\widetilde{Z}^{t+1} = Z^{(t+1) \wedge \tau} = Z^\tau$, and similarly, $\widetilde{Z}^t = Z^{t \wedge \tau} = Z^\tau$, so $|\widetilde{Z}^{t+1} - \widetilde{Z}^t| = 0$.

Case 2: $t < \tau$**.** Hence for t we have $\max_{i \in [n]} x_i^t \leqslant \frac{m}{n} \cdot \log n$ and thus Lemma A.2 implies that the biggest change in the quadratic potential is w.h.p. at most $2 \cdot m \cdot \log n + 4n$ and under this condition, using that $m \geqslant n$,

$$|\widetilde{Z}^{t+1} - \widetilde{Z}^t| \leqslant 2 \cdot m \cdot \log n + 4n + 2 \cdot \frac{m}{n} \cdot n \leqslant 3 \cdot m \cdot \log n.$$

Combining the two cases above, we conclude,

$$\mathbf{Pr}\left[\bigcap_{t \in [t_0, t_1 - 1]} \left\{|\widetilde{Z}^{t+1} - \widetilde{Z}^t| \leqslant 3 \cdot m \cdot \log n\right\}\right] \geqslant 1 - n^{-\omega(1)} \cdot (t_1 - t_0) \geqslant 1 - n^{-\omega(1)},$$

since $t_1 - t_0 \leqslant \mathrm{poly}(n)$.

Using the concentration inequality Theorem A.4 with bad event, $\mathcal{B}^t := \neg \bigcap_{t \in [t_0, t]} \{|\widetilde{Z}^{t+1} - \widetilde{Z}^t| \leqslant 3 \cdot m \cdot \log n\}$ and $\lambda = \hat{c} \cdot \frac{m^2}{n} \cdot \log^2 n$, we get

$$\begin{aligned}
\mathbf{Pr}\left[\widetilde{Z}^{t_1+1} - \widetilde{Z}^{t_0} > \lambda\right] &\leqslant \exp\left(-\frac{\lambda^2}{2 \cdot \sum_{t=t_0}^{t_1} (3 \cdot m \cdot \log n)^2}\right) + \mathbf{Pr}\left[\mathcal{B}\right] \\
&= \exp\left(-\frac{\hat{c}^2 \cdot \left(\frac{m^2}{n} \cdot \log^2 n\right)^2}{18 \cdot \hat{c} \cdot \left(\frac{m}{n} \cdot \log n\right)^2 \cdot (m \cdot \log n)^2}\right) + \mathbf{Pr}\left[\mathcal{B}\right] \\
&\leqslant e^{-\frac{\hat{c}}{18}} + n^{-\omega(1)} \leqslant 2 \cdot e^{-\frac{\hat{c}}{18}}.
\end{aligned}$$

Thus,

$$\mathbf{Pr}\left[\left\{Z^{t_1+1} \leqslant Z^{t_0} + \lambda\right\} \cup \bigcup_{t \in [t_0, t_1]} \left\{\max_{i \in [n]} x_i^t \geqslant \frac{m}{n} \cdot \log n\right\}\right] < 1 - 2 \cdot e^{-\frac{\hat{c}}{18}}.$$

Assume that $\{Z^{t_1+1} \leqslant Z^{t_0} + \lambda\}$ holds. Our aim is to show that $\{F_{t_0}^{t_1} < \frac{4n^2}{m} \cdot (t_1 - t_0 + 1)\}$ also holds. For the sake of a contradiction, assume that

$$F_{t_0}^{t_1} \geqslant \frac{4n^2}{m} \cdot (t_1 - t_0 + 1).$$

By $\{Z^{t_1+1} \leqslant Z^{t_0} + \lambda\}$, we have that

$$\Upsilon^{t_1+1} - 2 \cdot (t_1 - t_0 + 1) \cdot n + 2 \cdot \frac{m}{n} \cdot F_{t_0}^{t_1} \leqslant \Upsilon^{t_0} + \lambda.$$

Rearranging the inequality above gives

$$\begin{aligned}\Upsilon^{t_1+1} &\leqslant \Upsilon^{t_0} + \lambda + 2\cdot(t_1-t_0+1)\cdot n - 2\cdot\frac{m}{n}\cdot F_{t_0}^{t_1}\\ &\leqslant \Upsilon^{t_0} + \lambda + 2\cdot(t_1-t_0+1)\cdot n - 8\cdot n\cdot(t_1-t_0+1)\\ &\leqslant \Upsilon^{t_0} + \lambda - 6\cdot(t_1-t_0+1)\cdot n. \end{aligned} \tag{3.1}$$

Recall that we start from a round t_0 where $\{\max_{i\in[n]} x_i^{t_0} \leqslant \frac{m}{n}\cdot\log n\}$ holds, and therefore also $\{\Upsilon^{t_0} \leqslant n\cdot\left(\frac{m}{n}\cdot\log n\right)^2\}$ holds. Thus, by (3.1) we have

$$\Upsilon^{t_1+1} \leqslant n\cdot\left(\frac{m}{n}\cdot\log n\right)^2 + \hat{c}\cdot n\cdot\left(\frac{m}{n}\cdot\log n\right)^2 - 6\cdot\hat{c}\cdot\left(\frac{m}{n}\cdot\log n\right)^2\cdot n < 0$$

which is a contradiction for large n since $\hat{c} \geqslant 1$. We conclude that if $Z^{t_1+1} \leqslant Z^{t_0}+\lambda$, then $F_{t_0}^{t_1} < \frac{n^2}{4m}\cdot(t_1-t_0+1)$ or the stopping time was reached, i.e.

$$\mathbf{Pr}\left[\left\{F_{t_0}^{t_1} < \frac{n^2}{4m}\cdot(t_1-t_0+1)\right\} \cup \bigcup_{t\in[t_0,t_1]}\left\{\max_{i\in[n]} x_i^t \geqslant \frac{m}{n}\cdot\log n\right\}\right] \geqslant 1-2\cdot e^{-\frac{\hat{c}}{18}}. \qquad \blacktriangleleft$$

To complete the derivation of the lower bound we need to show that in an interval of length $T = \Theta((\frac{m}{n}\cdot\log n)^2)$ with an $\mathcal{O}(n/m)$ fraction of empty bins, the maximum load is $\Omega(\frac{m}{n}\cdot\log n)$. This follows by coupling the allocations of the RBB process in the interval with a One-Choice process with $T\cdot(1-\mathcal{O}(n/m))$ balls. By the following standard expression, for the maximum load, setting $c := \frac{(1-\gamma)^2}{200}\cdot\frac{1}{\gamma^2}$ (for $\gamma = \Theta(\frac{n}{m})$), we get the desired lower bound on the maximum load for the RBB setting.

[cf. [26, Lemma 10.4]] Consider the One-Choice process with $m = cn\log n$ balls, for any $c \geqslant 1/\log n$. Then, we have

$$\mathbf{Pr}\left[\max_{i\in[n]} x_i^m \geqslant \left(c + \frac{\sqrt{c}}{10}\right)\cdot\log n\right] \geqslant 1-n^{-2}.$$

Putting the lemmas together, we get the desired lower bound.

▶ **Lemma 3.3.** *Consider the RBB process with any $n \leqslant m \leqslant n^k$ for some constant $k \geqslant 1$ and let $\gamma := \frac{n}{4m}$. Then, for any round $t_0 \geqslant 0$ and for $t_1 := t_0 + \frac{1-\gamma}{200}\cdot\frac{1}{\gamma^2}\cdot\log^4 n$,*

$$\mathbf{Pr}\left[\bigcup_{t\in[t_0,t_1]}\left\{\max_{i\in[n]} x_i^t \geqslant 0.008\cdot\frac{m}{n}\cdot\log n\right\}\right] \geqslant 1-n^{-1}.$$

Proof. Using Lemma 3.2 (for $k := \frac{1-\gamma}{200}\cdot 16\cdot\log^2 n \geqslant 3\cdot 18\cdot\log n$), we have for $t_1 = t_0 + \frac{1-\gamma}{200}\cdot\frac{1}{\gamma^2}\cdot\log^4 n$,

$$\mathbf{Pr}\left[\left\{F_{t_0}^{t_1} < \frac{n^2}{4m}\cdot(t_1-t_0+1)\right\} \cup \bigcup_{t\in[t_0,t_1]}\left\{\max_{i\in[n]} x_i^t \geqslant \frac{m}{n}\cdot\log n\right\}\right] \geqslant 1-n^{-2}. \tag{3.2}$$

Consider the $\log^3 n$ sub-intervals $\mathcal{I}_1,\ldots,\mathcal{I}_{\log^3 n}$ of length $\Delta = \frac{1-\gamma}{200}\cdot\frac{1}{\gamma^2}\cdot\log n$ with starting points $s_j := t_0 + \Delta\cdot(j-1)$. We also define the events for $j\in[\log^3 n]$,

$$\mathcal{C}_j := \left\{F_{s_j}^{s_j+\Delta} < \frac{n^2}{4m}\cdot\Delta\right\}.$$

Running the repeated balls-into-bins process over the interval $[s_j, s_j + \Delta]$ involves reallocating $\Delta \cdot n - F_{s_j}^{s_j+\Delta}$ balls, meaning we sample $\Delta \cdot n - F_{s_j}^{s_j+\Delta}$ many times a bin uniformly at random. So if the event $\mathcal{C}_j$ holds, then we sample in total

$$\Delta \cdot n - \frac{n^2}{4m} \cdot \Delta = (1-\gamma) \cdot \Delta \cdot n =: \overline{m}$$

bins. Hence these re-allocations correspond to a One-Choice process with $\overline{m}$ balls into n bins; let us denote its load vector by y^t for any round $t \geqslant 0$ starting from the empty load configuration. By Section 3 with $c := \frac{(1-\gamma)^2}{200} \cdot \frac{1}{\gamma^2}$, we obtain

$$\mathbf{Pr}\left[\max_{i\in[n]} y_i^{\overline{m}} \geqslant \left(c + \frac{\sqrt{c}}{10}\right) \cdot \log n\right] \geqslant 1 - n^{-2}.$$

Further, note that

$$\begin{aligned}\max_{i\in[n]} y_i^{\overline{m}} &\geqslant \left(\frac{(1-\gamma)^2}{200\gamma^2} + \frac{1}{10} \cdot \sqrt{\frac{(1-\gamma)^2}{200\gamma^2}}\right) \cdot \log n \\ &\geqslant \frac{1-\gamma}{200\gamma^2} \cdot \log n + 0.002 \cdot \frac{\log n}{\gamma} = \Delta + 0.002 \cdot \frac{\log n}{\gamma}.\end{aligned}$$

In Δ rounds, at most Δ balls can be removed from any single bin $i \in [n]$, so for any bin $i \in [n]$, $x_i^{s_j+\Delta} \geqslant x_i^{s_j} + y_i^{\overline{m}} - \Delta \geqslant y_i^{\overline{m}} - \Delta$. and hence

$$\max_{i\in[n]} x_i^{s_j+\Delta} \geqslant \max_{i\in[n]} y_i^{\overline{m}} - \Delta \geqslant 0.002 \cdot \frac{\log n}{\gamma}.$$

Defining for any round $t \geqslant 0$

$$\mathcal{E}^t := \left\{\max_{i\in[n]} x_i^t \geqslant 0.008 \cdot \frac{m}{n} \cdot \log n\right\},$$

we have shown that for any $1 \leqslant j \leqslant \log^3 n$,

$$\mathbf{Pr}\left[\mathcal{E}^{s_j+\Delta} \cup \neg\mathcal{C}_j\right] \geqslant 1 - n^{-2}.$$

By taking the union bound over the $\log^3 n$ sub-intervals, we conclude

$$\mathbf{Pr}\left[\bigcap_{j\in[\log^3 n]} \left(\left\{\max_{i\in[n]} x_i^{s_j+\Delta} \geqslant 0.008 \cdot \frac{m}{n} \cdot \log n\right\} \cup \neg\mathcal{C}_j\right)\right] \geqslant 1 - (\log^3 n) \cdot n^{-2}. \tag{3.3}$$

Assuming that $\left\{F_{t_0}^{t_1} < \frac{n^2}{4m} \cdot (t_1 - t_0 + 1)\right\}$ holds, then using the pigeonhole principle, at least one of these intervals j satisfies $\mathcal{C}_j$, i.e., $\{\cup_{j\in[\log^3 n]}\mathcal{C}_j\}$ holds. Hence, by the union bound of Equation (3.2) and Equation (3.3) we conclude that

$$\begin{aligned}\mathbf{Pr}\left[\bigcup_{t\in[t_0,t_1]} \mathcal{E}^t\right] &\geqslant \mathbf{Pr}\left[\bigcup_{j\in[\log^3 n]} \mathcal{E}^{s_j+\Delta}\right] \\ &\geqslant 1 - \mathbf{Pr}\left[\neg \bigcap_{j\in[\log^3 n]} \left(\mathcal{E}^{s_j+\Delta} \cup \neg\mathcal{C}_j\right) \cup \bigcap_{j\in[\log^3 n]} \neg\mathcal{C}_j\right] \\ &\geqslant \mathbf{Pr}\left[\bigcup_{j\in[\log^3 n]} \left(\mathcal{E}^{s_j+\Delta} \cup \neg\mathcal{C}_j\right)\right] - \mathbf{Pr}\left[\bigcap_{j\in[\log^3 n]} \neg\mathcal{C}_j\right] \\ &\geqslant 1 - n^{-2} - (\log^3 n) \cdot n^{-2} \geqslant 1 - n^{-1}.\end{aligned}$$

◀

4 Upper Bounds on the Maximum Load and Convergence Time

In this section, we outline the proofs for the $\mathcal{O}(\frac{m}{n} \cdot \log n)$ matching upper bound on the maximum load and the $\mathcal{O}(m^2/n)$ upper bound on the convergence time of the RBB process for any $n \leqslant m \leqslant \text{poly}(n)$. The omitted proofs and details can be found in the full version [25]. In Section 4.1, we introduce the exponential potential function and demonstrate its use on the simpler setting with $m \leqslant n$ balls, and in Section 4.2 we outline the proof for the more challenging case with $m \geqslant n$ balls.

4.1 The Exponential Potential and an Upper Bound for $m \leqslant n$

For the upper bounds, we make use of the *exponential potential function* defined as

$$\Phi^t := \Phi^t(\alpha) := \sum_{i=1}^{n} \Phi_i^t := \sum_{i=1}^{n} e^{\alpha x_i^t},$$

where x_i^t is the load of bin $i \in [n]$ at round $t \geqslant 0$ and $\alpha > 0$ is a smoothing parameter. For any round t with $\Phi^t = \text{poly}(n)$, we can deduce that

$$\max_{i \in [n]} x_i^t = \mathcal{O}\Big(\frac{\log n}{\alpha}\Big).$$

By choosing a smoothing parameter $\alpha = \Theta(n/m)$, this will give the desired bound on the maximum load.

We start by giving a general formula for the expected change of Φ over one step.

▶ **Lemma 4.1.** *Consider the RBB process with any $m \geqslant n$ and the potential $\Phi := \Phi(\alpha)$ for any $\alpha > 0$. Then, for any round $t \geqslant 0$,*

$$\mathbf{E}\left[\, \Phi^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] \leqslant \Phi^t \cdot e^{-\alpha} \cdot e^{\frac{e^\alpha - 1}{n} \cdot \kappa^t} + (n - \kappa^t) \cdot e^{\frac{e^\alpha - 1}{n} \cdot \kappa^t}.$$

Proof. Consider the expected contribution of a bin $i \in [n]$ with $x_i^t \geqslant 1$,

$$\begin{aligned}
\mathbf{E}\left[\, \Phi_i^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] &= \sum_{z=0}^{\kappa^t} e^{\alpha(x_i^t + z - 1)} \cdot \binom{\kappa^t}{z} \cdot \Big(\frac{1}{n}\Big)^z \cdot \Big(1 - \frac{1}{n}\Big)^{\kappa^t - z} \\
&= \Phi_i^t \cdot e^{-\alpha} \cdot \sum_{z=0}^{\kappa^t} \binom{\kappa^t}{z} \cdot \Big(\frac{e^\alpha}{n}\Big)^z \cdot \Big(1 - \frac{1}{n}\Big)^{\kappa^t - z} \\
&\overset{(a)}{=} \Phi_i^t \cdot e^{-\alpha} \cdot \Big(1 - \frac{1}{n} + \frac{e^\alpha}{n}\Big)^{\kappa^t} \\
&\overset{(b)}{\leqslant} \Phi_i^t \cdot e^{-\alpha} \cdot e^{\frac{e^\alpha - 1}{n} \cdot \kappa^t},
\end{aligned}$$

using in (a) the binomial identity $\sum_{z=0}^{k} \binom{k}{z} p^z q^{k-z} = (p+q)^k$ and in (b) that $1 + z \leqslant e^z$ for any $z \geqslant 0$.

For an empty bin $i \in [n]$, its expected contribution is

$$\mathbf{E}\left[\, \Phi_i^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] = \sum_{z=0}^{\kappa^t} \binom{\kappa^t}{z} \cdot e^{\alpha z} \cdot \Big(\frac{1}{n}\Big)^z \cdot \Big(1 - \frac{1}{n}\Big)^{\kappa^t - z} = \Big(1 - \frac{1}{n} + \frac{e^\alpha}{n}\Big)^{\kappa^t} \leqslant e^{\frac{e^\alpha - 1}{n} \cdot \kappa^t}.$$

Aggregating over all bins, we have

$$\begin{aligned}
\mathbf{E}\left[\, \Phi^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] &= \sum_{i \in [n] : x_i^t \geqslant 1} \mathbf{E}\left[\, \Phi_i^{t+1} \mid \mathfrak{F}^t \,\right] + \sum_{i \in [n] : x_i^t = 0} \mathbf{E}\left[\, \Phi_i^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] \\
&\leqslant \Phi^t \cdot e^{-\alpha} \cdot e^{\frac{e^\alpha - 1}{n} \cdot \kappa^t} + (n - \kappa^t) \cdot e^{\frac{e^\alpha - 1}{n} \cdot \kappa^t}.
\end{aligned}$$

◀

Now, we will investigate the simpler setting where m is much smaller than n, to demonstrate the use of the exponential potential function. This implies that in each round, deterministically at least $n - m$ bins are empty. As we prove below, this implies for example, that for $m = \frac{n}{\log n}$ we get w.h.p. a maximum load of $\mathcal{O}(\frac{\log n}{\log \log n})$ after $\mathcal{O}(\frac{n}{\log n})$ rounds.

▶ **Lemma 4.2.** *Consider the RBB process with $m \leqslant \frac{1}{e^2}n$. Then for any round $t \geqslant 2m$,*

$$\mathbf{Pr}\left[\max_{i\in[n]} x_i^t \leqslant 4 \cdot \frac{\log n}{\log\left(\frac{n}{em}\right)}\right] \geqslant 1 - n^{-2}.$$

Proof. We will use the potential $\Phi := \Phi(\alpha)$ with $\alpha := \log\left(\frac{n}{em}\right) \geqslant 1$ (since $m \leqslant \frac{1}{e^2}n$). Note that for $m = o(n)$ the potential is super-exponential, as in [22]. Since $\kappa^t \leqslant m$, we have

$$\frac{e^\alpha - 1}{n} \cdot \kappa^t \leqslant e^\alpha \cdot \frac{m}{n} = \frac{1}{e}.$$

Hence, by using Lemma 4.1,

$$\begin{aligned}\mathbf{E}\left[\Phi^{t+1} \mid \mathfrak{F}^t\right] &\leqslant \Phi^t \cdot e^{-\alpha} \cdot e^{\frac{e^\alpha-1}{n}\cdot\kappa^t} + (n - \kappa^t) \cdot e^{\frac{e^\alpha-1}{n}\cdot\kappa^t} \\ &\leqslant \Phi^t \cdot e^{-\alpha} \cdot e^{1/e} + e \cdot n \\ &\leqslant \Phi^t \cdot e^{-\frac{\alpha}{2}} + e \cdot n,\end{aligned}$$

using in the last inequality that $\alpha \geqslant 1$.

At round $t = 0$ we have $\Phi^0 \leqslant e^{\alpha m}$. Hence applying Lemma A.5, we have for any $t \geqslant 2m$,

$$\mathbf{E}\left[\Phi^t\right] \leqslant e^{\alpha m} \cdot e^{-\frac{1}{2}\cdot\alpha t} + \frac{e \cdot n}{1 - e^{-\frac{\alpha}{2}}} \leqslant 1 + \frac{e \cdot n}{1 - e^{-1/2}} \leqslant 3e \cdot n.$$

By applying Markov's inequality for $t \geqslant 2m$,

$$\mathbf{Pr}\left[\Phi^t \leqslant 3e \cdot n^3\right] \geqslant 1 - n^{-2}.$$

When $\{\Phi^t \leqslant 3e \cdot n^3\}$ holds, we have for any bin $i \in [n]$,

$$x_i^t \leqslant \frac{1}{\alpha} \cdot (\log(3e) + 3\log n) \leqslant 4 \cdot \frac{\log n}{\log\left(\frac{n}{em}\right)},$$

completing the proof. ◀

4.2 Upper Bound for $m \geqslant n$

We now turn to outlining the proof for the more challenging case of $m \geqslant n$. The omitted proofs an be found in the full version [25].

On a high level, we show that in a large enough interval, an $\Omega(m/n)$ fraction of the bins are empty, the opposite of what we had in Section 3. This follows through a coupling with an idealized version of the process, which is simpler to analyze. Then, in an interval with $\Omega(m/n)$ fraction of empty bins, the exponential potential with a sufficiently small smoothing parameter $\alpha = \Theta(n/m)$ drops in expectation, at some point becoming poly(n) and implying the $\mathcal{O}(\frac{m}{n} \cdot \log n)$ maximum load (**convergence**). Then, with a similar analysis over a slightly smaller interval we show that it remains in such a configuration for $\mathcal{O}(\frac{m}{n} \cdot \log n)$ rounds (**stabilization**).

In the analysis, we make use of the following bound on the expected change of Φ, which is a restatement of Lemma 4.1 based on the fraction of empty bins f^t.

▶ **Lemma 4.3.** *Consider the RBB process with any $m \geqslant n$ and the potential $\Phi := \Phi(\alpha)$ with any $0 < \alpha < 1.5$. Then, for any round $t \geqslant 0$,*

$$\mathbf{E}\left[\, \Phi^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] \leqslant \Phi^t \cdot e^{\alpha^2 - \alpha f^t} + 6n,$$

In particular, when the fraction of empty bins satisfies $f^t = \Omega(\alpha)$, the potential drops in expectation over one round, as was trivially the case for $m \ll n$. So it will be central to our analysis to prove a lower bound on the fraction of empty bins in a sufficiently long interval. This idea is inspired by [3, Lemma 19], who proved that in case of $m = n$, for each round, a constant fraction of the bins are empty with very high probability. This is useful, as it implies a constant additive drift for the load of each non-empty bin, which will drop by a constant term > 0 in expectation.

However, for general $m \gg n$, there will be starting configurations in which all bins remain non-empty for several rounds. Only if the process runs for a sufficiently long time, a small fraction of bins will become (and, to some extent, remain) empty. The following lemma quantifies this behavior and proves that, after a waiting time of $\mathcal{O}((m/n)^2)$ (the square of the average load), a fraction of $\mathcal{O}(n/m)$ of the bins will be empty per round on average. Hence for a time interval of length $(m/n)^2$, the aggregated "empty bin/round pairs" will be $\approx (m/n)^2 \cdot n \cdot (n/m) = m$.

[Key Lemma for the Upper Bound] Consider the RBB process with $m \geqslant n$ and any round $t_0 \geqslant 0$. Then, for round $t_3 := t_0 + 744(m/n)^2$ it holds that

$$\mathbf{Pr}\left[\, F_{t_0}^{t_3} \geqslant \frac{1}{384} \cdot m \;\middle|\; \mathfrak{F}^{t_0} \,\right] \geqslant 1 - e^{-\Omega(n)}.$$

First, let us remark that for the simpler case $m = n$, a stronger result was shown in [3, Lemma 1], proving that for *any* round $t \geqslant 1$, $F^t = \Omega(n)$ holds with probability $1 - \exp(-\Omega(n))$. In fact adjusting the proof in [3] slightly, the same result holds for any $m = \mathcal{O}(n)$. Therefore, we may assume in the following proof for convenience, that $m \geqslant C \cdot n$ for a sufficiently large constant $C > 0$ (we will choose $C := 6$). Alternatively, we can also reduce the case with m balls for some $m \in [n, C \cdot m]$ balls to the case with $C \cdot m$ balls, by using the fact that $F_{t_0}^{t_3}$ becomes stochastically smaller if we add more balls.

In order to establish Section 4.2, we will relate the RBB process to a simpler process, which we call the *idealized process*. In the idealized process, we also remove one ball from each non-empty bin at each round, but we allocate exactly n balls, regardless of how many bins are empty.

Formally, fix any load configuration of m balls with load vector x^{t_0}. The load vector of the idealized process is denoted by $y^t, t \geqslant t_0$ and defined as follows. For any bin $i \in [n]$, $y_i^{t_0} := x_i^{t_0}$. Further, for any $t \geqslant t_0$, let $Z_1^t, Z_2^t, \ldots, Z_n^t \in \{1, \ldots, n\}$ be n independent, uniform random samples. Then define,

$$y_i^{t+1} := y_i^t - \mathbf{1}_{y_i^t > 0} + \sum_{j=1}^{n} \mathbf{1}_{Z_j^t = i}. \tag{4.1}$$

Note that the marginal distribution of y_i^{t+1} can be expressed as

$$y_i^{t+1} = y_i^t - \mathbf{1}_{y_i^t > 0} + \mathsf{Bin}(n, 1/n).$$

Comparing this to the RBB process (see Equation (2.1)) we have the same distribution apart from that $\mathsf{Bin}(n, 1/n)$ is replaced by $\mathsf{Bin}(\kappa^t, 1/n)$. Thus we see that the *idealized process* is a bit simpler and also has the advantage that the number of balls that are added to the bins does not depend on the load configuration.

▶ **Lemma 4.4.** *For any round $t_0 \geqslant 0$ and load vector x^{t_0}, there is a coupling between the load vectors $(x^t)_{t \geqslant t_0}$ and $(y^t)_{t \geqslant t_0}$ such that for all rounds $t \geqslant t_0$ and for all bins $i \in [n]$, $x_i^t \leqslant y_i^t$.*

Based on this coupling, we also define for two rounds $t_0 \leqslant t_3$,

$$G_{t_0}^{t_3} := \sum_{t=t_0}^{t_3} \sum_{i \in [n]} \mathbf{1}_{y_i^t = 0}.$$

Note that Lemma 4.4 implies that $F_{t_0}^{t_3}$ is stochastically larger than $G_{t_0}^{t_3}$, therefore it suffices to analyze $G_{t_0}^{t_3}$ in the following.

Our first lemma proves that starting from any load configuration with m balls at time t_0, any bin $i \in [n]$ whose load is close to the average load, has a constant probability > 0 of reaching zero load after $\mathcal{O}((m/n)^2)$ rounds.

▶ **Lemma 4.5.** *Consider the idealized process with an arbitrary initial load configuration at time t_0 with $m \geqslant 6n$ balls. Let $i \in [n]$ be any bin with $y_i^{t_0} \leqslant 2 \cdot m/n$. Then,*

$$\mathbf{Pr}\left[\bigcup_{t_1 \in [t_0, t_0 + 720 \cdot \frac{m^2}{n^2}]} \{y_i^{t_1} = 0\} \;\middle|\; \mathfrak{F}^{t_0}, y_i^{t_0} \leqslant 2 \cdot \frac{m}{n} \right] \geqslant \frac{1}{4}.$$

The next lemma shows that once $y_i^t = 0$ occurs, then with constant probability bin i will have zero load in $\Omega(m/n)$ further rounds until time $\mathcal{O}((m/n)^2)$.

▶ **Lemma 4.6.** *Consider the idealized process with an arbitrary load configuration at round t_1 with $m \geqslant 6n$ balls, such that there is a bin $i \in [n]$ with $y_i^{t_1} = 0$. Then, for round $t_2 := t_1 + 24 \cdot (m/n)^2$,*

$$\mathbf{Pr}\left[\sum_{t=t_1}^{t_2} \mathbf{1}_{y_i^t = 0} \geqslant \frac{1}{6} \cdot \frac{m}{n} \;\middle|\; \mathfrak{F}^{t_1}, y_i^{t_1} = 0 \right] \geqslant \frac{1}{4}.$$

By combining Lemma 4.5 and Lemma 4.6, we derive the following lower bound on $\mathbf{E}\left[G_{t_0}^{t_3} \right]$, which by the coupling also holds for $\mathbf{E}\left[F_{t_0}^{t_3} \right]$.

▶ **Lemma 4.7.** *Consider the idealized process with m balls, where $m \geqslant 6n$. Then, for any round $t_0 \geqslant 0$ and for $t_3 := t_0 + 744 \cdot (m/n)^2$,*

$$\mathbf{E}\left[G_{t_0}^{t_3} \;\middle|\; \mathfrak{F}^{t_0} \right] \geqslant \frac{1}{192} \cdot m.$$

Using the Method of Bounded Differences (Theorem A.3), we get this w.h.p.

▶ **Lemma 4.8.** *Consider the idealized process with m balls, where $m \geqslant 6n$. Then, for any round $t_0 \geqslant 0$ and for $t_3 := t_0 + 744 \cdot (m/n)^2$,*

$$\mathbf{Pr}\left[G_{t_0}^{t_3} \geqslant \frac{1}{384} \cdot m \;\middle|\; \mathfrak{F}^{t_0} \right] \geqslant 1 - e^{-\Omega(n)}.$$

Upper Bound on Convergence Time

To bound the convergence time for $m \geqslant n$, we use the Φ with $\alpha = \Theta(n/m)$ and show that in $\mathcal{O}(m^2/n)$ rounds the process reaches a configuration with $\Phi^t < \frac{48}{\alpha^2} \cdot n$. In such a step, the maximum load is $\mathcal{O}(m/n \cdot \log m)$, which becomes $\mathcal{O}(m/n \cdot \log n)$ for $n \leqslant m \leqslant \text{poly}(n)$.

We start by proving that potential drops in expectation when it is sufficiently large and there is a large fraction of empty bins.

▶ **Lemma 4.9.** *Consider the RBB process for any $m \geqslant n$, and the potential $\Phi := \Phi(\alpha)$ with $\alpha := \frac{1}{2\cdot 384\cdot 744} \cdot \frac{n}{m}$. Then for any round $t \geqslant 0$,*

$$\mathbf{E}\left[\, \Phi^{t+1} \,\middle|\, \mathfrak{F}^t \,\right] \leqslant \Phi^t \cdot e^{\alpha^2 - \alpha f^t} + 6n.$$

In particular,

$$\mathbf{E}\left[\Phi^{t+1} \,\middle|\, \mathfrak{F}^t, \Phi^t > \frac{48}{\alpha^2}\cdot n \right] \leqslant \Phi^t \cdot e^{1.5\alpha^2 - \alpha f^t}.$$

We now define the event

$$\mathcal{E}^t := \left\{ \Phi^t \leqslant \frac{48}{\alpha^2}\cdot n \right\}.$$

When $\mathcal{E}^t$ holds, the potential is small enough to imply a maximum load of $\mathcal{O}(m/n \cdot \log m)$. When it is large, it drops in expectation by a multiplicative factor in any round with $f^t = \Omega(m/n)$. We now define for any $t_0 \geqslant 0$, the *adjusted exponential potential function* $\widetilde{\Phi}^s_{t_0} := \widetilde{\Phi}^s_{t_0}(\alpha)$, with $\widetilde{\Phi}^{t_0}_{t_0} := \Phi^{t_0}(\alpha)$ and for any $s > t_0$

$$\widetilde{\Phi}^s_{t_0} := \mathbf{1}_{\cap_{t\in[t_0,s)} \neg\mathcal{E}^t} \cdot \Phi^s(\alpha) \cdot \exp\left(\sum_{t=t_0}^{s-1} (\alpha f^t - 1.5\alpha^2) \right).$$

This forms a super-martingale. Using Section 4.2, we will show that in a $\Theta(m^2/n)$ interval w.h.p. the potential becomes small at least once, implying the $\mathcal{O}(m/n \cdot \log m)$ bound.

[Convergence] Consider the RBB process for any $m \geqslant n$ and the potential $\Phi := \Phi(\alpha)$ for $\alpha > 0$ as defined in Lemma 4.9. Let $c_r := 16 \cdot 384^2 \cdot 744^2$. For any round $t_0 \geqslant 0$, for $t_1 := t_0 + c_r \cdot \frac{m^2}{n}$, we have

$$\mathbf{Pr}\left[\bigcup_{t\in[t_0,t_1]} \left\{ \Phi^t \leqslant \frac{48}{\alpha^2}\cdot n \right\} \right] \geqslant 1 - e^{-\Omega(n)}.$$

In particular, this implies that for $m = \mathrm{poly}(n)$, there exists a constant $C > 0$ such that

$$\mathbf{Pr}\left[\bigcup_{t\in[t_0,t_1]} \left\{ \max_{i\in[n]} x_i^t \leqslant C \cdot \frac{m}{n} \cdot \log m \right\} \right] \geqslant 1 - e^{-\Omega(n)}.$$

Upper Bound on the Maximum Load

We will now show that, for any $n \leqslant m \leqslant \mathrm{poly}(n)$, once a configuration with $\Phi^t \leqslant \frac{48}{\alpha^2}\cdot n$ is reached, then w.h.p. the process will re-visit such a configuration in the next $\mathcal{O}(m^2/n \cdot \log n)$ rounds. The proof is quite similar to Section 4.2, but with intervals of shorter lengths. By a One-Choice argument we will deduce that the maximum load in every of the in-between rounds is $\mathcal{O}(m/n \cdot \log n)$ and so the maximum load remains small for $\mathrm{poly}(n)$ rounds.

▶ **Lemma 4.10.** *Consider the RBB process with $n \leqslant m \leqslant n^k$ for some constant $k \geqslant 1$ and the potential $\Phi := \Phi(\alpha)$ for $\alpha > 0$ as defined in Lemma 4.9. Further, let $c_s := 8k \cdot 16 \cdot 384^2 \cdot 744^2$. Then, for any round $t_0 \geqslant 0$ and for $t_1 := t_0 + c_s \cdot \frac{m^2}{n^2} \cdot \log n$, we have*

$$\mathbf{Pr}\left[\bigcup_{t\in[t_0,t_1]} \left\{ \Phi^t \leqslant \frac{48}{\alpha^2}\cdot n \right\} \;\middle|\; \mathfrak{F}^{t_0}, \Phi^{t_0} \leqslant e^{\alpha \log n} \cdot \frac{48}{\alpha^2}\cdot n \right] \geqslant 1 - n^{-7k}.$$

Finally, combining Section 4.2 and Lemma 4.10 we can derive the following upper bound on the maximum load, which holds for poly(n) rounds.

▶ **Theorem 4.11** (Stabilization). *Consider the RBB process with any $n \leqslant m \leqslant n^k$ for some constant $k \geqslant 1$. There exists a constant $C > 0$ such that, for any $t \geqslant c_r \cdot \frac{m^2}{n}$, where $c_r > 0$ is the constant defined in Section 4.2,*

$$\mathbf{Pr}\left[\bigcap_{s\in[t,t+m^2]}\left\{\max_{i\in[n]} x_i^s \leqslant C\cdot\frac{m}{n}\cdot\log n\right\}\right] \geqslant 1-n^{-2k}.$$

5 The Multi-Token Traversal Time

As mentioned in [3], it is natural to regard the RBB process as a multi-token traversal problem, in which each ball should visit all bins as frequently as possible. This can be seen as a "cover time" of parallel and dependent random walks, which is the first time until each ball has been allocated at least once to every bin. In [3, Corollary 1], a w.h.p. bound of $\mathcal{O}(n\log^2 n)$ on this quantity was established (it was also shown that this bound holds even in an adversarial setting, where an adversary is able to re-allocate all tokens arbitrarily every $\mathcal{O}(n)$ rounds). For the original setting without the adversary, we show:

Consider the RBB with any $m \geqslant n$. Then, with probability $1-m^{-2}$, each of the m balls traverses all n bins within $28m\cdot\log m$ rounds. Furthermore, any fixed ball needs with probability at least $1-o(1)$ at least $1/16\cdot m\cdot\log n$ rounds until all n bins are traversed.

6 Experiments

We complement our analysis with some experimental results in Figure 2 and Figure 3.

In Figure 2, we the plot the maximum load vs the average number of balls for $n \in \{10^2, 10^3, 10^4\}$ and $m \in \{n, 2n, \dots 50n\}$ after 10^6 rounds starting with the uniform distribution. The trend seems to be linear in m/n as m grows, which is in accordance with the $\Theta(m/n\cdot\log n)$ bound on the maximum load shown by our theoretical analysis in Lemma 3.3 and Theorem 4.11. In Figure 3, we plot of the fraction of empty bins vs the average number of balls for $n \in \{10^2, 10^3, 10^4\}$ and $m \in \{n, 2n, \dots 50n\}$ averaged over 10^6 rounds, starting from the uniform load vector. The trend supports that the fraction is $\Theta(n/m)$ in the steady state, as proven in Lemma 3.2 and Section 4.2.

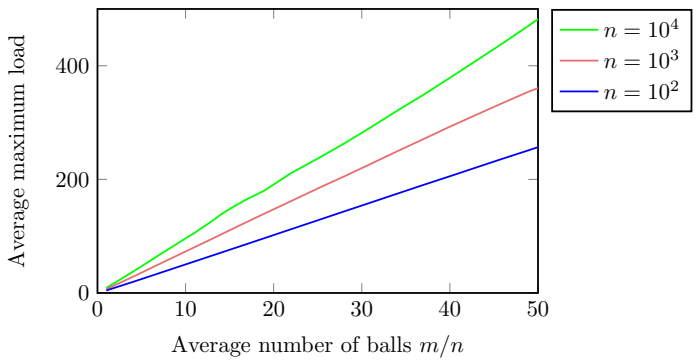


Figure 2 Maximum load vs average number of balls for $n \in \{10^2, 10^3, 10^4\}$ and $m \in \{n, 2n, \dots 50n\}$ after 10^6 rounds, starting from the uniform load vector (averaged over 25 runs).

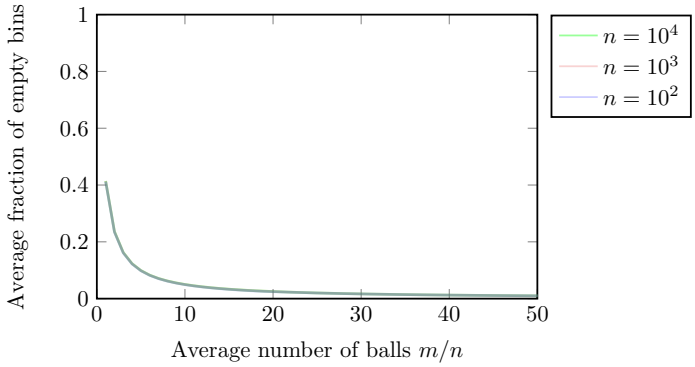


Figure 3 Fraction of empty bins vs the average load for $n \in \{10^2, 10^3, 10^4\}$ and $m \in \{n, 2n, \dots 50n\}$ averaged over 10^6 rounds, starting from the uniform load vector (averaged over 25 runs). Note that for all values of n, the curves are very close to one another.

7 Conclusions

We revisited the RBB process and proved that for any $m \geqslant n$ that w.h.p. after $\mathcal{O}(m^2/n)$ rounds it achieves an $\mathcal{O}(m/n \cdot \log m)$ maximum load. For $n \leqslant m \leqslant \text{poly}(n)$ we show that it stabilizes in a configuration with an $\mathcal{O}(m/n \cdot \log n)$ maximum load, for at least m^2 rounds and also prove a lower bound matching up to multiplicative constants. This resolved two conjectures in [3]. We also obtained an upper bound of $\mathcal{O}(m \cdot \log m)$ on the traversal time for the balls, which was shown to be tight for any $m = \text{poly}(n)$.

There are several possible extensions, such as generalizing the stabilization result for $m = n^{\omega(1)}$, determining whether the $\mathcal{O}(m^2/n)$ convergence time is tight for $m = \omega(n)$ and determining tight bounds for the maximum load when $m < n$.

Finally, as mentioned in [3], an interesting but also challenging generalization is the RBB process on graphs. We hope that at least some of our arguments could be leveraged, for example, the insight in Section 4.2 that many bins become empty within $\mathcal{O}((m/n)^2)$ rounds might extend to graphs.

References

1 Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999. doi:10.1137/S0097539795288490.

2 Jialu Bao, Marco Gaboardi, Justin Hsu, and Joseph Tassarotti. A separation logic for negative dependence. In *Proceedings of the ACM on Programming Languages (POPL)*, volume 6, New York, NY, USA, January 2022. Association for Computing Machinery. doi:10.1145/3498719.

3 Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Gustavo Posta. Self-stabilizing repeated balls-into-bins. *Distributed Computing*, 32(1):59–68, 2019. (earlier version in SPAA'15). doi:10.1007/s00446-017-0320-4.

4 Luca Becchetti, Andrea E. F. Clementi, Emanuele Natale, Francesco Pasquale, and Luca Trevisan. Stabilizing consensus with many opinions. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 620–635, Philadelphia, PA, 2016. SIAM. doi:10.1137/1.9781611974331.ch46.

5 Petra Berenbrink, Artur Czumaj, Matthias Englert, Tom Friedetzky, and Lars Nagel. Multiple-choice balanced allocation in (almost) parallel. In *Proceedings of the 16th International Workshop on Randomization and Computation (RANDOM)*, pages 411–422, Berlin Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-32512-0_35.

6 Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: the heavily loaded case. *SIAM Journal on Computing*, 35(6):1350–1385, 2006. doi:10.1137/S009753970444435X.

7 Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, Zengjian Hu, and Russell Martin. Distributed selfish load balancing. *SIAM Journal on Computing*, 37(4):1163–1181, 2007. doi:10.1137/060660345.

8 Petra Berenbrink, Tom Friedetzky, Peter Kling, Frederik Mallmann-Trenn, Lars Nagel, and Chris Wastell. Self-stabilizing balls and bins in batches: the power of leaky bins. *Algorithmica. An International Journal in Computer Science*, 80(12):3673–3703, 2018. doi:10.1007/s00453-018-0411-z.

9 Petra Berenbrink, Martin Hoefer, and Thomas Sauerwald. Distributed selfish load balancing on networks. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1487–1497. SIAM, Philadelphia, PA, 2011.

10 Nicoletta Cancrini and Gustavo Posta. Propagation of chaos for a balls into bins model. *Electronic Communications in Probability*, 24:Paper No. 1, 9, 2019. doi:10.1214/18-ECP204.

11 Nicoletta Cancrini and Gustavo Posta. Mixing time for the repeated balls into bins dynamics. *Electronic Communications in Probability*, 25:Paper No. 60, 14, 2020. doi:10.1214/20-ecp338.

12 Nicoletta Cancrini and Gustavo Posta. Propagation of chaos for a general balls into bins dynamics. *Electronic Journal of Probability*, 26:Paper No. 23, 20, 2021. doi:10.1214/21-EJP590.

13 Fan Chung and Linyuan Lu. Concentration inequalities and martingale inequalities: a survey. *Internet Mathematics*, 3(1):79–127, 2006. URL: http://projecteuclid.org/euclid.im/1175266369.

14 Colin Cooper. Random walks, interacting particles, dynamic networks: Randomness can be helpful. In *Proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 6796 of *Lecture Notes in Computer Science*, pages 1–14, Berlin, Heidelberg, 2011. Springer. doi:doi.org/10.1007/978-3-642-22212-2_1.

15 Artur Czumaj, Chris Riley, and Christian Scheideler. Perfectly balanced allocation. In *Proceedings of the 7th International Workshop on Randomization and Computation (RANDOM)*, volume 2764 of *Lecture Notes in Computer Science*, pages 240–251. Springer, Berlin, 2003. doi:10.1007/978-3-540-45198-3_21.

16 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, Cambridge, 2009. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

17 Yehuda Hassin and David Peleg. Distributed probabilistic polling and applications to proportionate agreement. *Information and Computation*, 171(2):248–268, 2001. doi:10.1006/inco.2001.3088.

18 Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 119–131, New York, NY, USA, 1990. ACM. doi:10.1145/93385.93409.

19 James R. Jackson. Jobshop-like queueing systems. *Management Science*, 50(12 Supplement):1796–1802, December 2004. doi:10.1287/mnsc.1040.0268.

20 Richard M. Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. *Algorithmica. An International Journal in Computer Science*, 16(4-5):517–542, 1996. doi:10.1007/BF01940878.

21 Frank P. Kelly. Networks of queues. *Advances in Applied Probability*, 8(2):416–432, 1976. doi:10.2307/1425912.

22 Dimitrios Los and Thomas Sauerwald. Balanced Allocations with Incomplete Information: The Power of Two Queries. In *Proceedings of the 13th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 215 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 103:1–103:23, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2022.103.

23 Dimitrios Los and Thomas Sauerwald. Balanced allocations with the choice of noise. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 164–175, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519270.3538428.

24 Dimitrios Los and Thomas Sauerwald. Brief announcement: Tight bounds for repeated balls-into-bins. In *Proceedings of the 34th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 419–421, New York, NY, USA, 2022. ACM. doi:10.1145/3490148.3538561.

25 Dimitrios Los and Thomas Sauerwald. Tight bounds for repeated balls-into-bins, 2022. doi:10.48550/arXiv.2203.12400.

26 Dimitrios Los, Thomas Sauerwald, and John Sylvester. Balanced Allocations: Caching and Packing, Twinning and Thinning. In *Proceedings of the 33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1847–1874, Alexandria, Virginia, 2022. SIAM. doi:10.1137/1.9781611977073.74.

27 Michael Mitzenmacher, Andréa W. Richa, and Ramesh Sitaraman. The power of two random choices: a survey of techniques and results. In *Handbook of randomized computing, Vol. I, II*, volume 9 of *Combinatorial Optimization*, pages 255–312. Kluwer Acad. Publ., Dordrecht, Netherlands, 2001. doi:10.1007/978-1-4615-0013-1_9.

28 Michael Mitzenmacher and Eli Upfal. *Probability and computing*. Cambridge University Press, Cambridge, second edition, 2017. Randomization and probabilistic techniques in algorithms and data analysis.

29 David Peleg and Eli Upfal. The token distribution problem. *SIAM Journal on Computing*, 18(2):229–243, 1989. doi:10.1137/0218015.

30 Yuval Peres, Kunal Talwar, and Udi Wieder. Graphical balanced allocations and the $(1+\beta)$-choice process. *Random Structures & Algorithms*, 47(4):760–775, 2015. doi:10.1002/rsa.20558.

31 Martin Raab and Angelika Steger. "Balls into bins"—a simple and tight analysis. In *Proceedings of the 2nd International Workshop on Randomization and Computation (RANDOM)*, volume 1518, pages 159–170. Springer, Barcelona, Spain, 1998. doi:10.1007/3-540-49543-6_13.

32 Udi Wieder. Hashing, load balancing and multiple choice. *Foundations and Trends in Theoretical Computer Science*, 12(3-4):275–379, 2017. doi:10.1561/0400000070.

33 Mingxi Yin, Yuli Yang, Jen-Ming Wu, and Bingli Jiao. Opportunistic bits in short-packet communications: A finite blocklength perspective. *IEEE Transactions on Communications*, 69(12):8085–8099, 2021. doi:10.1109/TCOMM.2021.3117606.

A Tools

A.1 Facts about the One-Choice Process

In this section, we prove two well-known properties of the One-Choice process. In the first lemma, we show that the quadratic potential is w.h.p. $\mathcal{O}(n)$ for n balls into n bins.

▶ **Lemma A.1.** *Consider the One-Choice process for n balls into n bins. Then, for any $t \geqslant 0$*

$$\mathbf{Pr}\left[\,\Upsilon^t \leqslant 3n\,\right] \geqslant 1 - n^{-\omega(1)}.$$

Proof. Recall that

$$\Upsilon^t := \sum_{i=1}^{n} \Upsilon_i^t = \sum_{i=1}^{n} (x_i^t)^2.$$

Since x_i^t has distribution $\mathsf{Bin}(t, 1/n)$ for any $t \geqslant 0$, $\mathbf{E}\left[\,(x_i^n)^2\,\right] = (1-1/n)+1 = 2-\frac{1}{n}$. Hence by linearity of expectations,

$$\mathbf{E}\left[\,\Upsilon^n\,\right] \leqslant 2n.$$

Define

$$\widetilde{\Upsilon}^n := \sum_{i=1}^{n} \min\{\Upsilon_i^n, \log^2 n\},$$

and note that $\Upsilon^n = \widetilde{\Upsilon}^n$ if and only if the maximum load is at most $\log n$. Then,

$$\mathbf{E}\left[\,\widetilde{\Upsilon}^n\,\right] \leqslant \mathbf{E}\left[\,\Upsilon^n\,\right] \leqslant 2n.$$

Further, $\widetilde{\Upsilon}^n$ is a function of n independent random variables (the random bin choices of the n balls), and changing one of these choices can change $\widetilde{\Upsilon}^n$ by at most $(\log n)^2 - (\log n - 1)^2 \leqslant 2\log n$. Hence by the Method of Bounded Differences (Theorem A.3),

$$\mathbf{Pr}\left[\,\widetilde{\Upsilon}^n - \mathbf{E}\left[\,\widetilde{\Upsilon}^n\,\right] \geqslant \lambda\,\right] \leqslant \exp\left(-\frac{\lambda^2}{2\sum_{i=1}^{n} 4(\log n)^2}\right),$$

and choosing $\lambda = n$ yields,

$$\mathbf{Pr}\left[\,\widetilde{\Upsilon}^n \geqslant 3n\,\right] \leqslant \mathbf{Pr}\left[\,\widetilde{\Upsilon}^n \geqslant \mathbf{E}\left[\,\widetilde{\Upsilon}^n\,\right] + n\,\right] \leqslant n^{-\omega(1)}.$$

Further, since the maximum load is larger than $\log n$ with probability $1 - n^{-\omega(1)}$, we have by the union bound

$$\mathbf{Pr}\left[\,\Upsilon^n \geqslant 3n\,\right] \leqslant \mathbf{Pr}\left[\left\{\widetilde{\Upsilon}^n \geqslant 3n\right\} \cup \left\{\max_{i\in[n]} x_i^n > \log n\right\}\right] \leqslant n^{-\omega(1)} + n^{-\omega(1)} = 2n^{-\omega(1)}. \blacktriangleleft$$

The next standard result was also used in [30, Section 4] and is based on [31]. For convenience of the reader, we give a self-contained proof, obtaining high probability bounds. [cf. [26, Lemma 10.4]] Consider the One-Choice process with $m = cn\log n$ balls, for any $c \geqslant 1/\log n$. Then, we have

$$\mathbf{Pr}\left[\max_{i\in[n]} x_i^m \geqslant \left(c + \frac{\sqrt{c}}{10}\right)\cdot \log n\right] \geqslant 1 - n^{-2}.$$

Proof. In order to use the Poisson Approximation [28, Chapter 5], let $Y_1, Y_2, \ldots, Y_n$ be n independent Poisson random variables with parameter $\lambda = \frac{m}{n} = c\log n$. Then,

$$\mathbf{Pr}\left[Y_i \geqslant \lambda + \frac{\sqrt{c}}{10}\cdot\log n\right] \geqslant \mathbf{Pr}\left[Y_i = \lambda + \frac{\sqrt{c}}{10}\cdot\log n\right] = e^{-\lambda}\cdot\frac{\lambda^{\lambda+\frac{\sqrt{c}}{10}\cdot\log n}}{(\lambda+\frac{\sqrt{c}}{10}\cdot\log n)!}.$$

Using that $z! \leqslant \sqrt{2\pi z}\left(\frac{z}{e}\right)^z e^{\frac{1}{12z}}$ for any integer $z \geqslant 1$,

$$\begin{aligned}\mathbf{Pr}\left[Y_i = \lambda + \frac{\sqrt{c}}{10}\cdot\log n\right] &\geqslant \frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{-\lambda}\cdot\left(\frac{e\lambda}{\lambda+\frac{\sqrt{c}}{10}\cdot\log n}\right)^{\lambda+\frac{\sqrt{c}}{10}\cdot\log n}\\ &\geqslant \frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{\frac{\sqrt{c}}{10}\log n}\cdot\left(1+\frac{1}{10\sqrt{c}}\right)^{-\lambda-\frac{\sqrt{c}}{10}\cdot\log n}\end{aligned}$$

$$\begin{aligned}
&\geqslant \frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{\frac{\sqrt{c}}{10}\log n}\cdot e^{-\frac{1}{10\sqrt{c}}\cdot(\lambda+\frac{\sqrt{c}}{10}\cdot\log n)}\\
&\geqslant \frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{\frac{\sqrt{c}}{10}\log n-\frac{1}{10\sqrt{c}}\lambda-\frac{1}{100}\log n}\\
&\geqslant \frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{-\frac{1}{100}\log n}
\end{aligned}$$

Since for any $k \geqslant 0$,

$$\frac{\mathbf{Pr}\left[Y_i = k+1\right]}{\mathbf{Pr}\left[Y_i = k\right]} = \frac{\lambda}{k+1},$$

we conclude that

$$\begin{aligned}
\mathbf{Pr}\left[Y_i \geqslant \lambda+\frac{\sqrt{c}}{10}\cdot\log n\right] &\geqslant \sum_{k=0}^{\sqrt{\lambda}-1}\mathbf{Pr}\left[Y_i = \lambda+\frac{\sqrt{c}}{10}\cdot\log n+k\right]\\
&\geqslant \sqrt{\lambda}\cdot\mathbf{Pr}\left[Y_i = \lambda+\frac{\sqrt{c}}{10}\cdot\log n+\sqrt{\lambda}\right]\\
&\geqslant \sqrt{\lambda}\cdot\mathbf{Pr}\left[Y_i = \lambda+\frac{\sqrt{c}}{10}\cdot\log n\right]\cdot\prod_{k=1}^{\sqrt{\lambda}}\left(\frac{\lambda}{\lambda+\frac{\sqrt{c}}{10}\cdot\log n+k}\right)\\
&\geqslant \sqrt{\lambda}\cdot\frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{-\frac{1}{100}\log n}\cdot\left(\frac{\lambda}{\lambda+\frac{\sqrt{c}}{10}\cdot\log n+\sqrt{\lambda}}\right)^{\sqrt{\lambda}}\\
&\geqslant \sqrt{\lambda}\cdot\frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{-\frac{1}{100}\log n}\cdot\left(1+\frac{1}{5\sqrt{c}}\right)^{-\sqrt{\lambda}}\\
&\geqslant \sqrt{\lambda}\cdot\frac{1}{4\cdot\sqrt{2\pi\lambda}}\cdot e^{-\frac{1}{100}\log n}\cdot e^{-\frac{1}{5}\sqrt{\log n}}\\
&\geqslant e^{-\frac{1}{99}\log n} = n^{-1/99},
\end{aligned}$$

where the last inequality holds for sufficiently large n. Hence,

$$\mathbf{Pr}\left[\bigcup_{i=1}^{n}\left\{Y_i \geqslant \lambda+\frac{\sqrt{c}}{10}\cdot\log n\right\}\right] \geqslant 1-\left(1-n^{-1/99}\right)^n \geqslant 1-n^{-3}.$$

Hence for $\widetilde{\mathcal{E}} := \left\{\max_{i\in[n]} Y_i \geqslant \lambda+\frac{\sqrt{c}}{10}\cdot\log n\right\}$, we have $\mathbf{Pr}\left[\neg\widetilde{\mathcal{E}}\right] \leqslant n^{-3}$. Note that $\widetilde{\mathcal{E}}$ is a monotone event under adding balls, and thus with $\mathcal{E} := \left\{\max_{i\in[n]} x_i^m \geqslant \lambda+\frac{\sqrt{c}}{10}\cdot\log n\right\}$, we have by [28, Corollary 5.11])

$$\mathbf{Pr}\left[\neg\mathcal{E}\right] \leqslant 2\cdot\mathbf{Pr}\left[\neg\widetilde{\mathcal{E}}\right] \leqslant 2\cdot n^{-3} \leqslant n^{-2}.$$ ◀

A.2 A Simple Bound for the RBB process

In this section, we show that given that the maximum load is small in the current step, then the change of the quadratic potential is w.h.p. small over the next step.

▶ **Lemma A.2.** *Consider the RBB process with $m \geqslant n$ balls and n bins. For any round $t \geqslant 0$, we have,*

$$\mathbf{Pr}\left[\left|\Upsilon^{t+1}-\Upsilon^t\right| \leqslant 2\cdot m\cdot\log n+4n \;\middle|\; \max_{i\in[n]} x_i^t \leqslant \frac{m}{n}\cdot\log n\right] \geqslant 1-n^{-\omega(1)}.$$

Proof. Let $k_i \in [0, n]$ be the number of balls that each bin receives at round t. For any bin $i \in [n]$ with $x_i^t > 0$,

$$\begin{aligned}|\Upsilon_i^{t+1} - \Upsilon_i^t| &= |(x_i^t + k_i - 1)^2 - (x_i^t)^2| = |2 \cdot (k_i - 1) \cdot x_i^t + (k_i - 1)^2| \\ &\leqslant 2 \cdot x_i^t \cdot k_i + (k_i)^2 + 1.\end{aligned}$$

For any bin $i \in [n]$ with $x_i^t = 0$,

$$|\Upsilon_i^{t+1} - \Upsilon_i^t| = k_i^2.$$

Aggregating over all bins, we have

$$|\Upsilon^{t+1} - \Upsilon^t| \leqslant \sum_{i=1}^{n} 2 \cdot x_i^t \cdot k_i + \sum_{i=1}^{n} k_i^2 + n. \tag{A.1}$$

Using Lemma A.1 we have

$$\mathbf{Pr}\left[\sum_{i=1}^{n} k_i^2 \leqslant 3n\right] \geqslant 1 - n^{-\omega(1)}.$$

When the event $\{\sum_{i=1}^{n} k_i^2 \leqslant 3n\}$ holds, and by the condition $\{\max_{i\in[n]} x_i^t \leqslant \frac{m}{n} \cdot \log n\}$ and $m \geqslant n$, we finally conclude from Equation (A.1)

$$|\Upsilon^{t+1} - \Upsilon^t| \leqslant 2 \cdot n \cdot \frac{m}{n} \cdot \log n + 4n = 2 \cdot m \cdot \log n + 4n. \qquad \blacktriangleleft$$

A.3 Concentration Inequalities

In this section, we state the Method of Bounded Differences and a concentration inequality with a bad event.

▶ **Theorem A.3** ([16, Corollary 5.2]). *Consider a function $f : \prod_{i\in[N]} \Omega_i \to \mathbb{R}$ such that it satisfies the Lipschitz condition with bounds $(c_i)_{i\in[N]}$. For independent random variables $X^1, \ldots, X^N$ with X^i taking values in Ω_i, we have that for any $\lambda > 0$*

$$\mathbf{Pr}\left[f(X^1, \ldots, X^N) \geqslant \mathbf{E}\left[f(X^1, \ldots, X^N)\right] + \lambda\right] \leqslant \exp\left(-\frac{2 \cdot \lambda^2}{\sum_{i=1}^{N} c_i^2}\right).$$

In order to state the concentration inequality for supermartingales conditional on a bad event not occurring, we introduce the following definitions from [13]. Consider any random variable X (in our case it will be the Z^t, the adjusted quadratic potential in Lemma 3.2) that can be evaluated by a sequence of decisions $Y^1, Y^2, \ldots, Y^N$ of finitely many outputs (the allocated balls). We can describe the process by a *decision tree* T, a complete rooted tree with depth n with vertex set $V(T)$. Each edge uv of T is associated with a probability p_{uv} depending on the decision made from u to v.

We say $f : V(T) \to \mathbb{R}$ satisfies an *admissible condition* P if $P = \{P_v\}$ holds for every vertex v. For an admissible condition P, the associated bad set $\mathcal{B}^i$ over the X_i is defined to be

$$\mathcal{B}^i = \{v \mid \text{the depth of } v \text{ is } i, \text{ and } P_u \text{ does not hold for some ancestor } u \text{ of } v\}.$$

▶ **Theorem A.4** (Theorem 8.3 in [13]). *For a filtration* $\mathbf{F}$*,*

$$\{\emptyset, \Omega\} = \mathfrak{F}^0 \subseteq \mathfrak{F}^1 \subseteq \ldots \subseteq \mathfrak{F}^N,$$

suppose that the random variable X^i *is* $\mathfrak{F}^i$*-measurable for* $0 \leqslant i \leqslant N$*. Let* $\mathcal{B} = \mathcal{B}^N$ *denote the bad set with the following admissible condition:*

$$\begin{aligned} &\mathbf{E}\left[\, X^i \mid \mathfrak{F}^{i-1} \,\right] \leqslant X^{i-1}, \\ &\quad |X^i - X^{i-1}| \leqslant c_i, \end{aligned}$$

for $1 \leqslant i \leqslant N$ *and for* $c_1, \ldots, c_N \geqslant 0$*. Then, we have*

$$\mathbf{Pr}\left[\, X^N \geqslant X^0 + \lambda \,\right] \leqslant \exp\left(-\frac{\lambda^2}{2 \cdot \sum_{i=1}^N c_i^2}\right) + \mathbf{Pr}\left[\, \mathcal{B} \,\right].$$

A.4 Auxiliary Probabilistic Claim

▶ **Lemma A.5.** *Consider a sequence of random variables* $(Z_i)_{i \in \mathbb{N}}$ *such that there are* $0 < a < 1$ *and* $b > 0$ *such that every* $i \geqslant 1$*,*

$$\mathbf{E}\left[\, Z_i \mid Z_{i-1} \,\right] \leqslant Z_{i-1} \cdot a + b.$$

Then for every $i \geqslant 1$*,*

$$\mathbf{E}\left[\, Z_i \mid Z_0 \,\right] \leqslant Z_0 \cdot a^i + \frac{b}{1-a}.$$

Proof. We will prove by induction that for every $i \in \mathbb{N}$,

$$\mathbf{E}\left[\, Z_i \mid Z_0 \,\right] \leqslant Z_0 \cdot a^i + b \cdot \sum_{j=0}^{i-1} a^j.$$

For $i = 0$, $\mathbf{E}\left[\, Z_0 \mid Z_0 \,\right] \leqslant Z_0$. Assuming the induction hypothesis holds for some $i \geqslant 0$, then since $a > 0$,

$$\begin{aligned} \mathbf{E}\left[\, Z_{i+1} \mid Z_0 \,\right] &= \mathbf{E}\left[\, \mathbf{E}\left[\, Z_{i+1} \mid Z_i \,\right] \mid Z_0 \,\right] \leqslant \mathbf{E}\left[\, Z_i \mid Z_0 \,\right] \cdot a + b \\ &\leqslant \left(Z_0 \cdot a^i + b \cdot \sum_{j=0}^{i-1} a^j\right) \cdot a + b \\ &= Z_0 \cdot a^{i+1} + b \cdot \sum_{j=0}^{i} a^j. \end{aligned}$$

The claims follows using that for $a \in (0,1)$, $\sum_{j=0}^{\infty} a^j = \frac{1}{1-a}$. ◀

Maintaining CMSO_2 Properties on Dynamic Structures with Bounded Feedback Vertex Number

Konrad Majewski
Institute of Informatics, University of Warsaw, Poland

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland

Marek Sokołowski
Institute of Informatics, University of Warsaw, Poland

Abstract

Let φ be a sentence of CMSO_2 (monadic second-order logic with quantification over edge subsets and counting modular predicates) over the signature of graphs. We present a dynamic data structure that for a given graph G that is updated by edge insertions and edge deletions, maintains whether φ is satisfied in G. The data structure is required to correctly report the outcome only when the feedback vertex number of G does not exceed a fixed constant k, otherwise it reports that the feedback vertex number is too large. With this assumption, we guarantee amortized update time $\mathcal{O}_{\varphi,k}(\log n)$.

By combining this result with a classic theorem of Erdős and Pósa, we give a fully dynamic data structure that maintains whether a graph contains a packing of k vertex-disjoint cycles with amortized update time $\mathcal{O}_k(\log n)$. Our data structure also works in a larger generality of relational structures over binary signatures.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases feedback vertex set, CMSO_2 formula, data structure, dynamic graphs, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.46

Related Version *Full Version*: `https://arxiv.org/abs/2107.06232` [13]

Funding 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).

1 Introduction

We consider data structures for graphs in a fully dynamic model, where the considered graph can be updated by the following operations: add an edge, remove an edge, add an isolated vertex, and remove an isolated vertex. Most of the contemporary work on data structures for graphs focuses on problems that in the static setting are polynomial-time solvable, such as connectivity or distance computation. In this work we follow a somewhat different direction and consider *parameterized problems*. That is, we consider problems that are NP-hard in the classic sense, even in the static setting, and we would like to design efficient dynamic data structures for them. The update time guarantees will typically depend on the size of the graph n and a parameter of interest k, and the goal is obtain as good dependence on n as possible while allowing exponential (or worse) dependence on k. The idea behind this approach is that the data structure will perform efficiently on instances where the parameter k is small, which is exactly the principle assumed in the field of parameterized complexity.

© Konrad Majewski, Michał Pilipczuk, and Marek Sokołowski;
licensed under Creative Commons License CC-BY 4.0

The systematic investigation of such *parameterized dynamic data structures* was initiated by Alman et al. [1], though a few earlier results of this kind can be found in the literature, e.g. [6, 7, 11]. Alman et al. revisited several techniques in parameterized complexity and developed their dynamic counterparts, thus giving suitable parameterized dynamic data structures for a number of classic problems, including Vertex Cover, Hitting Set, k-Path, and Feedback Vertex Set. The last example is important for our motivation. Recall that a *feedback vertex set* in an (undirected) graph G is a subset of vertices that intersects every cycle in G, and the *feedback vertex number* of G is the smallest size of a feedback vertex set in G. The data structure of Alman et al. monitors whether the feedback vertex number of a dynamic graph G is at most k (and reports a suitable witness, if so) with amortized update time $2^{\mathcal{O}(k \log k)} \cdot \log n$.

Dvořák et al. [6] and, more recently, Chen et al. [3] studied parameterized dynamic data structures for another graph parameter *treedepth*. Formally, the treedepth of a graph G is the least possible height of an *elimination forest* G: a rooted forest on the vertex set of G such that every edge of G connects a vertex with its ancestor. Intuitively, that a graph G has treedepth d means that G has a tree decomposition whose *height* is d, rather than width. Chen et al. [3] proved that in a dynamic graph of treedepth at most d, an optimum-height elimination forest can be maintained with update time $2^{\mathcal{O}(d^2)}$ (worst case, under the promise that the treedepth never exceeds d). This improved upon the earlier result of Dvořák et al. [6], who for the same problem achieved update time $f(d)$ for a non-elementary function f.

As already observed by Dvořák et al. [6], such a data structure can be used not only to the concrete problem of computing the treedepth, but more generally to maintaining satisfiability of any property that can be expressed in the *Monadic Second-Order logic* MSO$_2$. This logic extends standard First-Order logic FO by allowing quantification over subsets of vertices and subsets of edges, so it is able to express through constant-size sentences NP-hard problems such at Hamiltonicity or 3-colorability. More precisely, the following result was proved by Dvořák et al. [6] (see Chen et al. [3] for lifting the promise of boundedness of treedepth).

▶ **Theorem 1** ([3, 6]). *Given an* MSO$_2$ *sentence φ over the signature of graphs and $d \in \mathbb{N}$, one can construct a dynamic data structure that maintains whether a given dynamic graph G satisfies φ. The data structure is obliged to report a correct answer only when the treedepth of G does not exceed d, and otherwise it reports* Treedepth too large. *The updates work in amortized time $f(\varphi, d)$ for a computable function f, under the assumption that one is given access to a dictionary on the edges of G with constant-time operations.*

The proof of Theorem 1 is based on the following idea. If a graph G is supplied with an elimination forest of bounded depth, then, by the finite-state properties of MSO$_2$, whether φ is satisfied in G can be decided using a suitable bottom-up dynamic programming algorithm. Then it is shown that when G is updated by edge insertions and removals, one is able to maintain not only an optimum-height elimination forest F of G, but also a run of this dynamic programming algorithm on F. This blueprint brings the classic work on algorithmic meta-theorems in parameterized complexity to the setting of dynamic data structures, by showing that dynamic maintenance of a suitable decomposition is a first step to maintaining all properties that can be efficiently computed using this decomposition.

Notably, Chen et al. [3] apply this principle to two specific problems of interest: detection of k-paths and $(\geqslant k)$-cycles in undirected graphs. Using known connections between these objects and treedepth, they gave dynamic data structures for the detection problems that have update time $2^{\mathcal{O}(k^2)}$ for k-paths (assuming a dictionary on edges) and $2^{\mathcal{O}(k^4)} \cdot \log n$ for $(\geqslant k)$-cycles.

One of the main questions left open by the work of Dvořák et al. [6] and by Chen et al. [3] is whether in a dynamic graph of treewidth at most k it is possible to maintain a tree decomposition of width at most $f(k)$ with polylogarithmic update time. Note here that the setting of tree decompositions is the natural context in which MSO_2 on graphs is considered, due to Courcelle's Theorem [4], while the treedepth of a graph is always an upper bound on its treewidth. Thus, the works of Dvořák et al. [6] and of Chen et al. [3] can be regarded as partial progress towards resolving this question, where a weaker (larger) parameter treedepth is considered.

Our contribution. We approach the question presented above from another direction, by considering feedback vertex number – another parameter that upper bounds the treewidth. As mentioned, Alman et al. [1] have shown that there is a dynamic data structure that monitors whether the feedback vertex number is at most k with update time $2^{\mathcal{O}(k \log k)} \cdot \log n$. We extend this result by showing that in fact, every MSO_2-expressible property can be efficiently maintained in graphs of bounded feedback vertex number. Here is our main result.

▶ **Theorem 2.** *Given a sentence φ of CMSO_2 over the signature of graphs and $k \in \mathbb{N}$, one can construct a data structure that maintains whether a given dynamic graph G satisfies φ. The data structure is obliged to report a correct answer only if the feedback vertex number of G is at most k, otherwise it reports* Feedback vertex number too large. *The graph is initially empty and the amortized update time is $f(\varphi, k) \cdot \log n$, for some computable function f.*

Here, CMSO_2 is an extension of MSO_2 by modular counting predicates; this extends the generality slightly. Similarly as noted by Chen et al. [3], the appearance of the $\log n$ factor in the update time seems necessary: a data structure like the one in Theorem 2 could be easily used for connectivity queries in dynamic forests, for which there is an $\Omega(\log n)$ lower bound in the cell-probe model [15].

We prove Theorem 2 in a larger generality of relational structures over binary signatures, for a formal statement we refer the reader to the full version of the paper (Theorem 4.1). More precisely, we consider relational structures over signatures consisting of relation symbols of arity at most 2 that can be updated by adding and removing tuples from the relations, and by adding and removing isolated elements of the universe. In this language, graphs correspond to structures over a signature consisting of one binary relation signifying adjacency. As feedback vertex number we consider the feedback vertex number of the Gaifman graph of the structure. Generalization to relational structures is not just a mere extension of Theorem 2, it is actually a formulation that appears naturally in the inductive strategy that is employed in the proof.

As for this proof, we heavily rely on the approach used by Alman et al. [1] for monitoring the feedback vertex number. This approach is based on applying two types of simplifying operations, in alternation and a bounded number of times:

- contraction of subtrees in the graph; and
- removal of high-degree vertices.

We prove that in both cases, while performing the simplification it is possible to remember a bounded piece of information about each of the simplified parts, thus effectively enriching the whole data structure with information from which the satisfaction of φ can be inferred. Notably, for the contracted subtrees, this piece of information is the CMSO_2-type of appropriately high rank. To maintain these types in the dynamic setting, we use the top trees data structure of Alstrup et al. [2]. All in all, while our data structure is based on the same combinatorics of the feedback vertex number, it is by no means a straightforward lift of

the work of Alman et al. [1]: enriching the data structure with information about types requires several new ideas and insights, both on the algorithmic and on the logical side of the reasoning. A more extensive discussion can be found in Section 2.

Applications. Similarly as in the work of Chen et al. [3], we observe that Theorem 2 can be used to obtain dynamic data structures for specific parameterized problems through a win/win approach. Consider the *cycle packing number* of a graph G: the maximum number of vertex-disjoint cycles that can be found in G. A classic theorem of Erdős and Pósa [9] states that there exists a universal constant c such that if the feedback vertex number of a graph G is larger than $c \cdot p \log p$, then the cycle packing number of G is at least p. We can use this result to establish the following.

▶ **Theorem 3.** *For a given $p \in \mathbb{N}$ one can construct a dynamic data structure that for a dynamic graph G (initially empty) maintains whether the cycle packing number of G is at least p. The amortized update time is $f(p) \cdot \log n$, for a computable function f.*

Proof. For a given p, it is straightforward to write a CMSO_2 sentence φ_p that holds in a graph G if and only if G contains p vertex-disjoint cycles. Then we may use the data structure of Theorem 2 for φ_p and $k = c \cdot p \log p$, where c is the constant given by the theorem of Erdős and Pósa [9]. Note that if this data structure reports that *Feedback vertex number too large*, then the cycle packing number is at least p, so this outcome can be reported. ◀

The same principle can be applied to other problems related to cycle packings and feedback vertex sets, e.g. Connected Feedback Vertex Set, Independent Feedback Vertex Set, and Tree Deletion Set. We discuss these applications in the full version of the paper (Section 7).

Organization. Due to space constraints, in this extended abstract we present only an overview of our approach with the intention of explaining the main conceptual points without going into technical details. Complete proofs can be found in the full version of this work, which is attached as the appendix.

2 Overview

In this section we present an overview of the proof of Theorem 2. We deliberately keep the description high-level in order to convey the main ideas. In particular, we focus on the graph setting and delegate the notation-heavy aspects of relational structures to the full exposition.

Let G be the given dynamic graph. We focus on the model where we have a promise that the feedback vertex number of G is at most k at all times. If we are able to construct a data structure in this promise model, then it is easy to lift this to the full model described in Theorem 2 using the standard technique of *postponing invariant-breaking insertions*. This technique was also used by Chen et al. [3] and dates back to the work of Eppstein et al. [8].

Colored graphs. We will be working with edge- and vertex-colored graphs. That is, if Σ is a finite set of colors (a *palette*), then a *Σ-colored graph* is a graph where every vertex and edge is assigned a color from Σ. In our case, all the palettes will be of size bounded by functions of k and the given formula φ, but throughout the reasoning we will use different (and rapidly growing) palettes. For readers familiar with relational structures, in general we work with relational structures over binary signatures (involving symbols of arity $0, 1, 2$), which are essentially colored graphs supplied with flags.

Thus, we assume that the maintained dynamic graph G is also a Σ-colored graph for some initial palette Σ. When G is updated by a vertex or edge insertion, we assume that the color of the new feature is provided with the update.

Monadic Second-Order Logic. Logic MSO_2 is Monadic Second-Order Logic with quantification over vertex subsets and edge subsets. This is a standard logic considered in parameterized complexity in connection with treewidth and Courcelle's Theorem. We refer to [5, Section 7.4] for a thorough introduction, and explain here only the main features. There are four types of variables: *individual* vertex/edge variables that evaluate to single vertices/edges, and *monadic* vertex/edge variables that evaluate to vertex/edge subsets. These can be quantified both existentially and universally. One can check equality of vertices/edges, incidence between an edge and a vertex, and membership of a vertex/edge to a vertex/edge subset. In case of colored graphs, one can also check colors of vertices/edges using unary predicates. Negation and all boolean connectives are allowed.

Note that in Theorem 2 we consider logic CMSO_2, which is an extension of the above by modular counting predicates that can be applied to monadic variables. For simplicity, we ignore this extension for the purpose of this overview.

Types. The key technical ingredient in our reasoning are *types*, which is a standard tool in model theory. We refer to the work of Grohe [10] for a more thorough introduction. Let G be a Σ-colored graph and q be a nonnegative integer. With G we can associate its *rank-q type* $\mathsf{tp}^q(G)$, which is a finite piece of data that contains all information about the satisfaction of MSO_2 sentences of quantifier rank at most q in G (i.e., with quantifier nesting bounded by q). More precisely:

- For every choice of q and Σ there is a finite set $\mathsf{Types}^{q,\Sigma}$ containing all possible rank-q types of Σ-colored graphs. The size of $\mathsf{Types}^{q,\Sigma}$ depends only on q and Σ.
- For every MSO_2 sentence ψ of quantifier rank at most q, the type $\mathsf{tp}^q(G)$ uniquely determines whether ψ holds in G.

In addition to the above, we also need an understanding that types are *compositional* under gluing of graphs along small boundaries. For this, we work with the notion of a *boundaried graph*, which is a graph G together with a specified subset of vertices ∂G, called the *boundary*. Typically, these boundaries will be of constant size. We extend the notion of a type to boundaried graphs, where the rank-q type $\mathsf{tp}^q(G)$ of a boundaried graph G contains information not only about all rank-q MSO_2 sentences satisfied in G, but also about all such sentences that in addition can use the vertices of ∂G as parameters (one can also think that vertices of ∂G are given through free variables). Again, for every finite set D, there is a finite set of possible types $\mathsf{Types}^{q,\Sigma}(D)$ of boundaried Σ-colored graphs with boundary D, and the size of $\mathsf{Types}^{q,\Sigma}(D)$ depends only on q, Σ, and $|D|$.

Now, on boundaried graphs there are two natural operations. First, if G is a boundaried graph and $u \in \partial G$, then one can *forget* u in G. This yields a boundaried graph $\mathsf{forget}(G, u)$ obtained from G by removing u from the boundary (otherwise the graph remains intact). Second, if G and H are two boundaried graphs and ξ is a partial bijection between ∂G and ∂H, then the *join* $G \oplus_\xi H$ is the boundaried graph obtained from the disjoint union of G and H by identifying vertices that correspond to each other in ξ; the new boundary is the union of the old boundaries (with identification applied).

With these notions in place, the compositionality of types can be phrased as follows:

- Given $\mathsf{tp}^q(G)$ and $u \in \partial G$, one can uniquely determine $\mathsf{tp}^q(\mathsf{forget}(G, u))$.
- Given $\mathsf{tp}^q(G)$ and $\mathsf{tp}^q(H)$ and a partial bijection ξ between the boundaries of G and H, one can uniquely determine $\mathsf{tp}^q(G \oplus_\xi H)$.

The determination described above is effective, that is, can be computed by an algorithm.

Top trees. We now move to the next key technical ingredient: the *top trees* data structure of Alstrup et al. [2]. Top trees work over a dynamic forest F, which is updated by edge insertions and deletions (subject to the promise that no update breaks acyclicity) and insertions and deletions of isolated vertices. For each connected component T of F one maintains a *top tree* Δ_T, which is a hierarchical decomposition of T into *clusters*. Each cluster S is a subtree of T with at least one edge that is assigned a boundary $\partial S \subseteq V(S)$ of size at most 2 with the following property: every vertex of S that has a neighbor outside of S belongs to ∂S. Formally, the top tree Δ_T is a binary tree whose nodes are assigned clusters in T so that:

- the root of Δ_T is assigned the cluster $(T, \partial T)$, where ∂T is a choice of at most two vertices in T;
- the leaves of Δ_T are assigned single-edge clusters;
- for every internal node x of Δ_T, the edge sets of clusters in the children of x form a partition of the edge set of the cluster at x.

Note that the last property implies that the cluster at x, treated as a boundaried graph, can be obtained from the two clusters at the children of x by applying the join operation, possibly followed by forgetting a subset of the boundary. We will then say that the cluster at x is obtained by *joining* the two clusters at its children.

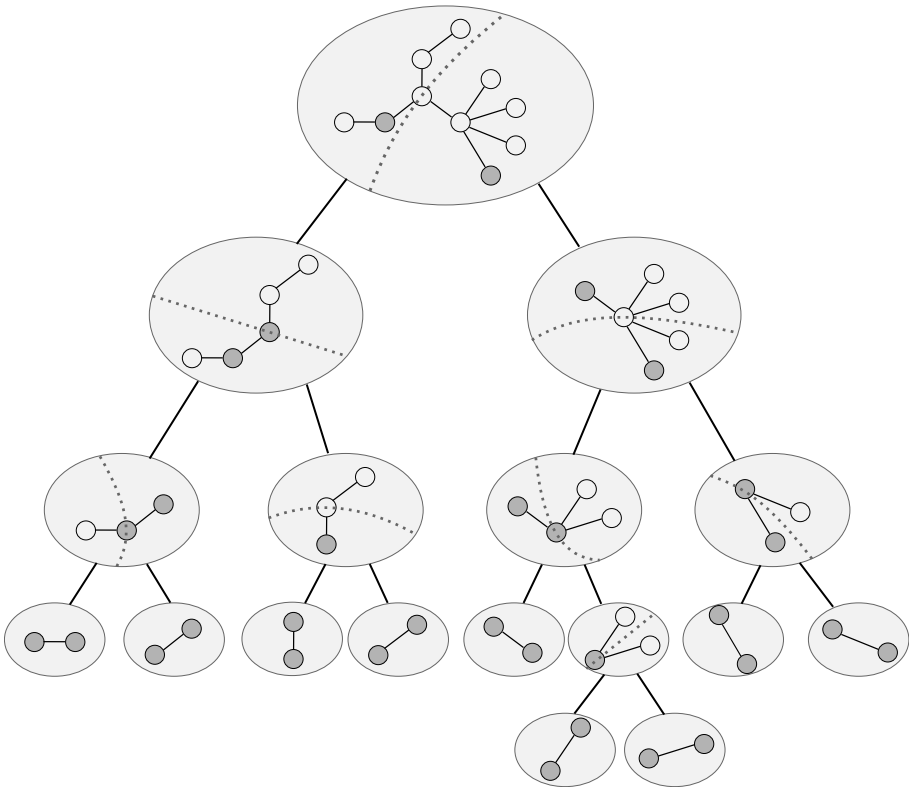


Figure 1 An example top tree Δ_T. Clusters correspond to light gray ovals. Boundary vertices in each cluster are marked dark gray. Note that in this example, Δ_T has two external boundary vertices. However, it may have fewer (zero or one) such vertices.

In [2], Alstrup et al. showed how to maintain, for a dynamic forest F, a forest of top trees $\{\Delta_T : T \text{ is a component of } F\}$ so that each tree Δ_T has depth $\mathcal{O}(\log n)$ and every operation is performed in worst-case time $\mathcal{O}(\log n)$. Moreover, they showed that the top trees data

structure can be robustly enriched with various kinds of auxiliary information about clusters, provided this information can be efficiently composed upon joining clusters. More precisely, suppose that with each cluster C we can associate a piece of information $\mathcal{I}(C)$ so that

- $\mathcal{I}(C)$ can be computed in constant time when C has one edge; and
- if C is obtained by joining two clusters C_1 and C_2, then from $\mathcal{I}(C_1)$ and $\mathcal{I}(C_2)$ one can compute $\mathcal{I}(C)$ in constant time.

Then, as shown in [2], with each cluster C one can store the corresponding piece of information $\mathcal{I}(C)$, and still perform updates in time $\mathcal{O}(\log n)$.

In our applications, we work with top trees over dynamic Σ-colored forests, where with each cluster C we store information on its type:

$$\mathcal{I}(C) = \mathsf{tp}^p(C)$$

for a suitably chosen $p \in \mathbb{N}$. Here, for technical reasons we need to be careful about the colors: the type $\mathsf{tp}^p(C)$ takes into account the colors of all the edges of C and all the vertices of C *except* the vertices of ∂C (formally, we consider the type of C with colors stripped from boundary vertices). The rationale behind this choice is that a single vertex u can participate in the boundary of multiple clusters, hence in the dynamic setting we cannot afford to update the type of each of them upon updating the color of u. Rather, every cluster C stores its type with the colors on ∂C stripped, and if we wish to compute the type of C with these colors included, it suffices to look up those colors and update the stripped type (using compositionality).

Brushing these technical details aside, after choosing the definitions right, the compositionality of types explained before perfectly fits the properties required from an enrichment of top trees. This means that with each cluster C we can store $\mathsf{tp}^p(C)$ while guaranteeing worst-case update time $\mathcal{O}_{p,\Sigma}(\log n)$. We remark that the combination of top trees and MSO_2 types appears to be a novel contribution of this work; we hope that it can be reused in the future.

So if F is a dynamic Σ-colored forest and p is a parameter, then for each tree T in F we can maintain a top tree Δ_T whose root is supplied with the type $\mathsf{tp}^p(T)$. Knowing the multiset of rank-p types of trees in F, we can use standard compositionality and idempotence of types to compute the type $\mathsf{tp}^p(F)$, from which in turn one can infer which rank-p sentences are satisfied in F. By taking p to be the quantifier rank of a given sentence φ, we obtain:

▶ **Theorem 4.** *Let Σ be a finite palette and φ be an MSO_2 sentence over Σ-colored graphs. Then there is a dynamic data structure that for a dynamic Σ-colored forest F maintains whether φ holds in F. The worst-case update time is $\mathcal{O}_{\varphi,\Sigma}(\log n)$.*

Note that the statement of Theorem 4 matches (the colored version of) the statement of Theorem 2 for $k = 0$. Curiously, we are not aware of this result existing already in the literature, despite the naturality of the problem. We remark that maintaining MSO queries over dynamic forests has been considered in the databases literature, see [14] and references therein, however under a different (and somewhat orthogonal) set of allowed updates.

The data structure of Alman et al. [1]. Our goal now is to lift Theorem 4 to the case of $k > 0$. For this we rely on the approach of Alman et al. [1] for monitoring the feedback vertex number, which is based on a sparsity-based strategy that is standard in parameterized complexity, see e.g. [5, Section 3.3].

The approach is based on two lemmas. The first one concerns the situation when the graph contains a vertex u of degree at most 2. In this case, it is safe to dissolve u: either remove it, in case it has degree 0 or 1, or replace it with a new edge connecting its neighbors, in case it

has degree 2. Note that dissolving a degree-2 vertex naturally can create a multigraph. This creates technical issues both in [1] and in this work, but we shall largely ignore them for the purpose of this overview. Formally, we have the following.

▶ **Lemma 5** (folklore). *Dissolving a vertex of degree at most* 2 *in a multigraph does not change the feedback vertex number.*

The second lemma concerns the situation when the graph has minimum degree at least 3. Then a sparsity-based argument shows that every feedback vertex set of size at most k intersects the set of $\mathcal{O}(k)$ vertices with highest degrees.

▶ **Lemma 6** (Lemma 3.3 in [5]). *Let G be a multigraph with minimum degree* 3 *and let B be the set of $3k$ vertices with highest degrees in G. Then every feedback vertex set of size at most k in G intersects B.*

Lemmas 5 and 6 can be used to obtain an fpt algorithm for Feedback Vertex Set with running time $(3k)^k \cdot (n+m)$ (see [5, Theorem 3.5]): apply the reduction of Lemma 5 exhaustively, and then branch on which of the $3k$ vertices with highest degrees should be included in the solution. This results in a recursion tree of total size at most $(3k)^k$.

The data structure of Alman et al. [1] is based on dynamization of the branching algorithm presented above. There are two main challenges:

- dynamic maintenance of the sequence of dissolutions given by Lemma 5; and
- dynamic maintenance of the set of high degree vertices.

For the first issue, it is explanatory to imagine performing the dissolutions not one by one iteratively, but all at once. It is not hard to see that the result of applying Lemma 5 exhaustively is that the input multigraph G gets contracted to a multigraph $\mathsf{Contract}(G)$ in the following way: the edge set of G into disjoint trees, and each of them either disappears or is contracted into a single edge in $\mathsf{Contract}(G)$; see Figure 2 for a visualization. (There may be some corner cases connected to loops in $\mathsf{Contract}(G)$ that result from contracting not trees, but unicyclic graphs; we ignore this issue in this overview.) We call the elements of this partition *ferns*, and the corresponding decomposition of G into ferns is called the *fern decomposition* of G. Importantly, the order of performing the contractions has no effect on the outcome, yielding always the same fern decomposition of G.

With each fern of S we can associate its boundary ∂S, which is the set of vertices of S incident to edges that lie outside of S. It is not hard to see that this boundary will always be of size 0, 1, or 2. The ferns that correspond to edges in $\mathsf{Contract}(G)$ are the ferns with boundary of size 2 (each such fern gets contracted to an edge connecting the two vertices of the boundary) and non-tree ferns with boundary of size 1 (each such fern gets contracted to a loop at the unique vertex of the boundary).

The idea of Alman et al. is to maintain the ferns in the fern decomposition using link-cut trees. It is shown that each update in G affects the fern decomposition only slightly, in the sense that it can be updated using a constant number of operations on link-cut trees. In this way, the fern decomposition and the graph $\mathsf{Contract}(G)$ can be maintained with worst-case $\mathcal{O}(\log n)$ time per update in G. This resolves the first challenge.

For the second challenge, Alman et al. observe that if in Lemma 6 one increases the number of highest degree vertices included in B from $3k$ to $12k$, then the set remains "valid" – in the sense of satisfying the conclusion of the lemma – even after $\mathcal{O}(m/k)$ updates are applied to the graph. Here, m denotes the number of edges of the graph on which Lemma 6 is applied, which is $\mathsf{Contract}(G)$ in our case. This means that it remains correct to perform a recomputation of the set B only every $\Theta(m/k)$ updates. Since such a recomputation takes time $\mathcal{O}(m)$, the amortized update time is $\mathcal{O}(k)$.

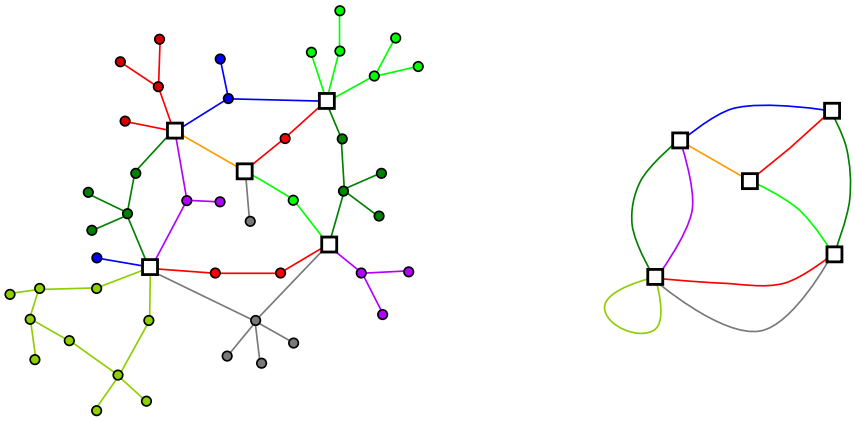


Figure 2 Left: A graph G together with its fern decomposition. Different ferns are depicted with different colors; these should not be confused with the coloring of edges of G with colors from Σ. Right: The multigraph $\mathsf{Contract}(G)$ obtained by contracting each fern. Note that in the construction of $\mathsf{Contract}^p(G)$ described in the discussion of the Contraction Lemma, we would not have parallel edges or loops. Instead, each pack of parallel edges would be replaced by a single one, colored with the joint type of the whole pack. Similarly, loops on a vertex would be removed and their joint type would be stored in the color of the vertex.

Once $\mathsf{Contract}(G)$ and $B \subseteq V(\mathsf{Contract}(G))$ are known, Lemma 6 asserts that if the feedback vertex number of G is at most k, there exists a vertex $b \in B$ whose deletion decreases the feedback vertex number. Therefore, the idea of Alman et al. is to construct a recursive copy of the data structure for each $b \in B$: the copy maintains the graph $\mathsf{Contract}(G) - b$ and uses parameter $k-1$ instead of k. Note that when B gets recomputed, all these data structures are reset, but thanks to amortization we have time to do it.

All in all, once one unravels the recursion, the whole construction is a tree of data structures of depth k and branching $12k$, which is maintained with amortized update time $2^{\mathcal{O}(k \log k)} \cdot \log n$. The graph has feedback vertex number at most k if and only if this tree contains at least one leaf with an empty graph.

Our data structure. We now describe the high-level idea of our data structure.

Lemmas 5 and 6 can be used not only to design an fpt algorithm for Feedback Vertex Set, but also an approximation algorithm. Consider the following procedure: apply the reduction of Lemma 5 exhaustively, then greedily take *all* the $3k$ vertices with highest degrees to the constructed feedback vertex set, and iterate these two steps in alternation until the graph becomes empty. Lemma 6 guarantees that provided the feedback vertex number was at most k in the first place, the iteration terminates after at most k steps; the $3k^2$ selected vertices form a feedback vertex set. We note that this application of Lemmas 5 and 6 for feedback vertex set approximation is not new, for instance it was recently used by Kammer and Sajenko [12] in the context of space-efficient kernelization.

Our data structure follows the design outlined above. That is, instead of a tree of data structures, we maintain a sequence of $2k+2$ data structures, respectively for multigraphs

$$G_0, H_0, G_1, H_1, \ldots, G_k, H_k.$$

These multigraphs essentially satisfy the following:

- $G_0 = G$;
- $H_i = \mathsf{Contract}(G_i)$ for $i = 0, 1, \ldots, k$; and
- $G_{i+1} = G_i - B_i$ for $i = 0, 1, \ldots, k-1$, where B_i is a set that satisfies the conclusion of Lemma 6 for G_i.

Note that these invariants imply that provided the feedback vertex number of G is at most k, the feedback vertex number of G_i and of H_i is at most $k-i$ for each $i \in \{0,1,\ldots,k\}$, implying that G_k is a forest and H_k is the empty graph.

The precise definitions of $\mathsf{Contract}(\cdot)$ and of deleting vertices used in the sequence above will be specified later. More precisely, graphs $G_0, H_0, \ldots, G_k, H_k$ will be colored with palettes $\Sigma_0, \Gamma_0, \ldots, \Sigma_k, \Gamma_k$ in order, where $\Sigma_0 = \Sigma$. These palettes will grow (quite rapidly) in sizes, but each will be always of size bounded in terms of k, Σ, and q – the quantifier rank of the fixed sentence φ whose satisfaction we monitor. The idea is that when obtaining H_i from G_i by contracting ferns, we use colors from Γ_i to store information about the contracted ferns on edges and vertices of H_i. Similarly, when removing vertices of B_i from H_i to obtain G_{i+1}, we use colors from Σ_{i+1} on vertices of G_{i+1} to store information about the adjacencies of the removed vertices. These steps are encompassed by two key technical statements – the Contraction Lemma and the Downgrade Lemma – which we explain below.

Contraction Lemma. We explain the Contraction Lemma for the construction of $H \coloneqq H_0$ from $G = G_0$; the construction for $i > 0$ is the same. Recall that eventually we are interested in monitoring whether the given sentence φ is satisfied in G. For this, it is sufficient to monitor the type $\mathsf{tp}^q(G)$, where q is the quantifier rank of φ. Consider the following construction:

- Pick some large $p \in \mathbb{N}$.
- Consider the fern decomposition $\mathcal{F}$ of G and let $\mathcal{K} \coloneqq \{\partial S \colon S \in \mathcal{F}\}$. For every $D \in \mathcal{K}$, let R_D be the join of all the ferns with boundary D, and with colors stripped from the vertices of D. Note that R_D is a boundaried graph with boundary D.
- For every $D \in \mathcal{K}$ with $|D| = 2$, contract R_D to a single edge with color $\mathsf{tp}^p(R_D)$ connecting the two vertices of D.
- For every $D \in \mathcal{K}$ with $|D| = 1$, contract R_D onto the single vertex d of D, and make d of color $\mathsf{tp}^p(R_D)$.
- Remove $R_\emptyset$, if present, and remember $\mathsf{tp}^p(R_\emptyset)$ through flags[1].
- The obtained colored graph is named $\mathsf{Contract}^p(G)$. Note that $\mathsf{Contract}^p(G)$ is a Γ^p-colored graph, where Γ^p is a palette consisting of all rank-p types of Σ-colored graphs with a boundary of size at most 2.

Thus, every fern S in G is essentially disposed of, but a finite piece of information (the rank-p type) about S is being remembered in $\mathsf{Contract}^p(G)$ on the boundary of S. The intuition is that if p is large enough, these pieces of information are enough to infer the rank-q type of G. This intuition is confirmed by the following Replacement Lemma.

▶ **Lemma 7** (Replacement Lemma, informal statement). *For any given $q \in \mathbb{N}$ and Σ, there exists $p \in \mathbb{N}$ large enough so that for any Σ-colored graph G, the type $\mathsf{tp}^p(\mathsf{Contract}^p(G))$ uniquely determines the type $\mathsf{tp}^q(G)$.*

The proof of the Replacement Lemma uses Ehrenfeucht-Fraïsse games. It is conceptually rather standard, but technically quite involved. We note that the obtained constant p is essentially the number of rank-q types of Σ-colored graphs, which is approximately a tower of exponentials of height q applied to $|\Sigma|$. Since Replacement Lemma is used k times in the construction, this incurs a huge explosion in the parameter dependence in our data structure.

Replacement Lemma shows that in order to monitor the type $\mathsf{tp}^q(G)$ in the dynamic setting, it suffices to maintain the graph $H \coloneqq \mathsf{Contract}^p(G)$ and the type $\mathsf{tp}^p(H)$. Maintaining H dynamically is the responsibility of the Contraction Lemma.

[1] We assume that a colored graph can be supplied with a bounded number of boolean flags, which thus can store a bounded amount of additional information. In the general setting of relational structures, flags are modeled by nullary predicates (predicates of arity 0).

▶ **Lemma 8** (Contraction Lemma, informal statement). *For a given $p \in \mathbb{N}$ and palette Σ, there is a dynamic data structure that for a dynamic graph G, maintains the graph* $\mathsf{Contract}^p(G)$ *under updates in G. The worst-case update time is $\mathcal{O}_{p,\Sigma}(\log n)$.*

The proof of Lemma 8 follows closely the reasoning of Alman et al. [1]. That is, in the same way as in [1], every update in G incurs a constant number of changes in the fern decomposition of G, expressed as splitting or merging of individual ferns. Instead of relying on link-cut trees as in [1], the ferns are stored using top trees. This is because we enrich the top trees data structure with the information about rank-p types of clusters, as in Theorem 4, so that for each fern S we know its rank-p type. This type is needed to determine the color of the feature (edge/vertex/flag) in $H = \mathsf{Contract}^p(G)$ to which S contributes.

Executing the plan sketched above requires an extreme care about details. Note for instance that in the construction of $\mathsf{Contract}^p(G)$, when defining R_D we explicitly stripped colors from the boundary vertices. This is for a reason similar to that discussed alongside Theorem 4: including the information on the colors of D in $\mathsf{tp}^p(R_D)$ would mean that a single update to the color of a vertex d would affect the types of all subgraphs R_D with $d \in D$, and there is potentially an unbounded number of such subgraphs. Further, we remark that Alman et al. [1] relied on an understanding of the fern decomposition through a sequence of dissolutions, which makes some arguments inconvenient for generalization to our setting. We need a firmer grasp on the notion of fern decomposition, hence we introduce a robust graph-theoretic description that is static – it does not rely on an iterative dissolution procedure. This robustness helps us greatly in maintaining ferns and their types in the dynamic setting.

Another noteworthy technical detail is that the operator $\mathsf{Contract}^p(\cdot)$, as defined above, does not create parallel edges or loops, and thus we stay within the realm of colored simple graphs (or, in the general setting, of classic relational structures over binary signatures). Unfortunately, this simplification cannot be applied throughout the whole proof, as in Lemma 6 we need to count the degrees with respect to the multigraph $\mathsf{Contract}(G)$ as defined in Alman et al. [1]. For this reason, in the full proof we keep trace of two objects at the same time: a relational structure $\mathbb{A}$ that we are interested in, and a multigraph H which is a supergraph of the Gaifman graph of $\mathbb{A}$ and that represents the structure of earlier contractions.

Downgrade Lemma. Finally, we are left with the Downgrade Lemma, which is responsible for the reducing the graph by removing a bounded number of vertices. Formally, we have a Γ-colored graph H and a set B of $\mathcal{O}(k)$ vertices, and we would like to construct a Σ'-colored graph $G' = \mathsf{Downgrade}(H, B)$ by removing the vertices of B and remembering information about them on the remaining vertices of H. This construction is executed as follows:

- Enumerate the vertices of B as $b_1, \ldots, b_\ell$, where $\ell = |B|$.
- Construct G' by removing vertices of B.
- For every color $c \in \Gamma$ and $i \in \{1, \ldots, \ell\}$, add to G' a flag signifying whether b_i has color c in G.
- For every pair $i, j \in \{1, \ldots, \ell\}$, $i < j$, and every color $c \in \Gamma$ add to G' a flag signifying whether b_i and b_j are connected in G by an edge of color c.
- For every vertex $u \in V(G) \setminus B$, every $i \in \{1, \ldots, \ell\}$, and every color $c \in \Gamma$, refine the color of u in G' by adding the information on whether u and b_i were connected in G by an edge of color c.
- The obtained graph is the graph G'. Note that G' is Σ'-colored, where $\Sigma' = \Gamma \times 2^{[\ell] \times \Gamma}$.

Thus, the information about vertices of B and edges incident to B is being stored in flags and colors on vertices of $V(G) \setminus B$. We have the following analogue of the Replacement Lemma.

▶ **Lemma 9.** *For any given $p \in \mathbb{N}$, there exists $q' \in \mathbb{N}$ large enough so that for any Γ-colored graph H and a subset B of $\mathcal{O}(k)$ vertices, the type $\mathsf{tp}^{q'}(\mathsf{Downgrade}(H, B))$ uniquely determines $\mathsf{tp}^{p}(H)$.*

The proof of Lemma 9 is actually very simple and boils down to a syntactic modification of formulas. From Lemma 9 it follows that to maintain the type $\mathsf{tp}^p(H)$, it suffices to maintain a bounded-size set B satisfying the conclusion of Lemma 6, the graph $G' = \mathsf{Downgrade}(H, B)$, and its type $\mathsf{tp}^{q'}(G')$. This is the responsibility of the Downgrade Lemma.

▶ **Lemma 10** (Downgrade Lemma, informal statement)**.** *For a given $p \in \mathbb{N}$ and palette Γ, there is a dynamic data structure that for a dynamic graph H of feedback vertex number at most k and with minimum degree 3, maintains a set of vertices $B \subseteq V(H)$ with $|B| \leqslant 12k$ and satisfying the conclusion of Lemma 6, and the graph $\mathsf{Downgrade}(H, B)$. The amortized update time is $\mathcal{O}_{p,\Gamma,k}(\log n)$.*

The proof of the Downgrade Lemma is essentially the same as that given for the corresponding step in Alman et al. [1]. We recompute B from scratch every $\Theta(m/k)$ updates, because the argument of Alman et al. shows that B remains valid for this long. Recomputing B implies recomputing $\mathsf{Downgrade}(H, B)$ in $\mathcal{O}_{p,\Gamma,k}(m)$ time, so the amortized complexity is $\mathcal{O}_{p,\Gamma,k}(1)$ (there are additional logarithmic factors from auxiliary data structures).

Endgame. We now have all the pieces to assemble the proof of Theorem 2. Let q_0 be the quantifier rank of the given sentence φ and let $G_0 = G$ be the considered dynamic graph. By Replacement Lemma, to monitor $\mathsf{tp}^{q_0}(G_0)$ (from which the satisfaction of φ can be inferred), it suffices to monitor $\mathsf{tp}^{p_0}(H_0)$, where $H_0 \coloneqq \mathsf{Contract}^{p_0}(G_0)$ and p_0 is as provided by the Replacement Lemma. By Contraction Lemma, we can efficiently maintain H_0 under updates of G_0. By Lemma 9, to monitor $\mathsf{tp}^{p_0}(H_0)$ it suffices to monitor $\mathsf{tp}^{q_1}(G_1)$, where $G_1 \coloneqq \mathsf{Downgrade}(H_0, B_0)$, and B_0 is a set that satisfies the conclusion of Lemma 6. By Downgrade Lemma, we can efficiently maintain such a set B_0 and the graph G_1. We proceed further in this way, alternating the usage of the Contraction Lemma and the Downgrade Lemma. Observe that each application of Downgrade Lemma strictly decrements the feedback vertex number, so after k steps we end up with an empty graph H_k. The type of this graph can be directly computed from its flags, and this type can be translated back to infer $\mathsf{tp}^q(G)$ by using Replacement Lemma and Lemma 9 alternately.

References

1 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic Parameterized Problems and Algorithms. *ACM Trans. Algorithms*, 16(4):45:1–45:46, 2020. doi:10.1145/3395037.

2 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005. doi:10.1145/1103963.1103966.

3 Jiehua Chen, Wojciech Czerwinski, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartlomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.

4 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.

5 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.

6 Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *Proceedings of the 22nd Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014. doi:10.1007/978-3-662-44777-2_28.

7 Zdeněk Dvořák and Vojtěch Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *Proceedings of the 13th International Symposium on Algorithms and Data Structures, WADS 2013*, volume 8037 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 2013. doi:10.1007/978-3-642-40104-6_27.

8 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. Planary testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996. doi:10.1006/jcss.1996.0002.

9 Paul Erdős and Lajos Pósa. On the maximal number of disjoint circuits of a graph. *Publ. Math. Debrecen*, 9:3–12, 1962.

10 Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2008.

11 Yoichi Iwata and Keigo Oka. Fast dynamic graph algorithms for parameterized problems. In *Proceedings of the 14th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2014*, volume 8503 of *Lecture Notes in Computer Science*, pages 241–252. Springer, 2014. doi:10.1007/978-3-319-08404-6_21.

12 Frank Kammer and Andrej Sajenko. FPT-space graph kernelizations. *CoRR*, abs/2007.11643, 2020. arXiv:2007.11643.

13 Konrad Majewski, Michal Pilipczuk, and Marek Sokolowski. Maintaining CMSO_2 properties on dynamic structures with bounded feedback vertex number. *CoRR*, abs/2107.06232, 2021. arXiv:2107.06232.

14 Matthias Niewerth. MSO queries on trees: Enumerating answers under updates using forest algebras. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 769–778. ACM, 2018. doi:10.1145/3209108.3209144.

15 Mihai Pătraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, STOC 2004*, pages 546–553. ACM, 2004.

Sublinear-Time Probabilistic Cellular Automata

Augusto Modanese ✉
Aalto University, Espoo, Finland

Abstract

We propose and investigate a probabilistic model of sublinear-time one-dimensional cellular automata. In particular, we modify the model of ACA (which are cellular automata that accept if and only if all cells simultaneously accept) so that every cell changes its state not only dependent on the states it sees in its neighborhood but also on an unbiased coin toss of its own. The resulting model is dubbed *probabilistic ACA* (PACA). We consider one- and two-sided error versions of the model (in the same spirit as the classes RP and BPP) and establish a separation between the classes of languages they can recognize all the way up to $o(\sqrt{n})$ time. As a consequence, we have a $\Omega(\sqrt{n})$ lower bound for derandomizing constant-time one-sided error PACAs (using deterministic ACAs). We also prove that derandomization of $T(n)$-time PACAs (to polynomial-time deterministic cellular automata) for various regimes of $T(n) = \omega(\log n)$ implies non-trivial derandomization results for the class RP (e.g., $\mathsf{P} = \mathsf{RP}$). The main contribution is an almost full characterization of the constant-time PACA classes: For one-sided error, the class equals that of the deterministic model; that is, constant-time one-sided error PACAs can be fully derandomized with only a constant multiplicative overhead in time complexity. As for two-sided error, we identify a natural class we call the *linearly testable languages* (LLT) and prove that the languages decidable by constant-time two-sided error PACAs are "sandwiched" in-between the closure of LLT under union and intersection and the class of locally threshold testable languages (LTT).

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

Keywords and phrases Cellular automata, local computation, probabilistic models, subregular language classes

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.47

Related Version *Full Version*: https://arxiv.org/abs/2203.14614 [15]

Funding Supported by Helsinki Institute for Information Technology (HIIT). Much of this work done while affiliated with the Karlsruhe Institute of Technology (KIT).

Acknowledgements I would like to thank Thomas Worsch for the helpful discussions and feedback as well as the anonymous reviewers for their insightful comments.

1 Introduction

Cellular automata (CAs) have been extensively studied as a natural model of distributed computation. A one-dimensional CA is composed of a row of fairly limited computational agents – the *cells* – which, by interacting with their immediate neighbors, realize a global behavior and work towards a common goal.

As every model of computation, CAs have been widely studied as language acceptors [10, 20]. These efforts apparently were almost exclusively devoted to the linear- or real-time case – to the detriment of the *sublinear-time* one [14]. This is unfortunate since, as it was recently shown in [13], the study of sublinear-time CA variants might help better direct efforts in resolving outstanding problems in computational complexity theory.

In this work, we consider a *probabilistic* sublinear-time CA model. Our main goal is to analyze to what extent – if at all – the addition of randomness to the model is able to make up for its inherent limitations. (For instance, sublinear-time CA models are usually restricted to a local view of their input [14] and are also unable to cope with long unary subwords [13].)


© Augusto Modanese;
licensed under Creative Commons License CC-BY 4.0


1.1 The Model

We consider only bounded one-dimensional cellular automata.

▶ **Definition 1** (Cellular automaton). *A* cellular automaton *(CA) is a triple* $C = (Q, \$, \delta)$ *where* Q *is the finite set of* states, $\$ \notin Q$ *is the* boundary symbol, *and* $\delta\colon Q_\$ \times Q \times Q_\$ \to Q$ *is the* local transition function, *where* $Q_\$ = Q \cup \{\$\}$. *The elements in the domain of* δ *are the possible* local configurations *of the cells of* C. *For a fixed width* $n \in \mathbb{N}_+$, *the* global configurations *of* C *are the elements of* Q^n. *The cells* 0 *and* $n-1$ *are the* border cells *of* C. *The* global transition function $\Delta\colon Q^n \to Q^n$ *is obtained by simultaneous application of* δ *everywhere; that is, if* $s \in Q^n$ *is the current global configuration of* C, *then*

$$\Delta(s) = \delta(\$, s_0, s_1)\,\delta(s_0, s_1, s_2)\cdots\delta(s_{n-2}, s_{n-1}, \$).$$

For $t \in \mathbb{N}_0$, Δ^t *denotes the* t*-th iterate of* Δ. *For an initial configuration* $s \in Q^n$, *the sequence* $s = \Delta^0(s), \Delta(s), \Delta^2(s), \ldots$ *is the* orbit *of* C *(for* s*). Writing the orbit of* C *line for line yields its* space-time diagram.

One key theme connecting CAs and models of physics is *causality*: If two cells i and j are t cells away from each other, then j requires at least t steps to receive any information from i. In the *sublinear-time* case, this means every cell only gets to see a very small section of the input. In some sense this is reminiscent of *locality* in circuits (e.g., [22]), though locality in the CA model carries a more literal meaning since it is connected to the notion of space (whereas in circuits there is no equivalent notion). One should keep this limitation (of every cell only seeing a portion of the input) in mind as it is central to several of our arguments.

The usual acceptance condition for CA-based language recognizers is that of a distinguished cell (usually the leftmost one) entering an accepting state [10]. This is unsuitable for sublinear-time computation since then the automaton is limited to verifying prefixes of a constant length [14]. The most widely studied [8, 9, 14, 18] acceptance condition for sublinear-time is that of all cells simultaneously accepting, yielding the model of *ACA* (where the first "A" in the acronym indicates that *all* cells must accept).

▶ **Definition 2** (DACA). *A* deterministic ACA *(DACA) is a CA* C *with an* input alphabet $\Sigma \subseteq Q$ *as well as a subset* $A \subseteq Q$ *of* accepting states. *We say* C accepts *an input* $x \in \Sigma^+$ *if there is* $t \in \mathbb{N}_0$ *such that* $\Delta^t(x) \in A^n$, *and we denote the set of all such* x *by* $L(C)$. *In addition,* C *is said to have* time complexity *(bounded by)* $T\colon \mathbb{N}_+ \to \mathbb{N}_0$ *if, for every* $x \in L(C) \cap \Sigma^n$, *there is* $t < T(|x|)$ *such that* $\Delta^t(x) \in A^n$.

We propose a probabilistic version of the ACA model inspired by the stochastic automata of [2] and the definition of probabilistic Turing machines (see, e.g., [1]). In the model of *probabilistic ACA* (PACA), at every step, each cell tosses a fair coin $c \in \{0, 1\}$ and then changes its state based on the outcome of c. There is a nice interplay between this form of randomness access and the overall theme of *locality* in CAs: Random events pertaining to a cell i depend exclusively on what occurs in the vicinity of i. Furthermore, events corresponding to distinct cells i and j can only be dependent if i and j are near each other; otherwise, they are necessarily independent (see Lemma 11).

We consider both one- and two-sided error versions of the model as natural counterparts of RP and BPP machines, respectively. Although PACAs are a conceptually simple extension of ACAs, the definition requires certain care, in particular regarding the model's time complexity. We refer to Section 3 for the formal definitions and further discussion.

Finally, we should also mention our model is more restricted than a *stochastic CA*,[1] which is a CA in which the next state of a cell is chosen according to an arbitrary distribution that depends on the cell's local configuration. For a survey on stochastic CAs, we refer to [11].

1.2 Results

Inclusion relations. As can be expected, two-sided error PACAs are more powerful than their one-sided error counterparts. Say a DACA C is *equivalent* to a PACA C' if they accept the same language (i.e., $L(C) = L(C')$).

▶ **Theorem 3.** *The following hold:*

1. *If C is a one-sided error PACA with time complexity T, then there is an equivalent two-sided error PACA C' with time complexity $O(T)$.*
2. *There is a language L recognizable by constant-time two-sided error PACA but not by any $o(\sqrt{n})$-time one-sided error PACA.*

We stress the first item does not follow immediately from the definitions since it requires error reduction by a constant factor, which requires a non-trivial construction. It remains open whether in the second item we can improve the separation from $o(\sqrt{n})$ to $o(n)$ time. Nevertheless, as it stands the result already implies a lower bound of $\Omega(\sqrt{n})$ time for the derandomization (to a DACA) of constant-time two-sided error PACA.

Another result we show is how time-efficient derandomization of PACA classes imply derandomization results for RP (with a trade-off between the PACA time complexity and the efficiency of the derandomization).

▶ **Theorem 4.** *Let $d \geq 1$. The following hold:*

- *If there is $\varepsilon > 0$ such that every n^ε-time (one- or two-sided error) PACA can be converted into an equivalent n^d-time deterministic CA, then $\mathsf{P} = \mathsf{RP}$.*
- *If every $\mathsf{polylog}(n)$-time PACA can be converted into an equivalent n^d-time deterministic CA, then, for every $\varepsilon > 0$, $\mathsf{RP} \subseteq \mathsf{TIME}[2^{n^\varepsilon}]$.*
- *If there is $b > 2$ so that any $(\log n)^b$-time PACA can be converted into an equivalent n^d-time deterministic CA, then, for every $a \geq 1$ and $c > a/(b-1)$, $\mathsf{RTIME}[n^a] \subseteq \mathsf{TIME}[2^{O(n^c)}]$.*

We write "deterministic CA" instead of "DACA" since, for $T(n) = \Omega(n)$, a T-time DACA is equivalent to an $O(T)$-time deterministic CA with the usual acceptance condition [14].

Characterization of constant time. As a first step we analyze and almost completely characterize constant-time PACA. Indeed, the constant-time case is already very rich and worth considering in and of itself. This may not come as a surprise since other local computational models (e.g., local graph algorithms [19]) also exhibit behavior in the constant-time case that is far from trivial.

In Appendix A we give an example of a one-sided error PACA that recognizes a language L strictly faster than any DACA for L. Nonetheless, as we prove, one-sided error PACA can be derandomized with only a constant multiplicative overhead in time complexity.

▶ **Theorem 5.** *For any constant-time one-sided error PACA C, there is a constant-time DACA C' such that $L(C) = L(C')$.*

[1] Unfortunately, the literature uses the terms stochastic and probabilistic CA interchangeably. We deem "probabilistic" more suitable since it is intended as a CA version of a probabilistic Turing machine.

In turn, the class of languages accepted by constant-time two-sided error PACA can be considerably narrowed down in terms of a novel subregular class LLT, dubbed the *locally linearly testable* languages. Below, $\mathsf{LLT}_{\cup\cap}$ is the closure over LLT under union and intersection and LTT its Boolean closure (i.e., its closure under union, intersection, and complement).

▶ **Theorem 6.** *The class of languages that can be accepted by a constant-time two-sided error PACA contains* $\mathsf{LLT}_{\cup\cap}$ *and is strictly contained in* LTT*.*

It is known that the constant-time class of DACA equals the closure under union $\mathsf{SLT}_{\cup}$ of the strictly local languages SLT [18]. (We refer to Section 4.2 for the definitions.) Since $\mathsf{SLT}_{\cup} \subsetneq \mathsf{LLT}_{\cup}$ is a proper inclusion, this gives a separation of the deterministic and probabilistic classes in the case of two-sided error and starkly contrasts with Theorem 5.

As far as we are aware of, the class LLT does not previously appear in the literature.[2] In Section 4.2 we show LLT lies in-between $\mathsf{SLT}_{\cup}$ and the class of locally threshold testable languages LTT. In this regard LLT is similar to the class LT of locally testable languages; however, as we can also show, both LLT and $\mathsf{LLT}_{\cup}$ are incomparable to LT.

1.3 Organization

The rest of the paper is organized as follows: Section 2 introduces basic concepts and notation. Following that, in Section 3 we define the PACA model and prove standard error reduction results as well as Theorem 3. In Section 4 we focus on the constant-time case and prove Theorems 5 and 6. Finally, in Section 5 we address the general sublinear-time case and prove Theorem 4. We conclude with Section 6 by mentioning a few further research directions.

2 Preliminaries

It is assumed the reader is familiar with the theory of cellular automata as well as with basic notions of computational complexity theory (see, e.g., the standard references [1, 4, 6]).

All logarithms are to the base 2. The set of integers is denoted by $\mathbb{Z}$, that of non-negative integers by $\mathbb{N}_0$, and that of positive integers by $\mathbb{N}_+$. For a set S and $n, m \in \mathbb{N}_+$, $S^{n\times m}$ is the set of n-row, m-column matrices over S. For $n \in \mathbb{N}_+$, $[n] = \{i \in \mathbb{N}_0 \mid i < n\}$ is the set of the first n non-negative integers. Also, for $a, b \in \mathbb{Z}$, by $[a, b] = \{i \in \mathbb{Z} \mid a \leq i \leq b\}$ we always refer to an interval containing only integers.

Symbols in words are indexed starting with zero. The i-th symbol of a word w is denoted by w_i. For an alphabet Σ and $n \in \mathbb{N}_0$, $\Sigma^{\leq n}$ contains the words $w \in \Sigma^*$ for which $|w| \leq n$. For an infix $m \in \Sigma^{\leq |w|}$ of w, $|w|_m$ is the number of occurrences of m in w. Without restriction, the empty word is not an element of any language that we consider. (This is needed for definitional reasons; see Definitions 1 and 2 below.)

We write U_n (resp., $U_{n\times m}$) for a random variable distributed uniformly over $\{0,1\}^n$ (resp., $\{0,1\}^{n\times m}$). We also need the following variant of the Chernoff bound (see, e.g., [21]):

▶ **Theorem 7** (Chernoff bound)**.** *Let* $X_1, \ldots, X_n$ *be independently and identically distributed Bernoulli variables and* $\mu = \mathrm{E}[X_i]$*. There is a constant* $c > 0$ *such that the following holds for every* $\varepsilon = \varepsilon(n) > 0$*:*

$$\Pr\left[\left|\frac{\sum_i X_i}{n} - \mu\right| > \varepsilon\right] < 2^{-cn\varepsilon^2}.$$

[2] A similar class is perhaps found in [5, 17], but there are clear distinctions to be made between the two.

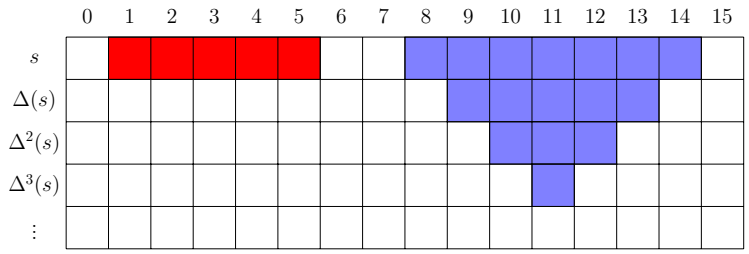


Figure 1 Space-time diagram of a CA with 16 cells for an initial configuration s. (States have been omitted for simplicity.) The cells marked in red form the 2-neighborhood of cell 3, the ones in blue the 3-lightcone of cell number 11.

Many of our low-level arguments make use of the notion of a *lightcone*.[3] For a set S and non-negative integers $n \leq m$, a lightcone $L = (\ell_{i,j})$ of *radius* m and *height* n over S is a trapezoidal (when $n < m$) or triangular (when $n = m$) array of elements $\ell_{i,j} \in S$, where $i \in [0, n]$ and $j \in [-m, m]$:

$$\begin{array}{ccccccccccc} \ell_{0,-m} & \ell_{0,-m+1} & \cdots & & \cdots & \cdots & \ell_{0,0} & \cdots & \cdots & \cdots & \ell_{0,m-1} & \ell_{0,m} \\ & \ell_{1,-m+1} & \cdots & & \cdots & \cdots & \ell_{1,0} & \cdots & \cdots & \cdots & \ell_{1,m-1} & \\ & & \ddots & & & & \vdots & & & \iddots & & \\ & & & \ell_{n,-m+n} & & \cdots & \ell_{n,0} & \cdots & \ell_{n,m-n} & & & \end{array}$$

The element $\ell_{0,0}$ is the *center* of the lightcone. The *layers* of L are indexed by i, where the i-th layer contains $2(m-i)+1$ elements. Hence, the top layer contains $2m+1$ elements and the bottom one $2(m-n)+1$; in particular, the bottom layer is a single element if and only if $n = m$. There are $\sum_{i=0}^{n}(2(m-i)+1) = (n+1)(2m-n+1)$ elements in a lightcone in total.

▶ **Definition 8** (Neighborhoods and lightcones). *Let C be a CA and $n \in \mathbb{N}_+$. For $i \in [n]$ and $r \in \mathbb{N}_0$, the interval $[i-r, i+r] \cap [n]$ forms the* r-neighborhood *of i. For $t \in \mathbb{N}_0$, the* t-lightcone *of i is the lightcone of radius and height t centered at i in the 0-th row (i.e., the initial configuration) of the space-time diagram of C.*[4]

3 Fundamentals

As customary for randomized models of computation, one may consider both online and offline views of our model. Since it gives a more natural presentation, in the definition below we first assume an online perspective and then address the definitional issue mentioned in the introduction. In the last part, we switch to an offline view that we will use for the rest of the paper; this is more comfortable to work with since we can then refer to the cells' coin tosses explicitly.

▶ **Definition 9** (PACA). *Let Q be a finite set of states and $\Sigma \subseteq Q$ an alphabet. A* probabilistic ACA *(PACA) C is a CA with two local transition functions $\delta_0, \delta_1 \colon Q^3 \to Q$. At each step of C, each cell tosses a fair coin $c \in \{0,1\}$ and updates its state according to δ_c; that is, if the current configuration of C is $s \in Q^n$ and the cells obtain coin tosses $r = r_0 \cdots r_{n-1} \in \{0,1\}^n$ (where r_i is the coin toss of the i-th cell), then the next configuration of C is*

[3] Some sources distinguish between *future* and *past* lightcones. Here we shall need only past lightcones.
[4] If the lightcone's dimensions overstep the boundaries of the space-time diagram (i.e., i is too close to either of the borders of C (e.g., $i < t$)), then some cells in the t-lightcone will have undefined states. In this case, we set the undefined states to \$, which ensures consistency with δ.

$$\Delta_r(s) = \delta_{r_0}(\$, s_0, s_1)\,\delta_{r_1}(s_0, s_1, s_2)\cdots\delta_{r_{n-1}}(s_{n-2}, s_{n-1}, \$).$$

Seeing this process as a Markov chain M over Q^n, we recast the global transition function $\Delta = \Delta_{U_n}$ as a family of random variables $(\Delta(s))_{s\in Q^n}$ parameterized by the current configuration s of C, where $\Delta(s)$ is sampled by starting in state s and performing a single transition on M (having drawn the cells' coin tosses according to U_n). Similarly, for $t \in \mathbb{N}_0$, $\Delta^t(s)$ is sampled by starting in s and performing t transitions on M.

A computation *of C for an input $x \in \Sigma^n$ is a path in M starting at x. The computation is* accepting *if the path visits A^n at least once. In order to be able to quantify the probability of a PACA accepting an input, we additionally require for every PACA C that there is a function $T\colon \mathbb{N}_+ \to \mathbb{N}_0$ such that, for any input $x \in \Sigma^n$, every accepting computation for x visits A^n for the first time in strictly less than $T(n)$ steps; that is, if there is $t \in \mathbb{N}_0$ with $\Delta^t(x) \in A^n$, then $\Delta^{t_1}(x) \in A^n$ for some $t_1 < T(n)$. (Hence, every accepting computation for x has an initial segment with endpoint in A^n and whose length is strictly less than $T(n)$.) If this is the case for any such T, then we say C has* time complexity *(bounded by) T.*

With this condition in place, we may now equivalently replace the coin tosses of C with a matrix $R \in \{0,1\}^{T(n)\times n}$ of bits with rows $R_0, \ldots, R_{T(n)-1}$ and such that $R_j(i)$ corresponds to the coin toss of the i-th cell in step j. (If C accepts in step t, then the coin tosses in rows $t, \ldots, T(n)-1$ are ignored.) We refer to R as a random input *to C.*[5] *Blurring the distinction between the two perspectives (i.e., online and offline randomness), we write $C(x, R) = 1$ if C accepts x when its coin tosses are set according to R, or $C(x, R) = 0$ otherwise.*

Definition 9 states the acceptance condition for a *single* computation (i.e., one fixed choice of a random input); however, we must still define acceptance based on *all* computations (i.e., for random inputs picked according to a uniform distribution). The two most natural candidates are the analogues of the well-studied classes RP and BPP, which we define next.

▶ **Definition 10** (*p*-error PACA). *Let $L \subseteq \Sigma^*$ and $p \in [0, 1)$. A* one-sided p-error PACA for L *is a PACA C with time complexity $T = T(n)$ such that, for every $x \in \Sigma^n$,*

$$x \in L \iff \Pr[C(x, U_{T\times n}) = 1] \geq 1 - p \quad \text{and} \quad x \notin L \iff \Pr[C(x, U_{T\times n}) = 1] = 0.$$

If $p = 1/2$, then we simply say C is a one-sided error PACA. *Similarly, for $p < 1/2$, a* two-sided p-error PACA for L *is a PACA C with time complexity $T = T(n)$ for which*

$$x \in L \iff \Pr[C(x, U_{T\times n}) = 1] \geq 1 - p \quad \text{and} \quad x \notin L \iff \Pr[C(x, U_{T\times n}) = 1] \leq p$$

hold for every $x \in \Sigma^*$. If $p = 1/3$, then we simply say C is a two-sided error PACA. *In both cases, we write $L(C) = L$ and say C* accepts *L.*

Note that, to each 0-error PACA C, one can obtain an equivalent DACA C' with the same time complexity by setting the local transition function to δ_0. In the rest of the paper, if it is not specified which of the two variants above (i.e., one- or two-sided error) is meant, then we mean both variants collectively.

[5] The number of rows of R is dependent on the choice of T. This is not an issue here since any superficial rows are ignored by C; that is, without restriction we may take T to be such that every value $T(n)$ is minimal and set the number of rows of R to $T(n)$. The motivation for letting R be larger is that, when simulating a PACA, it may be the case that it is more convenient (or even possible) to compute only an upper bound $T'(n) \geq T(n)$ instead of the actual minimal value $T(n)$.

From the perspective of complexity theory, it is interesting to compare the PACA model with probabilistic circuits. It is known that every $T(n)$-time DACA can be simulated by an L-uniform AC circuit (i.e., a Boolean circuit with gates of unbounded fan-in) having $\mathsf{poly}(n)$ size and $O(\max\{1, T(n)/\log n\})$ depth [14]. Using the same approach as in [14], we note the same holds for PACAs if we use probabilistic AC circuits instead. The proof in [14] bases on descriptive complexity theory, the central observation being that the state of a cell i after $\log n$ steps given its $(\log n)$-neighborhood is a predicate that is computable in logarithmic space. Hence, for the PACA case we need only factor in the auxiliary random input into this predicate.

A key property that PACAs have but probabilistic circuits do not, however, is *distance* between computational units. (Indeed, in circuits, there is no such thing as the "length" of a wire.) One consequence of this is the following simple fact.

▶ **Lemma 11** (Independence of local events). *Let C be a one- or two-sided error PACA, let $x \in \Sigma^n$ be an input to C, and let $T \in \mathbb{N}_+$. In addition, let $i, j \in [n]$ be such that $|i - j| > 2(T-1)$ and E_i (resp., E_j) be an event described exclusively by the states of the i-th (resp., j-th) cell of C in the time steps $0, \dots, T-1$ (e.g., the i-th cell accepts in some step t where $t < T$). Then E_i and E_j are independent.*

Proof. For any random input R, the states of $k \in \{i, j\}$ in the time steps between 0 and $T-1$ is uniquely determined by the values of $R(t, k-T+t+1), \dots, R(t, k+T-t-1)$ for $t \in [T]$. Without loss of generality, suppose $i \leq j$. Since $i + T - 1 < j - T + 1$, E_i and E_j are conditioned on disjoint sets of values of R, thus implying independence. ◀

Note the proof still holds in case $T = 1$, in which case the events E_i and E_j occur with probability either 0 or 1, thus also (trivially) implying independence.

3.1 Robustness of the Definition

We now prove that the definition of PACA is robust with respect to the choice of $p = 1/2$ (resp., $p = 1/3$) for the error of one-sided (resp., two-sided) error PACA.

For one-sided error, we can reduce the error p to any desired constant value p'.

▶ **Proposition 12.** *Let $p, p' \in (0, 1)$ be constant and $p' < p$. For every one-sided p-error PACA C, there is a one-sided p'-error PACA C' such that $L(C) = L(C')$. Furthermore, if C has time complexity $T(n)$, then C' has time complexity $O(T(n))$.*

It follows that the definition of PACA is robust under the choice of p (as long as it is constant) and regardless of the time complexity (up to constant multiplicative factors).

The proof is essentially a generalization of the idea used in [14] to show that the sublinear-time DACA classes are closed under union. Namely, C' simulates several copies $C_0, \dots, C_{m-1}$ of C in parallel and accepting if and only if at least one C_i accepts. This idea is particularly elegant because m can be chosen to be constant and we update the C_i in a round-robin fashion (i.e., first C_0, then C_1, C_2, etc., and finally C_0 again after C_{m-1}). The alternative is to simulate each C_i for $T(n)$ steps at a time, which is not possible in general since we would have to compute $T(n)$ first. The construction we give avoids this issue entirely.

Proof. We construct a PACA C' with the desired properties. Let $m = \lceil \log(1/p' - 1/p) \rceil$. Furthermore, let Q be the state set of C and Σ its input alphabet. We set the state set of C' to $Q^m \times [m] \cup \Sigma$. Given an input x, every cell of C' initially changes its state from $x(i)$ to $(x(i), \dots, x(i), 0)$. The cells of C' simulate m copies of C as follows: If the last component of

a cell contains the value j, then its j-th component[6] q_0 is updated to $\delta(q_{-1}, q_0, q_1)$, where q_{-1} and q_1 are the j-th components of the left and right neighbors, respectively (or \$ in case of a border cell); at the same time, the last component of the cell is set to $j+1$ if $j < m$ or 0 in case $j = m$. A cell of C' is accepting if and only if its last component is equal to j and its j-th component is an accepting state of C.

Denote the i-th simulated copy of C by C_i. Clearly, C' accepts in step $mt + i + 1$ for $i \in [m]$ if and only if C_i accepts in step t, so we immediately have that C' has $O(T(n))$ time complexity. For the same reason and since C' never accepts in step 0, C' does not accept any input $x \notin L(C)$. As for $x \in L(C)$, note the m copies of C are all simulated using independent coin tosses, thus implying

$$\Pr[C' \text{ does not accept } x] = \Pr[\forall i \in [m] : C_i \text{ does not accept } x] < p^m \leq p'.$$

Hence, C' accepts x with probability at least $1 - p'$, as desired. ◀

For two-sided error, we show the same holds for every choice of p for *constant-time* PACA. We remark the construction is considerably more complex than in the one-sided error case.

▶ **Proposition 13.** *Let $p, p' \in (0, 1/2)$ be constant and $p' < p$. For every two-sided p-error PACA C with constant time complexity $T = O(1)$, there is a two-sided p'-error PACA C' with time complexity $O(T) = O(1)$ and such that $L(C) = L(C')$.*

It remains open whether a similar result holds for general (i.e., non-constant-time) two-sided error PACA. Generalizing our proof of Proposition 13 would require at the very least a construction for intersecting non-constant-time PACA languages. Note that closure under intersection is open in the deterministic setting (i.e., of DACA) as well [14].

3.2 One- vs. Two-Sided Error

The results of Section 3.1 are also useful in obtaining the following:

▶ **Theorem 3.** *The following hold:*

1. *If C is a one-sided error PACA with time complexity T, then there is an equivalent two-sided error PACA C' with time complexity $O(T)$.*
2. *There is a language L recognizable by constant-time two-sided error PACA but not by any $o(\sqrt{n})$-time one-sided error PACA.*

Proof. The first item follows from Proposition 12: Transform C into a one-sided error PACA C' with error at most $1/3$ and then notice that C' also qualifies as a two-sided error PACA (as it simply never errs on "no" instances). For the second item, consider the language

$$L = \{x \in \{0,1\}^+ \mid |x|_1 \leq 1\}.$$

We obtain a constant-time two-sided error PACA for L as follows: If a cell receives a 0 as input, then it immediately accepts; otherwise, it collects two random bits r_0 and r_1 in the first two steps and then, seeing $r_0 r_1$ as the binary representation of an integer $1 \leq t \leq 4$, it accepts (only) in the subsequent t-th step. Hence, if the input x is such that $|x|_1 \leq 1$, the PACA always accepts; conversely, if $|x|_1 \geq 2$, then the PACA only accepts if all 1 cells pick the same value for t, which occurs with probability at most $1/4$.

[6] In the same manner as we do for the indices of a word, we number the components starting with zero.

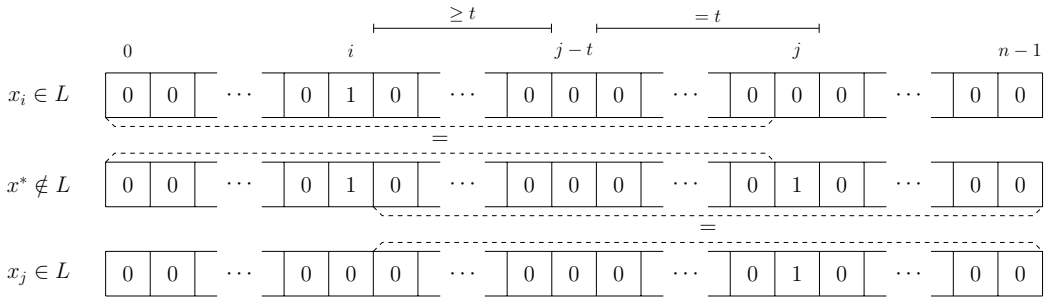


Figure 2 Constructing $x^* \notin L$ from $x_i, x_j \in L$. The numbers above the cells indicate their respective indices. Since every t-neighborhood of x^* appears in either x_i or x_j and both x_i and x_j are accepted in (exactly) t steps, it follows that C accepts x^* in t steps.

It remains to show $L(C) \neq L$ for any T-time one-sided error PACA C where $T = o(\sqrt{n})$. Let n be large enough so that $T = T(n) \leq \sqrt{n}/2$. Observe that $L \cap \{0,1\}^n = \{0^n, x_1, \dots, x_n\}$ where $x_i = 0^{i-1}10^{n-i}$. Let us now assume that $x_i \in L(C)$ holds for every i. Since C accepts with probability at least $1/2$, by the pigeonhole principle there is R such that $C(x_i, R) = 1$ for at least a $1/2$ fraction of the x_i. In addition, by averaging there is a step $t < T$ such that at least a $1/2T$ fraction of the x_i is accepted by C in step t. Since there are $n/2T \geq 2T \geq 2t+2$ such x_i, we can find $i, j \in [n]$ with $j \geq i + 2t + 1$ and $x_i, x_j \in L(C)$. Consider now the input

$$x^* = 0^{i-1}10^{j-i-1}10^{n-j},$$

which is not in L. We argue $C(x^*, R) = 1$, thus implying $L(C) \neq L$ and completing the proof.

We can see this by comparing the local views of the "bad" word x^* with the "good" ones x_i and x_j (see Figure 2): Let $k \in [n]$ be any cell of C. If $k < j - t$, then the t-neighborhood of k on input x^* is identical to that when the input is x_i, so k must be accepting in step t. Similarly, if $k \geq j - t$, then the t-neighborhood of k in x^* is the same as in x_j, so k is accepting as well. It follows all cells of C are accepting in step t for inputs x^* and R. ◀

4 The Constant-Time Case

In this section we now focus on constant-time PACA. Our goal will be to characterize the constant-time classes of both one- and two-sided error PACA (i.e., Theorems 5 and 6). First, we introduce the concept of *critical cells*, which is central to our analysis.

▶ **Definition 14** (Critical cell). *Let C be a one- or two-sided error PACA, and let $x \in L(C) \cap \Sigma^n$. We say a cell $i \in [n]$ is* critical *for x in step $t \in \mathbb{N}_0$ if there are random inputs $R, R' \in \{0,1\}^{t \times n}$ such that i is accepting in step t of $C(x, R)$ but not in step t of $C(x, R')$.*

In other words, if E is the event of i being accepting in step t of C on input x, then $0 < \Pr[E] < 1$ (where the probability is taken over the coin tosses of C). We should stress that whether a cell is critical or not may be *highly dependent* on x and t; for instance, there may be inputs $x_1 \neq x_2$ where the cell i is critical for x_1 but not for x_2.

As it turns out, the number of critical cells of a constant-time PACA is also constant.

▶ **Lemma 15.** *Let C be a T-time (one- or two-sided error) PACA for $T \in \mathbb{N}_+$, and let $x \in L(C) \cap \Sigma^n$. In addition, let $t \in [T]$ be a time step in which x is accepted by C with non-zero probability. Then there are $2^{O(T^2)}$ cells that are critical for x in step t. It follows there are $T \cdot 2^{O(T^2)} = 2^{O(T^2)}$ critical cells for x in total (i.e., all such time steps comprised).*

Proof. For $i \in [n]$, let E_i denote the event in which the i-th cell accepts in step t. Assume towards a contradiction that there is $x \in L(C)$ in which strictly more than $2T \cdot (1+\log T) \cdot 2^{T^2} = 2^{O(T^2)}$ cells are critical for x (in step t). By inspection, this implies there is a set $K \subseteq [n]$ of $|K| \geq (1 + \log T) \cdot 2^{T^2}$ cells such that every $k \in K$ is critical for x and, for every distinct $i, j \in K$, $|i - j| \geq 2T$. (For instance, in the extreme case where every $0 \leq i < 2T \cdot 2^{T^2}$ is critical, pick $K = \{2jT \mid j \in [2^{T^2}]\}$.) Since there are at most T^2 coin tosses that determine whether a cell $i \in K$ accepts or not (i.e., those in the T-lightcone of i), $\Pr[E_i] < 1$ if and only if $\Pr[E_i] \leq 1 - 2^{-T^2}$. By Lemma 11, the events E_i are all independent, implying

$$\Pr[C \text{ accepts } x \text{ in step } t] \leq \prod_{i \in K} \Pr[E_i] \leq \left(1 - \frac{1}{2^{T^2}}\right)^{|K|} < \frac{1}{e^{1+\log T}} < \frac{1}{2T}.$$

Since t was arbitrary, by a union bound it follows that the probability that C accepts x in step t is strictly less than $1/2$. This contradicts $x \in L(C)$ both when C is a one- and a two-sided error PACA. ◀

4.1 Characterization of One-Sided Error PACA

▶ **Theorem 5.** *For any constant-time one-sided error PACA C, there is a constant-time DACA C' such that $L(C) = L(C')$.*

Lemma 15 implies that, given any $x \in L(C)$, the decision of C accepting can be traced back to a set K of critical cells where $|K|$ is constant. To illustrate the idea, suppose that, if C accepts, then it always does so in a fixed time step $t < T$; in addition, assume the cells in K are all far apart (e.g., more than $2(T-1)$ cells away from each other as in Lemma 11).

Let us *locally* inspect the space-time diagram of C for x, that is, by looking at the t-lightcone of each cell i. Then we notice that, if $i \in K$, there is a choice of random bits in the t-lightcone of i that causes i to accept; conversely, if $i \notin K$, then *any* setting of random bits results in i accepting. Consider how this changes when $x \notin L(C)$ while assuming K remains unchanged. Every cell $i \notin K$ still behave the same; that is, it accepts regardless of the random input it sees. As for the cells in K, however, since they are all far apart, it cannot be the case that we still find random bits for every $i \in K$ that cause i to accept; otherwise C would accept x with non-zero probability, contradicting the definition of one-sided error PACA. Hence, there must be at least some $i \in K$ that *never* accepts. In summary, (under these assumptions) we can locally distinguish $x \in L(C)$ from $x \notin L(C)$ by looking at the cells of K and checking whether, for every $i \in K$, there is at least one setting of the random bits in the t-lightcone of i that causes it to accept.

To obtain the proof, we must generalize this idea to handle the case where the cells in K are not necessarily far from each other – which in particular means we can no longer assume that the events of them accepting are independent – as well as of K varying with the input. Since Lemma 15 only gives an upper bound for critical cells when the input is in $L(C)$, we must also account for the case where the input is not in $L(C)$ and $|K|$ exceeds said bound.

Proof. Let $T \in \mathbb{N}_+$ be the time complexity of C, and let M be the upper bound on the number of critical cells (for inputs in $L(C)$) from Lemma 15. Without restriction, we may assume $T > 1$. We first give the construction for C' and then prove its correctness.

Construction. Given an input $x \in \Sigma^n$, the automaton C' operates in two phases. In the first one, the cells communicate so that, in the end, each cell is aware of the inputs in its r-neighborhood, where $r = (2M-1)(T-1)$. (Note this is possible because r is constant.) The

second phase proceeds in T steps, with the cells assuming accepting states or not depending on a condition we shall describe next. After the second phase is over (and C' has not yet accepted), all cells unconditionally enter a non-accepting state and maintain it indefinitely. Hence, C' only ever accepts during the second phase.

We now describe when a cell $i \in [n]$ is accepting in the t-th step of the second phase, where $t \in [T]$. Let K be the set of critical cells of x in step t. The decision process is as follows:

1. First the cell checks whether there are strictly more than M cells in its r-neighborhood N that are critical in step t of C. If i cannot determine this for any cell $j \in N$ (since it did not receive the entire t-neighborhood of j during the first phase), then j is simply ignored. If this is the case, then i assumes a non-accepting state (in the t-th step of the second phase).
2. The cell then determines whether it is critical itself in step t of C. If this is *not* the case, then it becomes accepting if and only if it is accepting in step t of C (regardless of the random input).
3. Otherwise i is critical in step t. Let $B_i \subseteq [n]$ be the subset of cells that results from the following sequence of operations:
   a. Initialize B_i to $\{i\}$.
   b. For every cell $j \in B_i$, add to B_i every $k \in K$ such that $|j - k| \leq 2(T-1)$.
   c. Repeat step 2 until a fixpoint is reached.

   (This necessarily terminates since there are at most M critical cells in N and we checked the upper bound of M previously.) By choice of r, we have that $|i - j| \leq 2(M-1)(T-1)$ for every $j \in B_i$. In particular, we have that the t-neighborhood of every $j \in B_i$ is completely contained in N, which means cell i is capable of determining B_i.[7] The cell i then accepts if and only if there is a setting of random bits in the lightcone L_i of radius r and height t centered at i that causes every cell in B_i to accept in step t of C.

This process repeats itself in every step t of the second phase. Note that it can be performed by i instantaneously (i.e., without requiring any additional time steps of C') since it can be hardcoded into the local transition function δ.

Correctness. It is evident that C' is a constant-time PACA, so all that remains is to verify its correctness. To that end, fix an input x and consider the two cases:

$x \in L(C)$. Then there is a random input R such that C accepts x in step $t \in [T]$. This means that, for every critical cell $i \in K$, if we set the random bits in L_i according to R, then every cell in B_i accepts in step t of C. Likewise, every cell $i \notin K$ is accepting in step t of C by definition. In both cases we have that i also accepts in the t-th step of the second phase of C', thus implying $x \in L(C')$.

$x \notin L(C)$. Then, for every random input R and every step $t \in [T]$, there is at least one cell $i \in [n]$ that is not accepting in the t-th step of $C(x, R)$. If i is not critical, then i is also not accepting in the t-th step of the second phase of C' (regardless of the random input), and thus C' also does not accept x. Hence, assume that every such i (i.e., every i such that there is a random input R for which i is not accepting in the t-th step of $C(x, R)$) is a critical cell.

[7] Note it is not necessary for i to be aware of the actual numbers of the cells in B_i; it suffices for it to compute their positions relative to itself. For example, if $i = 5$ and $B_i = \{4, 5\}$, then it suffices for i to regard $j = 4$ as cell -1 (relative to itself). Hence, by "determining B_i" here we mean that i computes only these relative positions (and not the absolute ones, which would be impossible to achieve in only constant time).

Let $J \subseteq [n]$ denote the set of all such cells and, for $i \in J$, let $D_i \subseteq J$ be the subset that contains every $j \in J$ such that the events of i and j accepting in step t are *not* independent (conditioned on the random input to C). In addition, let A_i denote the event in which every cell of D_i is accepting in step t of $C(x, U_{T\times n})$. We show the following:

▷ Claim. There is an $i \in J$ such that $\Pr[A_i] = 0$; that is, for every R, there is at least one cell in D_i that is not accepting in step t of $C(x, R)$.

This will complete the proof since then i is also not accepting in the t-th step of the second phase of C' (since any cell in D_i is necessarily at most $2(M-1)(T-1)$ cells away from i), thus implying $x \notin L(C')$.
To see the claim is true, suppose towards a contradiction that, for every $i \in J$, we have $\Pr[A_i] > 0$. If there are $i, j \in J$ such that $j \notin D_i$ (and similarly $i \notin D_j$), then by definition

$$\Pr[C(x, U_{T\times n}) = 1] \geq \Pr[A_i \wedge A_j] = \Pr[A_i]\Pr[A_j] > 0,$$

contradicting $x \notin L(C)$. Thus, there must be $i \in J$ such that $J = D_i$; however, this then implies

$$\Pr[C(x, U_{T\times n}) = 1] = \Pr[A_i] > 0.$$

As this is also a contradiction, the claim (and hence the theorem) follows. ◀

4.2 Characterization of Two-Sided Error PACA

We introduce some notation. For $\ell \in \mathbb{N}_0$ and a word $w \in \Sigma^*$, $p_\ell(w)$ is the prefix of w of length ℓ if $|w| \geq \ell$, or w otherwise; similarly, $s_\ell(w)$ is the suffix of length ℓ if $|w| \geq \ell$, or w otherwise. The set of infixes of w of length (exactly) ℓ is denoted by $I_\ell(w)$.

The subregular language classes from the next definition are due to McNaughton and Papert [12] and Beauquier and Pin [3].

▶ **Definition 16** (SLT, LT, LTT). *A language $L \subseteq \Sigma^*$ is* strictly locally testable *if there is $\ell \in \mathbb{N}_+$ and sets $\pi, \sigma \subseteq \Sigma^{\leq \ell}$ and $\mu \subseteq \Sigma^\ell$ such that, for every $w \in \Sigma^*$, $w \in L$ if and only if $p_{\ell-1}(w) \in \pi$, $I_\ell(w) \subseteq \mu$, and $s_{\ell-1}(w) \in \sigma$. The class of all such languages is denoted by* SLT.

A language $L \subseteq \Sigma^*$ is locally testable *if there is $\ell \in \mathbb{N}_+$ such that, for every $w_1, w_2 \in \Sigma^*$ with $p_{\ell-1}(w_1) = p_{\ell-1}(w_2)$, $I_\ell(w_1) = I_\ell(w_2)$, and $s_{\ell-1}(w_1) = s_{\ell-1}(w_2)$, we have that $w_1 \in L$ if and only if $w_2 \in L$. The class of locally testable languages is denoted by* LT.

A language $L \subseteq \Sigma^*$ is locally threshold testable *if there are $\theta, \ell \in \mathbb{N}_+$ such that, for any two words $w_1, w_2 \in \Sigma^*$ for which the following conditions hold, $w_1 \in L$ if and only if $w_2 \in L$:*
1. *$p_{\ell-1}(w_1) = p_{\ell-1}(w_2)$ and $s_{\ell-1}(w_1) = s_{\ell-1}(w_2)$.*
2. *For every $m \in \Sigma^\ell$, if $|w_i|_m < \theta$ for any $i \in \{1, 2\}$, then $|w_1|_m = |w_2|_m$.*

The class of locally threshold testable languages is denoted by LTT.

The class LT equals the closure of SLT under Boolean operations (i.e., union, intersection, and complement) and the inclusion SLT $\subsetneq$ LT is proper. As for LTT, it is well-known that it contains every language $\mathsf{Th}(m, \theta) = \{w \in \Sigma^* \mid |w|_m \leq \theta\}$ where $m \in \Sigma^*$ and $\theta \in \mathbb{N}_0$. Also, we have that LT $\subsetneq$ LTT and that LTT is closed under Boolean operations. We write $\mathsf{SLT}_\cup$ for the closure of SLT.

▶ **Definition 17** (LLT). *For $\ell \in \mathbb{N}_0$, $\theta \in \mathbb{R}_0^+$, $\pi, \sigma \subseteq \Sigma^{\leq \ell-1}$, and $\alpha \colon \Sigma^\ell \to \mathbb{R}_0^+$, $\mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta)$ denotes the language of all $w \in \Sigma^+$ that satisfy $p_{\ell-1}(w) \in \pi$, $s_{\ell-1}(w) \in \sigma$, and*

$$\sum_{m \in \Sigma^\ell} \alpha(m) \cdot |w|_m \leq \theta.$$

A language $L \subseteq \Sigma^+$ is said to be locally linearly testable *if there are ℓ, π, σ, α, and θ as above such that $L = \mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta)$. We denote the class of all such languages by* LLT.

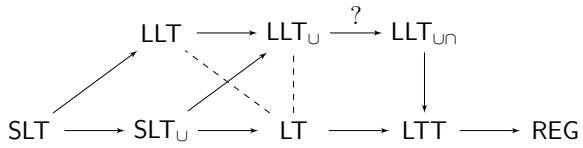


Figure 3 Placement of LLT in the subregular hierarchy. Arrows indicate inclusion relations; all are known to be strict except for the marked one. Dashed lines connect incomparable classes.

We write $\mathsf{LLT}_\cup$ for the closure of LLT under union and $\mathsf{LLT}_{\cup\cap}$ for its closure under both union and intersection.

▶ **Proposition 18.** *The following hold (see Figure 3):*
1. $\mathsf{SLT} \subsetneq \mathsf{LLT} \subsetneq \mathsf{LLT}_\cup \subseteq \mathsf{LLT}_{\cup\cap} \subsetneq \mathsf{LTT}$ *and* $\mathsf{SLT}_\cup \subsetneq \mathsf{LLT}_\cup$.
2. LLT *and* LT *as well as* $\mathsf{LLT}_\cup$ *and* LT *are incomparable.*
3. LTT *equals the Boolean closure of* LLT.

With this terminology in place, we now turn to:

▶ **Theorem 6.** *The class of languages that can be accepted by a constant-time two-sided error PACA contains* $\mathsf{LLT}_{\cup\cap}$ *and is strictly contained in* LTT.

The theorem is proven by showing the two inclusions. In the following, we give a brief overview of the ideas involved; for the full proof, see Appendix B.

First inclusion. The first step is showing that we can "tweak" the components of the LLT condition so that it is more amenable to being tested by a PACA. In particular, we prove we can assume the $\alpha(m)$ weights are such that $2^{-\alpha(m)}$ can be represented using a constant number of bits. Having done so, the construction is more or less straightforward: We collect the subwords of length ℓ in every cell and then accept with the "correct" probabilities; that is, if a cell sees a subword m, then it accepts with probability $2^{-\alpha(m)}$. To lift the inclusion from LLT to $\mathsf{LLT}_{\cup\cap}$, we show the following result:

▶ **Proposition 19.** *Let C_1 and C_2 be constant-time two-sided error PACA. Then there are constant-time two-sided error PACA $C_\cup$ and $C_\cap$ such that $L(C_\cup) = L(C_1) \cup L(C_2)$ and $C_\cap = L(C_1) \cap L(C_2)$.*

Second inclusion. The second inclusion is considerably more complex. Let C be a PACA with time complexity at most T. The proof again bases on the class LLT and uses the fact that LTT equals the Boolean closure over LLT (which we show as a separate result):
1. As a warm-up, we consider the case where the cells of C accept all independently from one another. In addition, we assume that, if C accepts, then it does so in a fixed time step $t < T$. The argument is then relatively straightforward since we need only consider subwords of length $\ell = 2T + 1$ and set their LLT weight according to the acceptance probability that the cell in the middle of the subword would have in C.
2. Next we relax the requirement on independence between the cells (but still assume a fixed time step for acceptance). The situation then requires quite a bit of care since the LLT condition does not account for subword overlaps at all. For instance, there may be cells c_1 and c_2 that are further than T cells apart and that both accept with non-zero probability but where c_i accepts if and only if c_{3-i} does not (see Appendix A for a concrete example). We solve this issue by blowing up ℓ so that a subword covers not only a single cell's

neighborhood but that of an *entire group* of cells whose behavior may be correlated with one other. Here we once more resort to Lemmas 11 and 15 to upper-bound the size of this neighborhood by a constant.

3. Finally, we generalize what we have shown so that it also holds in the case where C may accept in any step $t < T$. This is the only part in the proof where closure under complement is required. The argument bases on generalizing the ideas of the previous step to the case where the automaton may accept in multiple time steps and then applying the inclusion-exclusion principle.

5 The General Sublinear-Time Case

Recall we say a DACA C is *equivalent* to a PACA C' if $L(C) = L(C')$. In this section, we recall and briefly discuss:

▶ **Theorem 4.** *Let $d \geq 1$. The following hold:*

- *If there is $\varepsilon > 0$ such that every n^ε-time (one- or two-sided error) PACA can be converted into an equivalent n^d-time deterministic CA, then $\mathsf{P} = \mathsf{RP}$.*
- *If every $\mathsf{polylog}(n)$-time PACA can be converted into an equivalent n^d-time deterministic CA, then, for every $\varepsilon > 0$, $\mathsf{RP} \subseteq \mathsf{TIME}[2^{n^\varepsilon}]$.*
- *If there is $b > 2$ so that any $(\log n)^b$-time PACA can be converted into an equivalent n^d-time deterministic CA, then, for every $a \geq 1$ and $c > a/(b-1)$, $\mathsf{RTIME}[n^a] \subseteq \mathsf{TIME}[2^{O(n^c)}]$.*

To obtain Theorem 4, we prove the following more general result:

▶ **Proposition 20.** *There is a constant $c > 0$ such that the following holds: Let monotone functions $T, T', h, p \colon \mathbb{N}_+ \to \mathbb{N}_+$ be given with $h(n) \leq 2^n$, $p(n) = \mathsf{poly}(n)$, and $p(n) \geq n$ and such that, for every n and $p'(n) = \Theta(p(n) \log h(n))$, $T(6h(n)p(n)) \geq cp'(n)$. In addition, for n given in unary, let the binary representation of $h(n)$ and $p'(n)$ be computable in $O(p'(n))$ time by a Turing machine and, for N given in unary, let $T'(N)$ be computable in $O(T'(N))$ time by a Turing machine. Furthermore, suppose that, for every T-time one-sided error PACA C, there is a T'-time DACA C' such that $L(C) = L(C')$. Then*

$$\mathsf{RTIME}[p(n)] \subseteq \mathsf{TIME}[h(n) \cdot T'(h(n) \cdot \mathsf{poly}(n)) \cdot \mathsf{poly}(n)].$$

The first item of Theorem 4 is obtained by setting (say) $T(n) = n^\varepsilon$, $T'(n) = n^d$, and $h(n) = p(n)^{2(1/\varepsilon - 1)}$ (assuming $\varepsilon < 1$). For the second one, letting $p(n) = n^a$ where $a > 0$ is arbitrary and (again) $T'(n) = n^d$, set $T(n) = (\log n)^{2+a/\varepsilon}$ and $h(n) = 2^{n^\varepsilon/(d+1)}$. Finally, for the last one, letting again $p(n) = n^a$ and $T'(n) = n^d$, set $T(n) = (\log n)^b$ and $h(n) = 2^{n^c}$.

At the core of the proof of Proposition 20 is a padding argument. Nevertheless, we cannot stress enough that the padding itself is highly nontrivial. In particular, it requires a clever implementation that ensures it can be verified *in parallel* and also *without initial knowledge of the input length*. To see why this is so, observe that, if we simply use a "standard" form of padding where we map $x \in \{0,1\}^+$ to $x' = x0^{p(|x|)}$ (where $p \colon \mathbb{N}_+ \to \mathbb{N}_0$ gives the desired padding length), then it is impossible for the automaton to distinguish between this and, say, $x'' = x0^{p(|x|)/2}$ in $o(p)$ time (assuming, e.g., $p(|x|) = \Omega(|x|)$). The reason for this is that, since cells are initially completely unaware of their position in the input, the cells with an all-zeroes neighborhood must behave exactly the same. More specifically, we can use an argument as in the proof of Theorem 3 to show that the automaton must behave the same on both x' and x'' (in the sense that it accepts the one if and only if it accepts the other) unless it "looks at the whole input" (i.e., unless it has $\Omega(p(|x|)$ time complexity).

The padding technique we use can be traced back to [8]. In a nutshell, we split the input into blocks of the same size that redundantly encode the input length in a locally verifiable way. More importantly, the blocks are numbered from left to right in ascending order, which also allows us to verify that we have the number of blocks that we need. This is crucial in order to ensure the input is "long enough" and the PACA achieves the time complexity that we desire (as a function of the input length).

Proof. Let $L \in \mathsf{RP}$ be decided by an RP machine R whose running time is upper-bounded by p. Without restriction, we assume $p(n) \geq n$. Using standard error reduction in RP, there is then an RP machine R' with running time $p'(n) = \Theta(p(n) \log h(n))$ (i.e., polynomial in n), space complexity at most $p(n)$, and which errs on $x \in L$ with probability strictly less than $1/2h(n)$. Based on L we define the language

$$L' = \{\mathrm{bin}_n(0)\#x_0\#0^{p(n)}\% \cdots \% \,\mathrm{bin}_n(h(n)-1)\#x_{h(n)-1}\#0^{p(n)} \mid n \in \mathbb{N}_+, x_i \in L \cap \Sigma^n\},$$

where $\mathrm{bin}_n(i)$ denotes the n-bit representation of $i < 2^n$. Note the length of an instance of L' is $N \leq 6h(n)p(n) = O(h(n)\mathsf{poly}(n))$.

We claim there is $c > 0$ such that L' can be accepted in at most $cp'(n) = O(p'(n))$ (and in particular less than $T(N)$) time by a one-sided error PACA C. The construction is relatively straightforward: We refer to each group of cells $\mathrm{bin}_n(i)\#x_i\#0^{p'(n)}$ separated by the % symbols as a *block* and the three binary strings in each block (separated by the # symbols) as its *components*. First each block $a_1\#a_2\#a_3$ checks that its components have correct sizes, that is, that $|a_1| = |a_2|$ and $|a_3| = p(|a_1|)$. Then the block communicates with its right neighbor $b_1\#b_2\#b_3$ (if it exists) and checks that $|a_i| = |b_i|$ for every i and that, if $a_1 = \mathrm{bin}_n(j)$, then $b_1 = \mathrm{bin}_n(j+1)$. In addition, the leftmost block checks that its first component is equal to $\mathrm{bin}_n(0)$; similarly, the rightmost block computes $h(n)$ (in $O(p'(n))$ time) and checks that its first component is equal to $\mathrm{bin}_n(h(n)-1)$. Following these initial checks, each block then simulates R' (using bits from its random input as needed) on the input given in its second component using its third component as the tape. If R' accepts, then all cells in the respective block turn accepting. In addition, the delimiter % is always accepting unless it is a border cell.

Clearly C accepts if and only if its input is correctly formatted and R' accepts every one of the x_i (conditioned on the coin tosses that are chosen for it by the respective cells of C). Using a union bound, the probability that C errs on an input $x \in L'$ is

$$\Pr[C(x, U_{T\times n}) = 0] \leq \sum_{i=0}^{h(n)-1} \Pr[R(x_i) = 0] < \sum_{i=0}^{h(n)-1} \frac{1}{2h(n)} = \frac{1}{2}.$$

In addition, the total running time of C is the time needed for the syntactic checks (requiring $O(p'(n))$ time), plus the time spent simulating R' (again, $O(p'(n))$ time using standard simulation techniques). Hence, we can implement C so that it runs in at most $cp'(n)$ time for some constant $c > 0$, as desired.

Now suppose there is a DACA C' equivalent to C as in the statement of the theorem. We shall show there is a deterministic (single-tape) Turing machine that decides L with the purported time complexity. Consider namely the machine S which, on an input $x \in \{0,1\}^n$ of L, produces the input

$$x' = \mathrm{bin}_n(0)\#x\#0^{p(n)}\% \cdots \% \,\mathrm{bin}_n(h(n)-1)\#x\#0^{p(n)}$$

of L' and then simulates C' on x' for $T'(N)$ steps, accepting if and only if C' does. Producing x' from x requires $O(N \cdot \mathsf{poly}(n))$ time since we need only copy $O(n)$ bits from each block separated by the % delimiters to the next (namely the string x and the number of the

previous block). Using the standard simulation of cellular automata by Turing machines, the subsequent simulation of C' requires $O(N \cdot T'(N))$ time. Checking whether C' accepts or not can be performed in parallel to the simulation and requires no additional time. Hence, the time complexity of S is

$$O(N \cdot \mathsf{poly}(n) + N \cdot T'(N)) = O(h(n) \cdot \mathsf{poly}(n) \cdot T'(h(n) \cdot \mathsf{poly}(n))).$$ ◀

6 Further Directions

LLT and two-sided error PACA. Besides giving a separation between one- and two-sided error, Theorem 6 considerably narrows down the position of the class of languages accepted by constant-time two-sided error PACA in the subregular hierarchy. Nevertheless, even though we now know the class is "sandwiched" in-between $\mathsf{LLT}_{\cup\cap}$ and LTT, we still do not have a precise characterization for it. It is challenging to tighten the inclusion from Theorem 6 because the strategy we follow relies on closure under complement, but (as we also prove) the class of two-sided error PACA is *not* closed under complement. It appears that clarifying the relation between said class and $\mathsf{LLT}_{\cup\cap}$ as well as $\mathsf{LLT}_{\cup\cap}$ itself and $\mathsf{LLT}_{\cup}$ or also LT may give a "hint" on how to proceed.

The general sublinear-time case. Theorem 4 indicates that even polylogarithmic-time PACA can recognize languages for which no deterministic polynomial-time algorithm is currently known. Although the proof of Proposition 20 does yield explicit examples of such languages, they are rather unsatisfactory since in order to accept them we do not need the full capabilities of the PACA model. (In particular, communication between blocks of cells is only required to check certain syntactic properties of the input; once this is done, the blocks operate independently from one another.) It would be very interesting to identify languages where the capabilities of the PACA model are put to more extensive use.

Pseudorandom generators. From the opposite direction, to investigate the limitations of the PACA model, one possibility would be to construct *pseudorandom generators* (PRGs) that fool sublinear-time PACAs. Informally, such a PRG is a function $G\colon \{0,1\}^{s(n)} \to \{0,1\}^{r(n)}$ with $s(n) \ll r(n)$ and having the property that a PACA (under given time constraints) is incapable of distinguishing $G(x)$ from uniform when the seed x is chosen uniformly at random. PRGs have found several applications in complexity theory (see, e.g., [21] for an introduction).

Theorem 4 suggests that an unconditional time-efficient derandomization of PACAs is beyond reach of current techniques, so perhaps *space-efficient* derandomization should be considered instead. Indeed, as a PACA can be simulated by a space-efficient machine (e.g., by adapting the algorithm from [13]), it is possible to recast PRGs that fool space-bounded machines (e.g., [7, 16]) as PRGs that fool PACAs. Nevertheless, we may expect to obtain even better constructions by exploiting the locality of PACAs (which space-bounded machines do not suffer from).

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

2 Pablo Arrighi, Nicolas Schabanel, and Guillaume Theyssier. Stochastic cellular automata: Correlations, decidability and simulations. *Fundam. Informaticae*, 126(2-3):121–156, 2013. doi:10.3233/FI-2013-875.

3 Danièle Beauquier and Jean-Eric Pin. Factors of words. In *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, pages 63–79, 1989. doi:10.1007/BFb0035752.

4 Marianne Delorme and Jacques Mazoyer, editors. *Cellular Automata*. Number 460 in Mathematics and Its Applications. Springer Netherlands, 1999. doi:10.1007/978-94-015-9153-9.

5 Pedro García and José Ruiz. Threshold locally testable languages in strict sense. In Carlos Martín-Vide and Victor Mitrana, editors, *Grammars and Automata for String Processing: From Mathematics and Computer Science to Biology, and Back: Essays in Honour of Gheorghe Paun*, volume 9 of *Topics in Computer Mathematics*, pages 243–252. Taylor and Francis, 2003.

6 Oded Goldreich. *Computational Complexity: A Conceptional Perspective*. Cambridge University Press, 2008.

7 William M. Hoza and David Zuckerman. Simple optimal hitting sets for small-success RL. *SIAM J. Comput.*, 49(4):811–820, 2020. doi:10.1137/19M1268707.

8 Oscar H. Ibarra, Michael A. Palis, and Sam M. Kim. Fast parallel language recognition by cellular automata. *Theor. Comput. Sci.*, 41:231–246, 1985. doi:10.1016/0304-3975(85)90073-8.

9 Sam Kim and Robert McCloskey. A characterization of constant-time cellular automata computation. *Phys. D*, 45(1-3):404–419, 1990. doi:10.1016/0167-2789(90)90198-X.

10 Martin Kutrib. Cellular automata and language theory. In *Encyclopedia of Complexity and Systems Science*, pages 800–823. Springer, 2009. doi:10.1007/978-0-387-30440-3_54.

11 Jean Mairesse and Irène Marcovici. Around probabilistic cellular automata. *Theor. Comput. Sci.*, 559:42–72, 2014. doi:10.1016/j.tcs.2014.09.009.

12 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. The MIT Press, 1971.

13 Augusto Modanese. Lower bounds and hardness magnification for sublinear-time shrinking cellular automata. In Rahul Santhanam and Daniil Musatov, editors, *Computer Science – Theory and Applications – 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 – July 2, 2021, Proceedings*, volume 12730 of *Lecture Notes in Computer Science*, pages 296–320. Springer, 2021. doi:10.1007/978-3-030-79416-3_18.

14 Augusto Modanese. Sublinear-time language recognition and decision by one-dimensional cellular automata. *Int. J. Found. Comput. Sci.*, 32(6):713–731, 2021. doi:10.1142/S0129054121420053.

15 Augusto Modanese. Sublinear-time probabilistic cellular automata. *CoRR*, abs/2203.14614, 2022. arXiv:2203.14614.

16 Noam Nisan. Pseudorandom generators for space-bounded computation. *Comb.*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

17 José Ruiz, Salvador España Boquera, and Pedro García. Locally threshold testable languages in strict sense: Application to the inference problem. In *Grammatical Inference, 4th International Colloquium, ICGI-98, Ames, Iowa, USA, July 12-14, 1998, Proceedings*, pages 150–161, 1998. doi:10.1007/BFb0054072.

18 Rudolph Sommerhalder and S. Christian van Westrhenen. Parallel language recognition in constant time by cellular automata. *Acta Inf.*, 19:397–407, 1983. doi:10.1007/BF00290736.

19 Jukka Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013. doi:10.1145/2431211.2431223.

20 Véronique Terrier. Language recognition by cellular automata. In *Handbook of Natural Computing*, pages 123–158. Springer, 2012. doi:10.1007/978-3-540-92910-9_4.

21 Salil P. Vadhan. Pseudorandomness. *Found. Trends Theor. Comput. Sci.*, 7(1-3):1–336, 2012. doi:10.1561/0400000010.

22 Andrew Chi-Chih Yao. Circuits and local computation. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 186–196. ACM, 1989. doi:10.1145/73007.73025.

A An Example for PACA Being More Efficient than DACA

▶ **Example 21.** Let $\Sigma = \{0, 1, 2, 3\}$ and consider the language

$$L = \{0^k 1^l 2^m 3^n \mid k, l, m, n \in \mathbb{N}_0 \text{ and } ((l \geq 2 \text{ and } m \geq 3) \text{ or } (l \geq 3 \text{ and } m \geq 2))\}.$$

No DACA C accepts L in at most 3 steps. This can be shown using methods from [9, 14, 18].

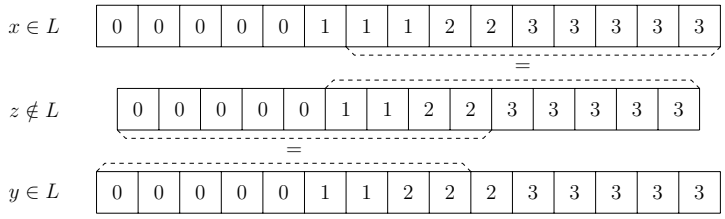


Figure 4 Comparing the words $x = 0^5 1^3 2^2 3^5 \in L$, $y = 0^5 1^2 2^3 3^5 \in L$, and $z = 0^5 1^2 2^2 3^5 \notin L$, we notice that every infix of length 5 of z appears in either x or y. This implies there is no DACA that accepts L with time complexity 3 or less.

Given a DACA C with time complexity 3, we can determine if C accepts a word $x \in \Sigma^*$ by looking only at the infixes of length 5 (and the prefix and suffix of length 4) of x. Consider the words $x = 0^5 1^3 2^2 3^5 \in L$, $y = 0^5 1^2 2^3 3^5 \in L$, and $z = 0^5 1^2 2^2 3^5 \notin L$ (Figure 4). Then every infix of length 5 (and the prefix and suffix of length 4) of z appears in x except for the infixes 00112 and 01122, which both appear in y. It follows that, if $x, y \in L(C)$, then C must also accept z, which proves there is no DACA for L with time complexity (at most) 3.

Nevertheless, there is a 3-time one-sided 7/8-error PACA C' for L. Checking that the input $x = 0^k 1^l 2^m 3^n$ is such that (x is of the form $0^* 1^* 2^* 3^*$ and) $l, m \geq 2$ can be done without need of randomness simply by looking at the infixes of length 5 of x: Every cell collects the infix m that corresponds to its position in the input and rejects if m is disallowed. (We refer to [9, 14, 18] for the general method.) This procedure is carried out in parallel to the one we describe next (and a cell turns accepting if and only if it both procedures dictate it to do so).

Now we use randomness to check that one of $m \geq 3$ and $l \geq 3$ holds. In time step 1, every cell exposes its coin toss of step 0 so that its neighbors can read it and use it to choose their state in step 2. Let c_σ denote the leftmost cell in which $\sigma \in \Sigma$ appears, and let l_σ and r_σ be the coin tosses of the left and right neighbors of c_σ, respectively. We have c_1 accept if and only if $r_1 = 1$, c_3 if and only if $l_3 = 1$, and c_2 if and only if $l_2 + r_2 < 2$. All other cells accept regardless of the coin tosses they see (as long as x satisfies the conditions we specified above).

For $i \in \{1, 2, 3\}$, let A_i denote the event of cell c_i accepting. The above results in the following behavior: If $l = m = 2$, we have $r_1 = l_2$ and $r_2 = l_3$ since the coin tosses belong to the same cells, in which case C' never accepts. If $l \geq 3$ and r_2 and l_3 belong to the same cell (i.e., $r_2 = l_3$), then r_1 and l_2 do not belong to the same cell and we have

$$\Pr[C(x, U_{T\times|x|}) = 1] = \Pr[A_1] \Pr[A_2 \wedge A_3] = \Pr[r_1 = 1] \Pr[l_2 = 0 \wedge r_2 = l_3 = 1] = \frac{1}{8}.$$

The case $m \geq 3$ and r_1 and l_2 belonging to the same cell is similar. Finally, if $l \geq 3$ and $m \geq 3$, the values r_1, l_2, r_2, and l_3 are all independent and we have

$$\Pr[C(x, U_{T\times|x|}) = 1] = \prod_{i=1}^{3} \Pr[A_i] = \Pr[r_1 = 1] \Pr[l_2 + r_2 < 2] \Pr[l_3 = 1] > \frac{1}{8}.$$

B Proof of Theorem 6

This appendix contains the proof of Theorem 6, which we recall for the reader's convenience.

▶ **Theorem 6.** *The class of languages that can be accepted by a constant-time two-sided error PACA contains* $\mathsf{LLT}_{\cup\cap}$ *and is strictly contained in* LTT.

For the proof we first need to make the linear condition of LLT a bit more manageable:

▶ **Lemma 22.** *For any* $L = \mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta)$, *there is* α' *such that* $L = \mathsf{LLin}_\ell(\pi, \sigma, \alpha', \theta')$ *and:*

1. *The threshold* θ' *is equal to* 1.
2. *There is a constant* $\varepsilon > 0$ *such that, for every* $w \in \Sigma^*$, *we have either* $f'(w) < 1 - \varepsilon$ *or* $f'(w) > 1 + \varepsilon$, *where* $f'(w) = \sum_{m \in \Sigma^\ell} \alpha'(m) \cdot |w|_m$.
3. *There is* $k \in \mathbb{N}_0$ *such that, for every* m, *there is* $n \in [2^k]$ *such that* $\alpha'(m) = k - \log(n+1)$.

The first two items ensure the sum $f'(w)$ is always "far away" from 1. In turn, the third item enables us to represent $2^{-\alpha'(m)}$ using at most k (and, in particular, $O(1)$ many) bits.

Proof. The case where $\alpha(m) = 0$ for every m is trivial, so suppose there is some m for which $\alpha(m) > 0$. Because $|w|_m \in \mathbb{N}_0$ for every w and every m, $f(w) = \sum_{m \in \Sigma^\ell} \alpha(m) \cdot |w|_m$ is such that, given any $b \in \mathbb{R}_0^+$, there are only finitely many values of $f(w) \leq b$ (i.e., the set $\{f(w) \mid f(w) \leq b\}$ is finite). Hence, by setting

$$r = \frac{1}{2}\left(\max_{f(w) \leq \theta} f(w) + \min_{f(w) > \theta} f(w)\right)$$

and $\alpha''(m) = \alpha(m)/r$, we have that $f''(w) = \sum_{m \in \Sigma^\ell} \alpha''(m) \cdot |w|_m$ satisfies the second condition from the claim (i.e., for every $w \in \Sigma^*$, either $f''(w) < 1 - \varepsilon$ or $f''(w) > 1 + \varepsilon$) for some adequate choice of $\varepsilon > 0$.

Now we show how to satisfy the last condition without violating the first two. Essentially, we will choose k to be sufficiently large and then "round up" each $\alpha''(m)$ to the nearest value of the form $k - \log(n+1)$. To that end, let a be the minimal $\alpha''(m)$ for which $\alpha''(m) > 0$. In addition, let $k \in \mathbb{N}_0$ be such that $\alpha''(m) < k/2$ for every m and that

$$\log(2^{k/2} + 1) - \log(2^{k/2}) = \log(2^{k/2} + 1) - \frac{k}{2} \leq \frac{a\varepsilon}{2|\Sigma|^\ell}.$$

(This is possible because $\log(n+1) - \log n$ tends to zero as $n \to \infty$ and the right-hand side is constant.) Then, for every m, we set $\alpha'(m) = k - \log(n+1)$ where $n \in [2^k]$ is maximal such that $\alpha'(m) \geq \alpha''(m)$. By construction, $f'(w) \geq f''(w)$, so we need only argue that there is $\varepsilon' > 0$ such that, for every w for which $f''(w) < 1 - \varepsilon$, we also have $f'(w) < 1 - \varepsilon'$. In particular every said w must be such that, for every m, $|w|_m \leq 1/a$ (otherwise we would have $f''(w) > 1$). Noting that $|\alpha'(m) - \alpha''(m)|$ is maximal when $\alpha'(m) = k/2$ and $\alpha''(m) = k - \log(2^{k/2} + 1) + \delta$ for very small $\delta > 0$, we observe that

$$f'(w) = \sum_{m \in \Sigma^\ell} \alpha'(m) \cdot |w|_m < f''(w) + \frac{|\Sigma|^\ell}{a}\left(\log(2^{k/2} + 1) - \frac{k}{2}\right) < 1 - \varepsilon + \frac{\varepsilon}{2} = 1 - \frac{\varepsilon}{2},$$

that is, $f'(w) < 1 - \varepsilon'$ for $\varepsilon' = \varepsilon/2$, as desired. ◀

Proof of Theorem 6. We prove the two inclusions from the theorem's statement. The first one we address is that of $\mathsf{LLT}_{\cup\cap}$ in the class of constant-time two-sided error PACA.

First inclusion. Given $L = \mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta)$, we construct a constant-time two-sided error PACA C with $L(C) = L$. This suffices since by the closure properties shown in Proposition 19. We apply Lemma 22 and assume $\theta = 1$ and that there are k and ε as in the statement of Lemma 22. For simplicity, we also assume $\ell = 2k + 1$.

The automaton C operates in k steps as follows: Every cell sends its input symbol in both directions as a signal and, at the same time, aggregates the symbols it sees, thus allowing it to determine the initial configuration $m \in \Sigma^\ell$ of its k-neighborhood. Meanwhile, every cell also collects k random bits $r \in \{0,1\}^k$. The decision to accept is then simultaneously made in the k-th step, where a cell with k-neighborhood m accepts with probability $2^{-\alpha(m)}$ (independently of other cells). (This can be realized, for instance, by seeing r as the representation of a k-bit integer in $[2^k]$ and accepting if and only if $r \leq n$, where n is such that $2^{-\alpha(m)} = (n+1)/2^k$.) In the case of the first (resp., last) cell of C, it also checks that the prefix (resp., suffix) of the input is in π (resp., σ), rejecting unconditionally if this is not the case.

Hence, for an input word $w \in \Sigma^+$, the probability that C accepts is

$$\prod_{m \in \Sigma^\ell} \left(\frac{1}{2^{\alpha(m)}}\right)^{|w|_m} = 2^{-f(w)},$$

where $f(w) = \sum_{m \in \Sigma^\ell} \alpha(m) \cdot |w|_m$. Thus, if $w \in L$, then C accepts with probability $2^{-f(w)} > (1/2)^{1-\varepsilon}$; conversely, if $w \notin L$, the probability that C accepts is $2^{-f(w)} < (1/2)^{1+\varepsilon}$. Since ε is constant, we may apply Proposition 13 and reduce the error to $1/3$.

Second inclusion. The proof of the second inclusion is more involved. Let C be a T-time two-sided error PACA for some $T \in \mathbb{N}_+$. We shall obtain $L(C) \in \mathsf{LTT}$ in three steps:

1. The first step is a warm-up where the cells of C accept all independently from one another and that, if C accepts, then it does so in a fixed time step $t < T$.
2. Next we relax the requirement on independence between the cells by considering groups of cells (of maximal size) that may be correlated with one another regarding their acceptance.
3. Finally, we generalize what we have shown so that it also holds in the case where C may accept in any step $t < T$. This is the only part in the proof where closure under complement is required. (Here we use item 3 of Proposition 18.)

Step 1. Suppose that C only accepts in a fixed time step $t < T$ and that the events of any two cells accepting are independent from one another. We show that $L(C) = \mathsf{LLin}(\pi, \sigma, \alpha, \theta)$ for an adequate choice of parameters. Set $\ell = 2t + 1$ and let p_m be the probability that a cell with t-neighborhood $m \in \Sigma^\ell$ accepts in step t. In addition, let $\pi = \{p_{\ell-1}(w) \mid w \in L(C)\}$ and $\sigma = \{s_{\ell-1}(w) \mid w \in L(C)\}$ as well as $\theta = \log(3/2)$ and $\alpha(m) = \log(1/p_m)$ for $m \in \Sigma^\ell$. The probability that C accepts a word $w \in \Sigma^+$ is

$$\prod_{m \in \Sigma^\ell} p_m^{|w|_m} = \prod_{m \in \Sigma^\ell} \left(\frac{1}{2^{\alpha(m)}}\right)^{|w|_m} = 2^{-f(w)},$$

which is at least $2/3$ if and only if $f(w) \leq \theta$. It follows that $L(C) = \mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta)$.

Step 2. We now relax the requirements from the previous step so that the events of any two cells accepting need no longer be independent from one another. (C still only accepts in the fixed time step t.) Let $K = 2^{O(T^2)}$ be the upper bound from Lemma 15 and $\ell = 2(K+2)T$. Again, we set $\pi = \{p_{\ell-1}(w) \mid w \in L(C)\}$, $\sigma = \{s_{\ell-1}(w) \mid w \in L(C)\}$, and $\theta = \log(3/2)$. As for $\alpha(m)$, we set $\alpha(m) = 0$ unless m is such that there is $d \leq K$ with

$$m = abr_1 s_1 r_2 s_2 \cdots r_d s_d c$$

where $a \in \Sigma^T$ is arbitrary, $b \in \Sigma^{2T}$ contains *no critical cells*, each $r_j \in \Sigma$ is a critical cell (for step t), the $s_j \in \Sigma^{\leq 2T-1}$ are arbitrary, and $c \in \Sigma^*$ has length $|c| \geq T$ and, if c contains any critical cell, then this cell accepts independently from r_d. In addition, we require c to be of maximal length with this property.

Note we need a as context to ensure that b indeed does not contain critical cells (since determining this requires knowledge of the states in the T-neighborhood of the respective cell); the same holds for r_d and c. By construction and Lemma 11, the group of cells $r_1, \dots, r_d$ is such that (although its cells are not necessarily independent from one another) its cells accepts independently from any other critical cell in C. Furthermore, b ensures m aligns properly with the group and that the group does not appear in any other infix. Letting p_m be the probability that every one of the r_j accept, for m as above we set $\alpha(m) = \log(1/p_m)$. Then, as before, the probability that C accepts a word $w \in \Sigma^+$ is $2^{-f(w)}$, which is at least $2/3$ if and only if $f(w) \leq \theta$, thus implying $L(C) = \mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta)$.

Step 3. In this final step we generalize the argument so it also applies to the case where C may accept in any time step $t < T$. First note that, given any $p > 0$, if we set $\theta = \log(1/p)$ in the second step above (instead of $\log(3/2)$), then we have actually shown that

$$\{w \in \Sigma^+ \mid \Pr[C \text{ accepts } w \text{ in step } t] \geq p\} = \mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta).$$

In fact, we can generalize this even further: Given any $\varnothing \neq \tau \subseteq [T]$, by setting α adequately we can consider the acceptance probability for the steps in τ altogether:[8]

$$L(\tau, p) = \{w \in \Sigma^+ \mid \Pr[C \text{ accepts } w \text{ in every step } t \in \tau] \geq p\} = \mathsf{LLin}_\ell(\pi, \sigma, \alpha, \theta).$$

This is because the bound on critical cells of Lemma 15 holds for all steps where C accepts with non-zero probability and, in addition, as defined above m already gives enough context to check if the respective critical cells also accept in any previous step. (That is, we construct m as above by using $t = \max \tau$; however, since the sets of critical cells for different time steps may not be identical, we must also relax the condition for the r_i so that r_i need only be a critical cell in at least one of the time steps of τ.) Since LTT is closed under complement, we then also have

$$\overline{L(\tau, p)} = \{w \in \Sigma^+ \mid \Pr[C \text{ accepts } w \text{ in every step } t \in \tau] < p\} \in \mathsf{LTT}.$$

Fix some input word $w \in \Sigma^+$ to C. For $\varnothing \neq \tau \subseteq [T]$, let Z_τ denote the event where C accepts w in every step $t \in \tau$. By the inclusion-exclusion principle, we have

$$\Pr[C \text{ accepts } w] = \Pr\left[\exists t \in [T] : Z_{\{t\}}\right] = \sum_{\substack{\tau \subseteq [T] \\ |\tau|=1}} \Pr[Z_\tau] - \sum_{\substack{\tau \subseteq [T] \\ |\tau|=2}} \Pr[Z_\tau] + \cdots + (-1)^{T+1} \Pr[Z_{[T]}]$$

(where the probabilities are taken over the coin tosses of C). This means that, if we are somehow given values for $p(\tau) = \Pr[Z_\tau]$ so that the sum above is at least $2/3$, then we can intersect a finite number of $L(\tau, p(\tau))$ languages and their complements and obtain some language that is guaranteed to contain only words in $L(C)$. Concretely, let $p(\tau) \geq 0$ for every $\varnothing \neq \tau \subseteq [T]$ be given so that

$$\sum_{\substack{\varnothing \neq \tau \subseteq [T] \\ |\tau| \text{ odd}}} p(\tau) - \sum_{\substack{\varnothing \neq \tau \subseteq [T] \\ |\tau| \text{ even}}} p(\tau) \geq \frac{2}{3}.$$

[8] Of course we are being a bit sloppy here since Definition 9 demands that a PACA should halt whenever it accepts. What is actually meant is that, having fixed some random input, if we extend the space-time diagram of C on input w so that it spans all of its first T steps (simply by applying the transition function of C), then, for every $t \in \tau$, the t-th line in the diagram contains only accepting cells.

Let $\mathcal{T}_{\text{odd}} = \{\tau \subseteq [T] \mid \tau \neq \varnothing, |\tau| \text{ odd}, p(\tau) > 0\}$ and $\mathcal{T}_{\text{even}} = \{\tau \subseteq [T] \mid \tau \neq \varnothing, |\tau| \text{ even}, p(\tau) > 0\}$. Then necessarily

$$L(p) = \left(\bigcap_{\tau \in \mathcal{T}_{\text{odd}}} L(\tau, p(\tau)) \right) \cap \left(\bigcap_{\tau \in \mathcal{T}_{\text{even}}} \overline{L(\tau, p(\tau))} \right) \subseteq L(C)$$

contains every $w \in L(C)$ for which $\Pr[Z_\tau] \geq p(\tau)$ for $\tau \in \mathcal{T}_{\text{odd}}$ and $\Pr[Z_\tau] \leq p(\tau)$ for $\tau \in \mathcal{T}_{\text{even}}$. The key observation is that *there are only finitely many values the* $\Pr[Z_\tau]$ *may assume*. This is because Z_τ only depends on a finite number of coin tosses, namely the ones in the lightcones of the cells that are critical in at least one of the steps in τ (which, again, is finite due to Lemma 15). Hence, letting P denote the set of all possible mappings of the τ subsets to these values, we may write

$$L(C) = \bigcup_{p \in P} L(p) \in \mathsf{LTT}.$$

Strictness of inclusion. The final statement left to prove is that the inclusion just proven is proper. This is comparatively much simpler to prove. We show that the language

$$L = \{w \in \{0,1\}^+ \mid |w|_1 \geq 2\} \in \mathsf{LTT}$$

cannot be accepted by two-sided error PACA in constant time. For the sake of argument, assume there is such a PACA C with time complexity $T \in \mathbb{N}_+$. Consider which cells in C are critical based on their initial local configuration. Certainly a cell with an all-zeroes configuration 0^{2T-1} cannot be critical. Since $0^n 10^n \notin L(C)$ for any n (but $0^n 10^n 1 \in L(C)$), there must be m_1, m_2 so that $m_1 + m_2 = 2T - 2$ and $c = 0^{m_1} 10^{m_2}$ is the initial local configuration of a critical cell. This means that in

$$x = 0^{2T}(c0^{2T})^{T2^T} \in L$$

we have at least 2^T cells in x that are critical for the same time step $t \in [T]$ (by an averaging argument) and that are also all independent from one another (by Lemma 11). In turn, this implies the following, which contradicts $x \in L(C)$:

$$\Pr[C \text{ accepts } x] \leq \left(1 - 2^{-T}\right)^{2^T} < \frac{1}{e} < \frac{2}{3}. \qquad \blacktriangleleft$$

Real Numbers Equally Compressible in Every Base

Satyadev Nandakumar
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Subin Pulari
Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India

Abstract

This work solves an open question in finite-state compressibility posed by Lutz and Mayordomo [20] about compressibility of real numbers in different bases.

Finite-state compressibility, or equivalently, finite-state dimension, quantifies the asymptotic *lower density of information* in an infinite sequence.

Absolutely normal numbers, being finite-state incompressible in every base of expansion, are precisely those numbers which have finite-state dimension equal to 1 in every base. At the other extreme, for example, every rational number has finite-state dimension equal to 0 in every base.

Generalizing this, Lutz and Mayordomo in [20] (see also Lutz [19]) posed the question: are there numbers which have absolute positive finite-state dimension strictly between 0 and 1 - equivalently, is there a real number ξ and a compressibility ratio $s \in (0,1)$ such that for every base b, the compressibility ratio of the base-b expansion of ξ is precisely s? It is conceivable that there is no such number. Indeed, some works explore "zero-one" laws for other feasible dimensions [11] - *i.e.* sequences with certain properties either have feasible dimension 0 or 1, taking no value strictly in between.

However, we answer the question of Lutz and Mayordomo affirmatively by proving a more general result. We show that given any sequence of rational numbers $\langle q_b \rangle_{b=2}^{\infty}$, we can *explicitly construct* a single number ξ such that for any base b, the finite-state dimension/compression ratio of ξ in base-b is q_b. As a special case, this result implies the existence of absolutely dimensioned numbers for *any* given rational dimension between 0 and 1, as posed by Lutz and Mayordomo.

In our construction, we combine ideas from Wolfgang Schmidt's construction of absolutely normal numbers from [23], results regarding low discrepancy sequences and several new estimates related to exponential sums.

2012 ACM Subject Classification Mathematics of computing → Information theory

Keywords and phrases Finite-state dimension, Finite-state compression, Absolutely dimensioned numbers, Exponential sums, Weyl criterion, Normal numbers

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.48

Related Version *arXiv Version*: https://arxiv.org/abs/2208.06340

Acknowledgements The authors wish to thank anonymous reviewers for helpful suggestions. The authors would also like to thank Jack Lutz and Theodore Slaman for helpful discussions. The second author would like to thank the Institute of Mathematical Sciences, National University of Singapore for their hospitality during the IMS Graduate Summer School in Logic 2022. Parts of this work was completed during the course of the summer school.

1 Introduction

Finite-state compressibility is the lower asymptotic ratio of compression achievable on an infinite string using information-lossless finite-state compressors [28, 25]. Finite-state dimension was originally defined by Dai, Lathrop, Lutz and Mayordomo in [10] using finite-state s-gales, as a finite-state analogue of Hausdorff dimension [18, 2]. Surprisingly, these notions are equivalent [10, 7]. They also have several equivalent characterizations in terms of automatic Kolmogorov complexity [16], finite-state predictors [13], block-entropy rates [13], etc. establishing their mathematical robustness.

© Satyadev Nandakumar and Subin Pulari;
licensed under Creative Commons License CC-BY 4.0

In this work, we solve an open question posed by Lutz and Mayordomo [20] about the existence of numbers whose finite-state compression ratio does not depend on the base of expansion, where the compression ratio is neither 0 nor 1. We explain the context behind this question below.

In [24], Schnorr and Stimm show the celebrated result that finite-state incompressibility in base-b is equivalent to Borel normality in base-b, establishing a deep connection between information theory and metric number theory. Absolutely normal numbers [1, 4, 20, 22, 23], being finite-state incompressible in every base-b, are precisely the set of numbers whose finite-state dimension is 1 in every base. Almost every number in $[0, 1]$ is absolutely normal (see [6]). Several explicit constructions absolutely normal numbers are known [23, 1, 20].

At the other extreme, there are numbers which have finite-state dimension 0 in every base. For example, any rational number in $[0, 1]$ has eventually periodic expansion in any base b and hence, finite-state dimension 0 in any base of expansion.

But, no method for even proving the existence of numbers with absolute finite-state dimension strictly between 0 and 1 are known at the time of writing this paper. Lutz and Mayordomo in [20] ask the following question:

> *Does there exist a real number ξ whose finite-state dimension/compressibility, $\dim^b_{FS}(\xi)$, does not depend on the choice of base b, and such that $0 < \dim^b_{FS}(\xi) < 1$?*

This question was originally posed by Jack Lutz during the session "Alan Turing in the twenty-first century: normal numbers, randomness, and finite automata" in CCR 2012 at the Isaac Newton Institute of Mathematical Sciences, University of Cambridge [19]. In this work, we answer the question in the affirmative, by proving a stronger result: *Given any list $\langle q_b \rangle_{b=1}^{\infty}$ of rationals in $(0, 1]$ (respecting the natural equivalence between bases b), there exists an explicitly constructible number ξ such that $\dim^b_{FS}(\xi) = q_b$ for any base b.* In the special case when every q_b is equal, this provides an affirmative answer (and an explicitly constructible example) to the above question posed by Lutz and Mayordomo.

We now state our main result. Two positive integer bases r and s are said to be equivalent, denoted $r \sim s$, if there are $m, n \in \mathbb{N}$ such that $r^m = s^n$. (Formal definitions follow in Section 2). If $r \sim s$, then we can verify that for any $\xi \in [0, 1]$, $\dim^r_{FS}(\xi) = \dim^s_{FS}(\xi)$. Our main result is the following.

▶ **Theorem 1.** *Let $\langle q_b \rangle_{b=1}^{\infty}$ be a sequence of rationals in $(0, 1]$ such that for any r and s, if $r \sim s$, then $q_r = q_s$. Then, there exists a $\xi \in [0, 1]$ such that for any base b, $\dim^b_{FS}(\xi) = q_b$.*

When $q_b = q$ for every base b, for some $q \in \mathbb{Q} \cap [0, 1]$, we have the following corollary.

▶ **Corollary 2.** *Let $q \in [0, 1] \cap \mathbb{Q}$. Then, there exists a $\xi \in [0, 1]$ such that $\dim^b_{FS}(\xi) = q$ for every base $b \geq 2$.*

For $q \in (0, 1]$, the existence of ξ in the above corollary follows from Theorem 1. For $q = 0$, any rational number ξ in $[0, 1]$ satisfies the required conclusion. Therefore, this corollary provides a positive answer to the question posed by Lutz and Mayordomo in [20].

Explicit constructions of numbers with specific compressibility ratios often use combinatorial techniques (for example, see [8]). Combinatorial constructions are often simpler to understand. However, in our work, we control multiple bases and dimensions in each base. Since this implies working with different alphabets simultaneously, a combinatorial approach to the solution is not easy. We take an approach which involves exponential sums, which allows us to handle the construction requirements successfully.

In this construction, we modify and combine techniques from Wolfgang Schmidt's construction of absolutely normal numbers in [23] along with results regarding low discrepancy sequences [12, 21] and several new estimates. Schmidt's method has been generalized to

construct numbers exhibiting specific kinds of normality (or non-normality) in different bases (see for example [3]). Another interesting such generalization is [5] where Schmidt's method is adapted to demonstrate the pairwise independence of discrepancy functions for non-equivalent bases. Our construction is yet another generalization of Schmidt's method which yields numbers having prescribed rates of information in different bases. While the construction in [5] yields absolutely normal numbers with controlled oscillation of discrepancy rates in different bases, we use results regarding low discrepancy sequences and Schmidt's method to construct numbers with controlled oscillation of block entropy rates in different bases (which are not normal in base b unless $q_b = 1$).

The construction consists of multiple stages. In every stage we arrange a *controlled oscillation* of block entropies in a particular base b between q_b and 1, by fixing digits in base b using multiple *low discrepancy* strings. Meanwhile we *stabilize* the entropies in other non-equivalent bases around 1, using generalizations of bounds from [23] and estimates relating exponential averages to block entropies. We also ensure that the block entropies in different bases do not *fall too much* during the transition between consecutive stages. This is accomplished by using estimates involving low discrepancy strings and lower bounds based on the concavity of the Shannon entropy function ([9]).

The rest of the paper is organized as follows. After the preliminaries in Section 2, we outline the construction and the *requirements* that it should satisfy in Section 3. We also prove that if the requirements hold, then Theorem 1 follows. In Section 4, we develop some important technical tools required for the construction. In Section 5 we describe our stage-wise construction in detail. Finally, in Section 6, we *verify* that the construction in Section 5 satisfies all the *requirements* given in Section 3.

2 Preliminaries

2.1 Basic definitions and notation

We use Σ to denote any finite alphabet. For any natural number $b > 1$, Σ_b denotes the alphabet $\{0, 1, 2, \dots b-1\}$. Observe that for any $b_1 \leq b_2$, we have $\Sigma_{b_1} \subseteq \Sigma_{b_2}$. For any finite alphabet Σ, we use Σ^* to represent the set of finite binary strings and Σ^∞ to represent the set of infinite sequences in alphabet Σ. We use capital letters X, Y to denote infinite strings in any finite alphabet Σ_b. We use small letters w, z to represent finite strings in any finite alphabet Σ_b. For any $X = X_1X_2X_3\dots$ from Σ^∞, we use X_i^{i+n} to denote the substring $X_iX_{i+1}\dots X_{i+n}$ and for any $w \in \Sigma^*$, w_i^{i+n} to represent $w_iw_{i+1}\dots w_{i+n}$.

Greek letters α, β and γ are used to represent constants which are either absolute or dependent on some of the parameters which are relevant in the context of their use. Small letters x, y and the Greek letter ξ are used to denote general real numbers in $[0, 1]$. Small letters b, r and s are used for representing integer bases of expansion greater than or equal to 2. Now, we define the occurrence (sliding) probabilities of finite strings.

▶ **Definition 3.** *We define the* occurrence count *of $z \in \Sigma^*$ in $w \in \Sigma^*$ denoted $N(z, w)$, as $N(z, w) = |\{i \in [1, |w| - |z| + 1] : w_i^{i+|s|-1} = z\}|$. The* occurrence probability *of z in w denoted $P(z, w)$, is defined as $P(z, w) = N(z, w)/(|w| - |z| + 1)$.*

The following is the definition of the base-b finite-state dimension of a sequence ξ using the block entropy characterization of finite-state dimension given in [7], which we use in this work instead of the original definition using s-gales from [10].

▶ **Definition 4** ([10, 7]). *For a given base b and a block length l, we define the* l-length block entropy *over $w \in \Sigma_b^*$ as follows.*

$$H_l^b(w) = -(l\log(b))^{-1} \quad \sum_{z\in\Sigma_b^l} \quad P(z,w)\log P(z,w).$$

Let $\xi \in [0,1]$ and let $X \in \Sigma_b^\infty$ represent the base-b expansion of ξ (for numbers having two base-b expansions, let X denote any one of these). The base-b finite-state dimension *of ξ, denoted $\dim_{FS}^b(\xi)$, is defined by*

$$\dim_{FS}^b(\xi) = \inf_l \liminf_{n\to\infty} H_l^b(X_1^n).$$

Numbers of the form k/b^n for some $k \in \mathbb{Z}$ and $n \in \mathbb{N}$ have two base-b expansions. But their finite-state dimension is equal to 0 irrespective of the infinite sequence chosen as the base-b expansion of x. Therefore, the base-b finite-state dimension is well-defined for every $\xi \in [0,1]$.

▶ Remark. The fact that $\dim_{FS}(x)$ is equivalent to the lower finite-state compressibility of x using lossless finite-state compressors, follows from the results in [28] and [10].

For every $x \in \mathbb{R}$, let $e(x) = e^{2\pi i x}$. For $x \in \mathbb{R}$ and $d > 0$, let $B_d(x)$ denote the open neighborhood of x having radius d, i.e, $B_d(x) = \{y \in \mathbb{R} : |y - x| < d\}$. For any $w \in \Sigma_b^*$, let $v_b(w)$ denote the rational number,$v_b(w) = \sum_{i=1}^{|w|} w_i b^{-i}$. Therefore, the interval $I_w^b = [v_b(w), v_b(w) + b^{-|w|})$ denotes the set of all numbers in $[0,1]$ whose base-b expansion begins with the string w. The *characteristic function* χ_w of a string $w \in \Sigma_b^*$ is defined as, $\chi_w(x) = 1$ if $x \in I_w^b$ and $\chi_w(x) = 0$ otherwise.

The following is a well-known Fourier series approximation for characteristic functions of cylinder sets (see [22, 15]) which acts as the basic *bridge* between the combinatorial and analytic approaches.

▶ **Lemma 5** ([22, 15]). *For any string $w \in \Sigma_b^*$, $\delta > 0$ and $i \in \{1,2\}$, there exists coefficients C_t^i satisfying $\left|C_t^i\right| \le \frac{2}{t^2\delta}$, such that for every real number x,*

$$b^{-|w|} - \delta + \sum_{t\in\mathbb{Z}\setminus\{0\}} C_t^1 e(tx) \quad \le \quad \chi_w(x) \quad \le \quad b^{-|w|} + \delta + \sum_{t\in\mathbb{Z}\setminus\{0\}} C_t^2 e(tx).$$

2.2 Low discrepancy sequences

Let $X_1X_2X_3\ldots$ be the sequence in Σ_b^∞ representing the base-b expansion of $x \in [0,1]$. For any real interval $(\alpha_1,\alpha_2) \subseteq [0,1]$ let $R^b(x,n,\alpha_1,\alpha_2)$ be defined as follows.

$$R^b(x,n,\alpha_1,\alpha_2) = \left|\frac{|\{1 \le i \le n \mid 0.X_iX_{i+1}X_{i+2}\cdots \in (\alpha_1,\alpha_2)\}|}{n} - (\alpha_2 - \alpha_1)\right|$$

The *discrepancy function* $D_n^b(x)$ is defined to be the supremum over all $(\alpha_1,\alpha_2) \subseteq [0,1]$ of $R^b(x,n,\alpha_1,\alpha_2)$ ([12], [21]). The following theorem regarding the low discrepancy of almost every real number follows from the results in [12] and [21].

▶ **Theorem 6** ([12, 21]). *For any base b, there exists a constant C_b such that for almost every x, $\limsup_{n\to\infty} \frac{nD_n^b(x)}{\sqrt{n\log\log n}} < C_b$.*

The following lemma is a corollary of Theorem 6.

▶ **Lemma 7.** *For any base b, there exists a constant C_b such that for any $\epsilon > 0$, there exists $N_b(\epsilon)$ such that outside a set of measure at most ϵ, for any $\alpha_1 < \alpha_2$ and $n \geq N_b(\epsilon)$, we have $R^b(x, n, \alpha_1, \alpha_2)$ is strictly less than $C_b \cdot \frac{\sqrt{\log\log n}}{\sqrt{n}}$.*

The following is a corollary of the above lemma that we need in our construction.

▶ **Corollary 8.** *For any base b, there exists a constant C_b such that for any $\epsilon > 0$, there exists $N_b(\epsilon)$ satisfying the following: for any $n' > N_b(\epsilon)$, every string w of length n' except at most $\epsilon \cdot b^{n'}$ of them is such that given any string z of length at most $n' - N_b(\epsilon) + 1$ and n ranging from $N_b(\epsilon)$ to $n' - |z| + 1$, we have*

$$\left| \frac{N(z, w_1^{n+|z|-1})}{n} - \frac{1}{b^{|z|}} \right| < C_b \cdot \frac{\sqrt{\log\log n}}{\sqrt{n}}. \tag{1}$$

For any base b and $\epsilon = \frac{1}{2}$, let the collection of all strings of length n' satisfying inequality (1) be referred to as $\mathcal{G}_b^{n'}$. Corollary 8 therefore says that for any length $n' > N_b(1/2)$, $|\mathcal{G}_b^{n'}|$ is at least $b^{n'}/2$.

2.3 Schmidt's construction method ([23])

The individual *steps* of the construction in the proof of Theorem 1 are based on the construction method used by Schmidt in his construction of absolutely normal numbers [23]. As far as possible we use Schmidt's notation from [23] in this paper. We give a brief account of Schmidt's method below.

Let $u(1), u(2), u(3), \ldots$ be any sequence of natural numbers. This sequence represents the bases in which we *fix* the digits in each step of our construction. Now, as in [23], define, $\langle m \rangle = \lceil e^{\sqrt{m}} + 2 \cdot u(1) \cdot m^3 \rceil$ and $\langle m; r \rangle = \lceil \langle m \rangle / \log(r) \rceil$. Also define, $a_m = \langle m; u(m) \rangle$ and $b_m = \langle m + 1; u(m) \rangle$. Notice that for any m, if $u(m) = u(m+1)$, we have $b_m = a_{m+1}$. For convenience of notation, we define $\langle 0 \rangle = 0$. Let p be a function from $\mathbb{N}$ to $\mathbb{N}$ such that for every $m \geq 1$, $p(u(m)) \leq u(m)$. The exact function p we use in our construction is defined in section 3. For any $m \geq 1$ and positive real number λ, let $g_m(\lambda)$ denote the smallest natural number such that, $g_m(\lambda) u(m)^{-a_m} \geq \lambda$. Now, define $\eta_m(\lambda) = g_m(\lambda) u(m)^{-a_m}$. We define $\sigma_m(\lambda)$ to be the set of numbers,

$$\eta_m(\lambda) + c_{a_m+1}^{u(m)} u(m)^{-(a_m+1)} + c_{a_m+2}^{u(m)} u(m)^{-(a_m+2)} + \cdots + c_{b_m-2}^{u(m)} u(m)^{-(b_m-2)}$$

with coefficients $c_i^{u(m)}$ taking values from the set $\{0, 1, \ldots u(m) - 1\}$. We define $\sigma_m^*(\lambda)$ to be the set in which the coefficients $c_i^{u(m)}$ takes values from $\{0, 1, \ldots p(u(m)) - 1\}$. Let $\xi_0 = 0$ and let $\xi_1, \xi_2, \xi_3 \ldots$ be a sequence of real numbers such that, $\xi_m \in \sigma_m(\xi_{m-1})$ or $\xi_m \in \sigma_m^*(\xi_{m-1})$.

Since $\xi_m \geq \xi_{m-1}$, it follows that there exists $\xi \in [0, 1]$ such that $\lim_{m\to\infty} \xi_m = \xi$. Furthermore, digits $c_{a_m+1}^{u(m)} c_{a_m+2}^{u(m)} \ldots c_{b_m-2}^{u(m)}$ appears in positions $a_m + 1$ to $b_m - 2$ in the base-$u(m)$ expansion of ξ. This follows as a consequence of the following lemma.

▶ **Lemma 9** ([23])**.** $|\xi - \xi_m| < u(m)^{-(b_m-2)}$.

We refer to the limit ξ as the *constructed number*. The number ξ is uniquely determined by the choice of the sequence $\langle u(m) \rangle$ and the function p. The exact function p we use in our construction is given in section 3 while the sequence $\langle u(m) \rangle$ is defined in a stage-wise manner in sections 3 and 5.

Adapting the technique in [23], we use the following function A_m while choosing ξ_m from $\sigma_m(\xi_{m-1})$ or $\sigma_m^*(\xi_{m-1})$ in the construction of the required number ξ,

$$A_m(x) = \sum_{\substack{t=-m \\ t\neq 0}}^{m} \sum_{\substack{h=1 \\ u(h)\not\sim u(m)}}^{m} \left| \sum_{j=\langle m;u(h)\rangle+1}^{\langle m+1;u(h)\rangle} e(u(h)^{j-1}tx) \right|^2.$$

In our construction in each step, ξ_m is chosen according to either of the following criteria:

1. (Criterion 1.) ξ_m is any element of $\sigma_m^*(\xi_{m-1})$ such that $c_{a_m+1}^{u(m)} c_{a_m+2}^{u(m)} \cdots c_{b_m-2}^{u(m)} \in \mathcal{G}_{p(u(m))}^{b_m-a_m+2}$ with the minimum A_m value among all elements of $\sigma_m^*(\xi_{m-1})$ satisfying this condition.
2. (Criterion 2.) ξ_m is any element of $\sigma_m(\xi_{m-1})$ such that $c_{a_m+1}^{u(m)} c_{a_m+2}^{u(m)} \cdots c_{b_m-2}^{u(m)} \in \mathcal{G}_{u(m)}^{b_m-a_m+2}$ with the minimum A_m value among elements of $\sigma_m(\xi_{m-1})$ satisfying this condition.

The function A_m in Schmidt's paper [23] is defined so that the exponential sums are *minimized* over a sequence of bases $\langle r_i \rangle$ while *fixing digits* (using only 0's and 1's) in another sequence of bases $\langle s_i \rangle$. This ensures that the constructed number ξ is normal in all bases from $\langle r_i \rangle$ while it is not even simply normal in all bases from $\langle s_i \rangle$. The function A_m we use in our construction is defined so that the exponential sums are *minimized* in all bases $u(h) \not\sim u(m)$ for h ranging from 1 to m while *fixing digits* in base $u(m)$ during step number m. Such a definition of A_m is necessary because we need to ensure that the block entropy rate in every base b reaches sufficiently *close* to q_b infinitely often (and it remains *close* to 1 when it is away from q_b). Since the definition of finite-state definition is based on a lim inf, this modified construction yields a number ξ having the required finite-state dimension q_b in every base b. We demonstrate this in sections 3 and 5.

In Lemma 13, we show an upper bound for the function A_m defined above, which is similar to the upper bound in Lemma 7 from [23] for the original definition of A_m. The validity of this upper bound enables us to use Schmidt's technique in our construction with the modified definition of A_m.

3 Overview of the Proof of Theorem 1

We need to construct a number ξ having finite-state dimension equal to q_b in base b for every b. For every $b \geq 2$, let e_b and d_b be natural numbers such that $q_b = e_b/d_b$ in the lowest terms. Below we demonstrate the construction of a number ξ with dimension q_b in base b^{d_b} for every b. This number has dimension equal to q_b in base b for every $b \geq 2$.

Let $\langle r_k \rangle_{k=1}^{\infty}$ be any sequence of natural numbers greater than or equal to 2 such that every equivalence class of numbers (according to the relation $\sim$) has a unique representative in the sequence, which appears infinitely often. Furthermore, we require that $r_1 = 2$ and no consecutive elements in the sequence are equal.

It is straightforward to construct sequences satisfying the above conditions. Given the sequence $\langle q_b \rangle_{b=1}^{\infty}$, define the function $p : \mathbb{N} \to \mathbb{N}$ as $p(b) = \lfloor b^{q_b} \rfloor$. For every $k \geq 1$, define $v(k) = r_k^{d_{r_k}}$ and $v^*(k) = p(v(k))$. From the defining property of the sequence $\langle q_b \rangle_{b=1}^{\infty}$, we get that for any k, $q_{r_k^{d_{r_k}}} = q_{r_k}$. Therefore, $v^*(k) = r_k^{e_{r_k}}$.

Let the sequence $\langle u(m) \rangle$ be initially empty and let $\xi_0 = 0$. At every *step* $m \geq 1$ in the construction, we fix the m^{th} value of the sequence $\langle u(m) \rangle$ and choose ξ_m from $\sigma_m(\xi_{m-1})$ or $\sigma_m^*(\xi_{m-1})$. The whole construction is divided into a sequence of *stages* such that each individual stage comprises of two different *substages*. Each of the *substages* consist of multiple consecutive *steps*.

Suppose by stage $k-1$, $u(1)$, $u(2)$, …, $u(n_{k-1})$ have been determined. Then in the k^{th} stage, we set $u(n_{k-1}+1) = u(n_{k-1}+2) = \cdots = u(n_k) = v(k)$, and fix the digits in the base $v(k)$ expansion of ξ. Hence, $u(1) = v(1) = r_1^{d_{r_1}} = 2^{d_2}$. Within the first substage, at step m, we choose ξ_m from $\sigma_m^*(\xi_{m-1})$ according to Criterion 1. Within the second substage, at step m, we choose ξ_m from $\sigma_m(\xi_{m-1})$ according to Criterion 2.

For any $k \geq 1$, let $X(k)$ denote the infinite sequence in alphabet $\Sigma_{v(k)}$ representing the base-$v(k)$ expansion of ξ. Let P_k^1 denote the final step number contained within the first substage of stage k. Similarly, let P_k^2 denote the final step number contained within the second substage of stage k. For convenience, let $P_0^1 = P_0^2 = 0$. Define $I_1^1 = 1$ and $I_k^1 = \langle P_{k-1}^2 + 1; v(k)\rangle + 1$ for $k > 1$. Now, I_k^1 denotes the index of the initial digit in $X(k)$ fixed during stage k. Also, $F_k^1 = \langle P_k^1 + 1; v(k)\rangle$ denotes the index of the final digit in $X(k)$ fixed during the first substage of stage k. Let $I_k^2 = \langle P_k^1 + 1; v(k)\rangle + 1$ denote the index of the initial digit in $X(k)$ fixed during the second substage of stage k. Finally, let $F_k^2 = \langle P_k^2 + 1; v(k)\rangle$ denote the index of the final digit in $X(k)$ fixed during the second substage of stage k.

The number of individual steps in the stages and substages are ensured to be large enough so that the constructed ξ satisfies the following *requirements*. For every $k \geq 1$, we have the following *end of substage* requirements:

1. $\mathcal{F}_k$: $|H_l^{v(k)}(X(k)_1^n) - q_{r_k}| \leq 2^{-k}$ for every $l \leq k$ when $n = F_k^1$.
2. $\mathcal{S}_{k,1}$: $|H_l^{v(k)}(X(k)_1^n) - 1| \leq 2^{-(k+1)}$ for every $l \leq k$ when $n = F_k^2$.
3. $\mathcal{S}_{k,2}$: If there exists $k' < k$ such that $v(k') = v(k+1)$, then $|H_l^{v(k+1)}(X(k+1)_1^n) - 1| \leq 2^{-k}$ for every $l \leq k$ when $n = \langle P_k^2 + 1; v(k+1)\rangle$.

Requirement $\mathcal{F}_k$ ensures that the block entropies of ξ in base-$v(k)$ are *close* to q_{r_k} by the end of the first substage of stage k. Similarly, $\mathcal{S}_{k,1}$ ensures that $H_l^{v(k)}$ are *close* to 1 by the end of the second substage of stage k. $\mathcal{S}_{k,2}$ ensures that the block entropies of ξ in base-$v(k+1)$ are *close* to 1 before the start of stage $k+1$ (provided that $v(k+1)$ has appeared as $v(k')$ for some $k' < k$).

For every $k > 1$, the following requirement specifies the *stability of non-equivalent base entropies*:

4. $\mathcal{R}_k$: For any $k' < k$ such that $v(k') \not\sim v(k)$, $|H_l^{v(k')}(X(k')_1^n) - 1| \leq 2^{-(k'+1)}$ for every $l \leq k'$ when $\langle P_{k-1}^2 + 1; v(k')\rangle + 1 \leq n \leq \langle P_k^2 + 1; v(k')\rangle$.

Requirement $\mathcal{R}_k$ ensures that the block entropies of ξ in any base $v(k')$ for $k' < k$ which is not equivalent to $v(k)$ remains *stable* around 1 *throughout the course* of stage k.

Two particularly important requirements we need to enforce for every k are the *transition requirements*:

5. $\mathcal{T}_{k,1}$: $H_l^{v(k)}(X(k)_1^n) \geq q_{r_k} - 2^{-(k-1)}$ for every $l \leq k$ when $F_k^1 \leq n \leq F_k^2$.
6. $\mathcal{T}_{k,2}$: If there exists $k' < k$ such that $v(k') = v(k+1)$, then $H_l^{v(k+1)}(X(k+1)_1^n) \geq q_{r_{k+1}} - 2^{-(k-1)}$ for every $l \leq k$ when $I_{k+1}^1 \leq n \leq F_{k+1}^1$.

$\mathcal{T}_{k,1}$ ensures that the base-$v(k)$ entropies do not *fall much below* q_{r_k} during the transition between the first and second substage of stage k. Similarly, $\mathcal{T}_{k,2}$ ensures that the base-$v(k+1)$ entropies do not *fall much below* $q_{r_{k+1}}$ during the transition between stages k and $k+1$.

We now show that if we satisfy the above requirements, then Theorem 1 follows (It will then suffice to show that these requirements are met by our construction).

Proof of Theorem 1. Assume that the construction satisfies the requirements $\mathcal{F}_k$, $\mathcal{S}_{k,1}$, $\mathcal{S}_{k,2}$, $\mathcal{T}_{k,1}$ and $\mathcal{T}_{k,2}$ for every $k \geq 1$ and $\mathcal{R}_k$ for $k > 1$. Let b be an arbitrary base of expansion. Let $\bar{k}$ be the smallest number such that $r_{\bar{k}} \sim b$. Such a *representative* $r_{\bar{k}}$ exists for any b due to the condition imposed on the sequence $\langle r_k\rangle_{k=1}^{\infty}$ at the start of this section. Let $X(\bar{k}) \in \Sigma_{v(\bar{k})}^{\infty}$ denote the base-$v(\bar{k})$ expansion of ξ. In order to prove Theorem 1, it is enough to show that $\inf_l \limsup_{n\to\infty} H_l^{v(\bar{k})}(X(\bar{k})_1^n) = q_{r_{\bar{k}}}$. This implies that, $\dim_{FS}^b(\xi) = \dim_{FS}^{v(\bar{k})}(\xi) = \inf_l \limsup_{n\to\infty} H_l^{v(\bar{k})}(X(\bar{k})_1^n) = q_{r_{\bar{k}}} = q_b$. Here we used the fact that the sequence of rational dimensions is such that $q_r = q_s$ if $r \sim s$. We show that for every $l \geq 1$, $\limsup_{n\to\infty} H_l^{v(\bar{k})}(X(\bar{k})_1^n) = q_{r_{\bar{k}}}$.

Fix any length l. Let k' be any large enough number such that $k' > \max\{\bar{k}, l\}$ and $r_{k'} = r_{\bar{k}}$ (therefore $v(k') = v(\bar{k})$). In the rest of the argument by referring to the index in $X(\bar{k})$ at the *start of stage* k we mean the index $n = \langle P_{k-1}^2 + 1; v(\bar{k})\rangle + 1$, and the index at the *end of stage* k refers to the index $n = \langle P_k^2; v(\bar{k})\rangle$.

Since the requirement $\mathcal{F}_{k'}$ is met, after the first substage of stage k', $H_l^{v(\bar{k})} = H_l^{v(k')}$ is inside $B_{2^{-k'}}(q_{r_{\bar{k}}}) = B_{2^{-k'}}(q_{r_{k'}})$. Since the requirement $\mathcal{S}_{k',1}$ is met, by the end of stage k', $H_l^{v(\bar{k})}$ moves to $B_{2^{-k'}}(1)$ such that at any index n during this transition, $H_l^{v(\bar{k})}(X(\bar{k})_1^n) \geq q_{r_{\bar{k}}} - 2^{-(k'-1)}$. This inequality follows from the fact that $\mathcal{T}_{k',1}$ is satisfied. We know that $\langle r_k\rangle_{k=1}^{\infty}$, satisfies $r_{k'} \neq r_{k'+1}$ (and therefore $v(k') \neq v(k'+1)$). Since $\mathcal{R}_{k'+1}$ is satisfied, during the transition to the next stage and during the course of the next stage, $H_l^{v(\bar{k})}$ remains inside $B_{2^{-(k'+1)}}(1)$. Furthermore, $H_l^{v(\bar{k})}$ remains inside $B_{2^{-(k'+1)}}(1)$ until stage k'' where k'' is the smallest number such that $k'' > k'$ and $r_{k''} = r_{k'} = r_{\bar{k}}$. This follows from the fact that $\mathcal{R}_{k'+i}$ is satisfied for every $i < k'' - k'$. Since, $\mathcal{S}_{k''-1,2}$ is satisfied and $v(\bar{k}) = v(k') = v(k'') = v(k''-1+1)$, by the end of the second substage of stage $k''-1$, $H_l^{v(\bar{k})}$ is inside $B_{2^{-k''}}(1)$. During stage k'', $H_l^{v(\bar{k})}$ starts being inside $B_{2^{-(k'')}}(1)$ and moves to $B_{2^{-k''}}(q_{r_{\bar{k}}})$ (since $\mathcal{F}_{k''}$ is met). During this transition, since $\mathcal{T}_{k''-1,2}$ is met and $v(k'') = v(k''-1+1) = v(k')$ for $k' < k''-1$, it follows that $H_l^{v(\bar{k})}(X(k)_1^n) > q_{r_{\bar{k}}} - 2^{-(k''-2)}$. Therefore, $H_l^{v(\bar{k})}$ remains above $q_{r_{\bar{k}}} - 2^{-(k''-2)}$. Since k' was arbitrary, the above observations together imply that, $\liminf_{n\to\infty} H_l^{v(\bar{k})}(X(k)_1^n) = q_{r_{\bar{k}}}$. This completes the proof of Theorem 1. ◀

Hence, the proof of Theorem 1 is complete if we show the construction of a number ξ satisfying all the above requirements. We demonstrate the construction of ξ and verify that all the requirements are satisfied, in the following sections.

4 Technical Lemmas for the Main Construction

We require two main technical lemmas for the main construction in the proof of Theorem 1. In order to state the first lemma, we require the following generalization of Lemma 5 from [23].

▶ **Lemma 10.** *Consider any two bases r and s. Let K and l be natural numbers such that $\ell \geq s^K$. Then there exists a constant $\alpha(r,s)$ depending only on r and s such that,*

$$\sum_{n=0}^{N-1} \prod_{i=K+1}^{\infty} \left| \frac{\sin(p(s)\pi r^n \ell/s^i)}{p(s)\sin(\pi r^n \ell/s^i)} \right| \leq 2 \cdot N^{1-\alpha(r,s)}.$$

Lemma 5 from [23] is a special case of the above lemma when $p(s) = 2$. We assume that for any r and s, the constant $\alpha(r,s)$ in the above lemma is at most $1/2$ and $\alpha(r,s) = \alpha(s,r)$.

Proof of Lemma 10. The function, $f(x) = |\sin(p(s)\pi x)/p(s)\sin(\pi x)|$ has the limit 1 as $x \to 0$ and $f(x)$ takes values strictly less then 1 when $|x| < 1$. The first property follows from the fact that $\lim_{x\to 0} \sin(x)/x = 1$ and the second fact follows from the inequality $|\sin(nx)| < n|\sin(x)|$, which is easily proved using induction on n when $|x| < \pi$. If x has an obedient digit pair [23] (i.e, a pair of digits in base s that are not both equal to 0 or both equal to $s-1$) then,

$$\left| \frac{\sin(p(s)\pi x/s^i)}{p(s)\sin(\pi x/s^i)} \right| \leq \left| \frac{\sin(p(s)\pi/s^2)}{p(s)\sin(\pi/s^2)} \right| < \gamma_1 < 1.$$

The constant γ_1 above depends only on s (since the function p is fixed). Let $z_K(x)$ denote the number of obedient digit pairs $c_{i+1}c_i$ in x for $i \geq K$ (where $c_{\lceil \log_s(x) \rceil} \dots c_2 c_1$ is the representation of x in base s). Lemma 4 from [23] implies that if $l \geq s^K$, then among the numbers $\ell, \ell r, \ell r^2 \dots \ell r^{N-1}$, there are at most $N^{1-a_{14}}$ numbers such that z_K is smaller than $a_{15} \log N$ (where a_{14} and a_{15} are constants depending only on r and s and independent of ℓ and K). If for some n, ℓr^n has z_K greater than $a_{15} \log N$, then,

$$\prod_{i=K+1}^{\infty} \left| \frac{\sin(p(s)\pi r^n \ell / s^i)}{p(s)\sin(\pi r^n \ell / s^i)} \right| \leq \gamma_1^{a_{15} \log N} = N^{1-\gamma_2}$$

for some $\gamma_2 \in (0,1)$ dependent only on r and s. Now, using the above bound along with Lemma 4 from [23], we get that,

$$\sum_{n=0}^{N-1} \prod_{i=K+1}^{\infty} \left| \frac{\sin(p(s)\pi r^n \ell / s^i)}{p(s)\sin(\pi r^n \ell / s^i)} \right| \leq N^{1-a_{14}} + N^{1-\gamma_2} \leq 2 \cdot N^{1-\alpha(r,s)}$$

for some $\alpha(r,s) \in (0,1)$ dependent only on r and s. ◀

Now, we define the notion of good sequences of natural numbers.

▶ **Definition 11** (Good sequences of natural numbers). *A sequence $\langle u(m) \rangle_{m=1}^{\infty}$ is a* good sequence *of natural numbers if the following conditions are satisfied:*

1. $\prod_{i=b_m-1}^{\infty} \left| \frac{\sin(p(u(m))\pi/2^{i+1})}{p(u(m))\sin(\pi/2^{i+1})} \right| \geq \prod_{i=1}^{\infty} \left| \cos(\pi/2^{i+1}) \right|$ *for every $m > 1$.*
2. $\beta_m \geq \beta_1 \frac{1}{\sqrt[4]{m}}$ *for every $m \geq 1$ where*

$$\beta_m = \min\left(\{\alpha(u(i), u(j)) : 1 \leq i \leq j \leq m \text{ such that } u(i) \not\sim u(j)\} \cup \{1/2\}\right).$$

3. $u(m) \leq u(1)m$ *for every $m \geq 1$.*
4. *For any $m \geq 1$, if there exists any $m' < m$ such that $u(m') \neq u(m)$, then $b_m - a_m \geq \max\{N_{u(m)}(1/2), N_{p(u(m))}(1/2)\}$, where the constants on the right are from Corollary 8.*

From condition 2 and the fact that there does not exist any $u(j) \not\sim u(1)$ with $j \leq 1$, it follows that $\beta_1 = 1/2$. For every i, we use the notation β_i' to denote $\beta_i/2$. The existence of good sequences follows from following lemma and the fact that b_m is increasing in m.

▶ **Lemma 12.** *For any $n \in \mathbb{N}$, the infinite product $\prod_{i=1}^{\infty} \left| \frac{\sin(n\pi/2^{i+1})}{n\sin(\pi/2^{i+1})} \right|$ is convergent.*

Since b_m is strictly increasing in m, any *sufficiently delayed* sequence of natural numbers is a good sequence. Therefore, it is straightforward to verify that given any subset S of $\mathbb{N}$ there is a good sequence $\langle u(m) \rangle$ containing exactly the elements of S such that every element of S occurs infinitely many times in $\langle u(m) \rangle$. This observation is important in our construction.

The first technical lemma is a generalization of the bound on exponential sums given in Lemma 7 from [23].

▶ **Lemma 13.** *Let $\langle u(m) \rangle_{m=1}^{\infty}$ be any good sequence of bases greater than or equal to 2. Let ξ be the real number that is obtained as the limit of $\langle \xi_m \rangle_{m=1}^{\infty}$ where each ξ_m is chosen according to Criterion 1 or 2. Then, there exists a constant δ depending only on $u(1)$ such that for every $m \geq 1$, $A_m(\xi) \leq \delta m^2 (\langle m+1 \rangle - \langle m \rangle)^{2-\beta_m}$.*

The proof of Lemma 13 is similar to that of Lemma 7 from [23], except for the following important differences. We *fix* digits in step m using strings from the subset $\mathcal{G}_{p(u(m))}^{b_m-a_m+2}$ of $\Sigma_{p(u(m))}^{b_m-a_m+2}$. Nevertheless, the upper bounds from [23] are true in our setting up to a multiplicative factor of 2. This follows using the Markov's inequality since $|\mathcal{G}_{p(u(m))}^{b_m-a_m+2}| \geq |\Sigma_{p(u(m))}^{b_m-a_m+2}|/2$ (the argument is similar in the case with $u(m)$ instead of $p(u(m))$). We fix digits in every step m using the larger alphabet $\Sigma_{p(u(m))}$ instead of Σ_2 used in [23]. So, we require upper bounds on products of terms of the form $|\sin(p(u(m))x)/p(u(m))\sin(x)|$ instead of $|\cos(x)|$ as in the proof of Lemma 7 from [23]. These bounds are obtained from Lemma 10 and condition 1 in Definition 11 (since $\langle u(m)\rangle$ is a good sequence). The constant δ depends only on $u(1)$ as a consequence of the *growth control* imposed on the sequence $\langle u(m)\rangle$ in conditions 3 and 4 of Definition 11.

We make crucial use of the following lemma regarding the *uniform normality* of the infinite sequences constructed by choosing successive set of digits from $\mathcal{G}_j^{b_m-a_m-2}$, in the proof of Theorem 1.

▶ **Lemma 14.** *Let b be any base and $j \leq b$. For any finite string $w \in \Sigma_j^*$, $\epsilon > 0$, there exists an integer $L'_{b,j}(w,\epsilon)$ satisfying the following property. If X is any infinite sequence in Σ_j^∞ such that, $X_{\langle m;b\rangle+1}X_{\langle m;b\rangle+2}\ldots X_{\langle m+1;b\rangle-2} \in \mathcal{G}_j^{\langle m+1;b\rangle-\langle m;b\rangle-2}$ for every $m > 0$ and if $T \geq 0$ is any non-negative integer, then for all $n \geq \langle T;b\rangle + L'_{b,j}(w,\epsilon)$, $|P(X^n_{\langle T;b\rangle+1}, w) - j^{-|w|}| \leq \epsilon$.*

Note especially that $L'_{b,j}(w,\epsilon)$ is a constant which depends only on w and ϵ. This constant is independent of the infinite sequence X and the *starting block number* T. When $T = 0$, we have $\langle 0;b\rangle = 0$ and hence the above statement asserts that, $|N(w,X_1^n)/n - j^{-|w|}| \leq \epsilon$ for all $n \geq L'_{b,j}(w,\epsilon)$.

Proof Sketch of Lemma 14. We equivalently prove the existence of a number $L'_{b,j}(w,\epsilon)$ such that

$$\left|\frac{N(w,X^n_{\langle T;b\rangle+1})}{n-\langle T;b\rangle-|w|+1} - \frac{1}{j^{|w|}}\right| \leq \epsilon$$

for all $n \geq \langle T;b\rangle + L'_{b,j}(w,\epsilon)$. To abbreviate the expressions in the proof, let $\hat{\ell}_w = |w| - 1$. . For any $m > 0$, if $\langle m+1;b\rangle - \langle m;b\rangle - 2 \geq N_b(1/2) + N_1^j + \hat{\ell}_w$, then from Corollary 8 it follows that

$$\left|\frac{N(w,X^{\langle m+1;b\rangle-2}_{\langle m;b\rangle+1})}{\langle m+1;b\rangle-\langle m;b\rangle-\hat{\ell}_w} - \frac{1}{j^{|w|}}\right| \leq \epsilon. \tag{2}$$

Since $\langle m+1;b\rangle - \langle m;b\rangle$ is increasing in m, $\langle m+1;b\rangle - \langle m;b\rangle - 2$ is greater than $N_b(1/2) + N_1^j + \hat{\ell}_w$ for all but finitely many m. Let M_2 denote the smallest integer such that $\langle m+1;b\rangle - \langle m;b\rangle - 2 \geq N_b(1/2) + N_1^j + \hat{\ell}_w$ for every $m \geq M_2$. Consider any $n \geq \langle M_2;b\rangle$. Let M' be the (unique) integer such that $\langle M';b\rangle \leq n < \langle M'+1;b\rangle$. We have $M' \geq M_2$. Let N_T denote the following quantity.

$$N_T = \left|\frac{N(w,X^n_{\langle T;b\rangle+1})}{n-\langle T;b\rangle-\hat{\ell}_w+2} - \frac{1}{j^{|w|}}\right|.$$

We first consider the case when $n = \langle M';b\rangle$ for some M'. Then,

$$N_T \leq \frac{S(w,\epsilon)}{n - \langle T;b\rangle - \hat{\ell}_w + 2} \sum_{m=T}^{M_2-1} \frac{\langle m+1;b\rangle - \langle m;b\rangle - \hat{\ell}_w}{S(w,\epsilon)} \left| \frac{N(w, X_{\langle m;b\rangle+1}^{\langle m+1;b\rangle-2})}{\langle m+1;b\rangle - \langle m;b\rangle - \hat{\ell}_w} - \frac{1}{j^{|w|}} \right|$$
$$+ \frac{S'(w,\epsilon)}{n - \langle T;b\rangle - \hat{\ell}_w + 2} \sum_{m=M_2}^{M'-1} \frac{\langle m+1;b\rangle - \langle m;b\rangle - \hat{\ell}_w}{S'(w,\epsilon)} \left| \frac{N(w, X_{\langle m;b\rangle+1}^{\langle m+1;b\rangle-2})}{\langle m+1;b\rangle - \langle m;b\rangle - \hat{\ell}_w} - \frac{1}{j^{|w|}} \right|$$
$$+ \left(1 - \frac{S(w,\epsilon) + S'(w,\epsilon)}{n - \langle T;b\rangle - \hat{\ell}_w + 2}\right)\left(1 + \frac{1}{j^{|w|}}\right)$$

where, $S(w,\epsilon) = \sum_{T \leq m < M_2} \langle m+1;b\rangle - \langle m;b\rangle - \hat{\ell}_w$ and $S'(w,\epsilon) = \sum_{M_2 \leq m < M'} \langle m+1;b\rangle - \langle m;b\rangle - \hat{\ell}_w$. From the definition of M_2 and the properties of $N_b(1/2)$ and N_1^j, the upper bound in (2) is true for every $m \geq M_2$. Therefore, the second term in the upper bound for N_T above is at most ϵ. The sum of the other two terms are shown to be at most 3ϵ for large enough values of n. Hence there exists a large enough number $L'_{b,j}(w,\epsilon)$ such that for any $n \geq \langle T;b\rangle + L'_{b,j}(w,\epsilon)$ satisfying $n = \langle M';b\rangle$ for some M', we have $N_T \leq 4\epsilon$. The case when n is between $\langle M';b\rangle$ and $\langle M'+1;b\rangle$ for some M' follows using similar arguments. ◀

The following is an immediate corollary of the above lemma.

▶ **Corollary 15.** *Let b be any base and $j \leq b$. For any $k > 0$ and $\epsilon > 0$, there exists an integer $L_{b,j}(k,\epsilon)$ satisfying the following property. If X is any infinite sequence in Σ_j^∞ such that,$X_{\langle m;b\rangle+1}X_{\langle m;b\rangle+2}\ldots X_{\langle m+1;b\rangle-2} \in \mathcal{G}_j^{\langle m+1;b\rangle-\langle m;b\rangle-2}$ for every $m > 0$ and if $T \geq 0$ is any non-negative integer, then for every $w \in \Sigma^*$ with $|w| \leq k$ and all $n \geq \langle T;b\rangle + L_{b,j}(k,\epsilon)$, the inequality in Lemma 14 holds.*

5 Main construction

The construction for proving Theorem 1 works by stages. In the k^{th} stage, we *fix* digits in base $v(k)$ by fixing elements of the sequence $\langle u(m)\rangle$ to be $v(k)$. Each stage consists of two substages, which have several steps. In both the substages, we fix $u(m) = v(k)$ for sufficiently large number of steps m. During the first substage, we choose ξ_m from $\sigma_m^*(\xi_{m-1})$ according to Criterion 1 and during the second substage, we choose ξ_m from $\sigma_m(\xi_{m-1})$ according to Criterion 2.

We ensure that $\langle u(m)\rangle$ is a good sequence of natural numbers. During stage 1, $u(1)$ is set to 2^{d_2}, since $r_1 = 2$. All the conditions in the definition of a good sequence are trivially satisfied at this stage. Further checks are performed at the end of the second substage of every stage k to ensure that on transitioning to stage $k+1$, where $u(m)$ is set to $v(k+1)$, none of the conditions in Definition 11 are being violated.

The *duration* of the substages are controlled carefully so that all the requirements given in Section 3 are satisfied. We describe the construction of the two substages of the k^{th} stage in the subsections below.

5.1 First substage of the k^{th} stage

For any $\epsilon > 0$, there exists a constant $\delta_k(\epsilon)$ satisfying the following: If μ_1 and μ_2 are probability distributions over $\Sigma_{v(k)}^l$ for any $l \leq k$ such that $|\mu_1(w) - \mu_2(w)| \leq \delta_k(\epsilon)$ for every $w \in \Sigma_{v(k)}^l$, then $|H(\mu_1) - H(\mu_2)| \leq \epsilon$. The existence of $\delta_k(\epsilon)$ follows from the uniform continuity of the Shannon entropy function (see [14], [9]).

In the first substage of stage k we set $u(m) = v(k)$ for sufficiently large number of m's and choose ξ_m from $\sigma^*_m(\xi_{m-1})$ according to Criterion 1. As in Section 3, let P^1_k denote the index of the last step in the first substage of stage k. Recall that $F^1_k = \langle P^1_k + 1; v(k)\rangle$ denotes the index of the final digit in $X(k)$ fixed during the first substage of stage k. We make P^1_k large enough so that the following conditions are satisfied:

1. $|H^{v(k)}_l(X(k)^n_1) - q_{r_k}| \leq 2^{-k}$ for every $l \leq k$ when $n = F^1_k$.
2. For every $m \geq P^1_k$,

$$b_m - a_m \geq \frac{L_{v(k),v(k)}(k, \delta_k(2^{-k})/2) + 2k}{\min\{\delta_k(2^{-k}), 2^{-k}\}/2} + k. \tag{3}$$

The constant $L_{v(k),v(k)}(k, \delta_k(2^{-k})/2)$ in condition 2 is from Corollary 15. Recall that $v^*(k) = r_k^{e_{r_k}}$. Condition 1 is satisfied for large enough P^1_k because the occurrence probability of any finite string w in alphabet $\{0, 1, \ldots, v^*(k) - 1\}$ converges to $v^*(k)^{-|w|}$ on choosing ξ_m from $\sigma^*_m(\xi_{m-1})$ according to Criterion 1 for sufficiently large number of m's. This follows as a consequence of Lemma 14. Since $b_m - a_m \geq (e^{\sqrt{m+1}} - e^{\sqrt{m}} + m^2)/\log(v(k)) - 1$ and the right hand side of (3) is a constant depending only on k, condition 2 is satisfied for all sufficiently large m.

5.2 Second substage of the k^{th} stage

In the second substage, we set $u(m) = v(k)$ in every step m and choose ξ_m from $\sigma_m(\xi_{m-1})$ according to Criterion 2. In order to describe the construction of the second substage, we need the following technical lemmas.

▶ **Lemma 16.** *Let b be an arbitrary base and $\epsilon > 0$. Let $\langle c_i\rangle_{i=1}^{\infty}$ be any non-increasing sequence of real numbers in $[0, 1]$ such that $c_1 = 1/4$ and $c_i \geq c_1/\sqrt[4]{i}$. Let δ be the constant from Lemma 13. Then, there exists a large enough number $M(\epsilon, b)$ depending only on ϵ and b satisfying the following. For $m \geq M(\epsilon, b)$ and any $l \leq \langle m + 1; b\rangle - \langle m; b\rangle$,*

$$(\langle m; b\rangle + l)^{-1}\left(\delta m \sum\nolimits_{i=1}^{m-1}(\langle i + 1\rangle - \langle i\rangle)^{1-c_i} + l\right) \leq \epsilon.$$

Proof Sketch. We consider the case when $l = 0$. In this case, we have,

$$\delta m \sum_{i=1}^{m-1}(\langle i + 1\rangle - \langle i\rangle)^{1-c_i} \leq \delta m \sum_{i=1}^{m-1}(\langle i + 1\rangle - \langle i\rangle)^{1-c_m} \leq \delta m^2 \langle m\rangle^{1-c_m}.$$

The second inequality follows due to the Hölder's inequality with $p = 1/(1-c_m)$ and $q = 1/c_m$. Therefore,

$$\frac{1}{\langle m; b\rangle}\left(\delta m \sum_{i=1}^{m-1}(\langle i + 1\rangle - \langle i\rangle)^{1-c_i}\right) \leq \delta \log b \frac{m^2}{\langle m\rangle^{c_m}} \leq \delta \log b \frac{m^2}{e^{\sqrt{m}c_m}}.$$

We have $c_m \geq \frac{c_1}{\sqrt[4]{m}} = \frac{1}{4\sqrt[4]{m}}$. Since $m^2 = o(e^{\sqrt[4]{m}/4})$, the right hand side term above converges to 0 for large enough m. The speed of convergence of this term to 0 is independent of the sequence $\langle c_i\rangle$, and hence given any $\epsilon > 0$, there exists a large enough number $M'(\epsilon, b)$ such that for every $m \geq M'(\epsilon, b)$,

$$\frac{1}{\langle m; b\rangle}\left(\delta m \sum_{i=1}^{m-1}(\langle i + 1\rangle - \langle i\rangle)^{1-c_i}\right) \leq \epsilon.$$

The case when $l \neq 0$ follows using similar arguments by proving that for any $\epsilon > 0$, there exists a large enough number $M''(\epsilon, b)$ such that for any $m \geq M''(\epsilon, b)$ and $l \leq \langle m+1; b\rangle - \langle m; b\rangle$,

$$\frac{\langle m+1; b\rangle - \langle m; b\rangle}{\langle m; b\rangle + l} < \epsilon. \qquad \blacktriangleleft$$

▶ **Lemma 17.** *Let b be an arbitrary base, k be any natural number and $\epsilon > 0$. There exists a large enough integer $T(\epsilon, b, k)$ and a positive real number $\gamma(\epsilon, b, k)$ satisfying the following. Let x is any real number in $[0, 1]$ having base-b expansion $X \in \Sigma_b^\infty$. If $|\frac{1}{n}\sum_{i=1}^{n} e(tb^{(j-1)}x)| < \gamma(\epsilon, b, k)$ for every t with $|t| \leq T(\epsilon, b, k)$, then, $\left|H_l^b(X_1^n) - 1\right| < \epsilon$ for every $l \leq k$.*

Proof Sketch. Consider any $w \in \Sigma^*$. Using the Fourier expansion of characteristic functions of cylinder sets given in Lemma 5 ([15]), it is shown that

$$\left|\frac{N(w, X_1^n)}{n} - \frac{1}{b^{|w|}}\right| < \frac{\epsilon}{2} + \sum\nolimits_{t\in\mathbb{Z}\setminus\{0\}} \frac{8}{t^2\epsilon}\frac{|\sum_{j=1}^{n} e(tb^{(j-1)}x)|}{n}$$

where the constant δ is set to $\epsilon/2$. Terms with $t \geq T'(\epsilon) = 64/\epsilon^2$ contribute at most $\epsilon/4$ to the above sum. If the exponential averages for every non-zero parameter t between $-T'(\epsilon)$ and $T'(\epsilon)$ are less than $\gamma'(\epsilon) = \epsilon^2/32 \cdot T'(\epsilon)$, then the sum of the remaining terms is also less than $\epsilon/4$. Therefore $|N(w, X_1^n)/n - b^{-|w|}| < \epsilon$. The constants $T(\epsilon, b, k)$ and $\gamma(\epsilon, b, k)$ are obtained from T' and γ' using the continuity of the Shannon entropy function ([14], [9]). ◀

As in Section 3, let P_k^2 denote the denote the index of the last step in the second substage of stage k. We make P_k^2 large enough so that the following conditions are satisfied at the end of the substage:

1. (Entropy rates in base $v(k)$ are close to 1) $|H_l^{v(k)}(X(k)_1^n) - 1| \leq 2^{-(k+1)}$ for every $l \leq k$ when $n = F_k^2$.
2. (Exponential averages for base $v(k)$ are small) For every t with $|t| \leq T(2^{-(k+1)}, v(k), k)$,

$$\left|(F_k^2)^{-1}\sum\nolimits_{i=1}^{F_k^2} e(tv(k)^{(j-1)}\xi)\right| < \gamma(2^{-(k+1)}, v(k), k)/2. \tag{4}$$

   where T and γ are the constants from Lemma 17.
3. $P_k^2 \geq \max\{M(\gamma(2^{-(k+1)}, v(k), k)/2, v(k)), T(2^{-(k+1)}, v(k), k)\}$, where M is the constant from Lemma 16.
4. (Entropy rates in non-equivalent bases are close to 1) If there exists $k' < k$ such that $v(k') = v(k+1)$, then $|H_l^{v(k+1)}(X(k+1)_1^n) - 1| \leq 2^{-k}$ for every $l \leq k$ when $n = \langle P_k^2 + 1; v(k+1)\rangle$.
5. For every $m \geq P_k^2$,

$$b_m - a_m \geq \frac{L_{v(k+1),v^*(k+1)}(k, \delta_{k+1}(2^{-k})/2) + 2k}{\min\{\delta_k(2^{-k}), \delta_{k+1}(2^{-k}), 2^{-k}\}/2} + k \tag{5}$$

6. The sequence $\langle u'(m)\rangle_{m=1}^{\infty}$ defined such that $u'(m) = u(m)$ for every $m \leq P_k^2$ and $u'(m) = v(k+1)$ for $m \geq P_k^2 + 1$, is a good sequence.

The constant $L_{v(k+1),v^*(k+1)}(k, \delta_{k+1}(2^{-k})/2)$ in condition 5 is from Corollary 15. Condition 1 is satisfied for large enough P_k^2 because the occurrence probability of any finite string $w \in \Sigma_{v(k)}^*$ converges to $v(k)^{-|w|}$ on choosing ξ_m from $\sigma_m(\xi_{m-1})$ according to Criterion 2 for sufficiently large number of m's, as a consequence of Lemma 14. For any t, on extending the second substage by increasing P_k^2, the corresponding exponential averages in (4) converges to 0 as a consequence of the Weyl Criterion for normality (see [27, 17])

and Lemma 14. Therefore, condition 2 is satisfied for large enough values of P_k^2. Since $b_m - a_m \geq (e^{\sqrt{m+1}} - e^{\sqrt{m}} + m^2)/\log(v(k)) - 1$ and the right hand side of (5) is a constant depending only on k, condition 5 is satisfied for all sufficiently large m.

It is easily verified from the definition of a good sequence that for large enough P_k^2, on setting $u(P_k + 1) = v(k + 1)$ the sequence $\langle u(m)\rangle_{m=1}^{P_k^2+1}$ satisfies all the conditions in the definition of good sequences (Definition 11). On extending the sequence from this value of $P_k^2 + 1$ onwards, by setting $u(m) = v(k + 1)$ for every $k \geq P_k^2 + 2$, none of the conditions in Definition 11 are violated. Therefore, condition 6 is satisfied for all large enough values of P_k^2. During stage 1, all the conditions in the definition of a good sequence are trivially satisfied. Therefore, the validity of condition 6 at the end of every second substage, inductively ensures that the constructed sequence $\langle u(m)\rangle$ is a good sequence.

Proving that condition 4 holds for large enough values of P_k^2, requires an argument using Lemmas 16 and 17.

▶ **Lemma 18.** *Condition 4 in the construction is true for all large enough values of P_k^2.*

Proof Sketch. From the statement of condition 4, we have $v(k') = v(k + 1)$ for some $k' < k$. Recall that the sequence $\langle r_k\rangle_{k=1}^{\infty}$ satisfies $r_k \neq r_{k+1}$. Therefore, we get $v(k) \neq v(k + 1)$ and hence $v(k') \not\sim v(k)$. Consider any index m of $\langle u(m)\rangle$ that is set during the second substage of stage k. Since ξ_m was chosen from $\sigma_m(\xi_{m-1})$ using Criterion 2, from Lemma 13, we get that for any non-zero t with $|t| < m$,

$$\left|\sum_{j=\langle m;v(k+1)\rangle+1}^{\langle m+1;v(k+1)\rangle} e(v(k+1)^{j-1}t\xi)\right| \leq \delta m(\langle m+1\rangle - \langle m\rangle)^{1-\beta'_m}.$$

Now, the convergence of base $v(k + 1)$ entropies to 1 for large enough P_k^2 is obtained by using Lemma 16 (for an appropriately defined sequence $\langle c_i\rangle$) followed by an application of Lemma 17. ◀

6 Verification

In this section we prove that all the requirements given in section 3 are satisfied by the construction in section 5. The following proofs along with the argument provided at the end of Section 3 complete the proof of Theorem 1.

▶ **Lemma 19.** *For all $k \geq 1$, the requirements $\mathcal{F}_k$, $\mathcal{S}_{k,1}$ and $\mathcal{S}_{k,2}$ are met by the construction.*

Proof. $\mathcal{F}_k$ follows from the validity of condition 1 from section 5.1 at the end of the first substage of every stage k. From the validity of condition 1 from section 5.2 at the end of the second substage of every stage k, we get that $|H_l^{v(k)}(X(k)_1^n) - 1| \leq 2^{-(k+1)}$ for every $l \leq k$ when $n = F_k^2$. Therefore, $\mathcal{S}_{k,1}$ is satisfied for every $k \geq 1$. $\mathcal{S}_{k,2}$ follows directly from the validity of condition 4 from section 5.2 at the end of the second substage of every stage k. ◀

▶ **Lemma 20.** *For every $k > 1$, the requirement $\mathcal{R}_k$ is satisfied by the construction.*

Proof Sketch. Let k' be any stage number below k such that $v(k') \not\sim v(k)$. Without loss of generality, let us assume that there does not exist any k'' between k' and k such that $v(k'') = v(k')$. Since the equivalence class of base $v(k')$ has a unique representative in the sequence $\langle r_k\rangle_{k=1}^{\infty}$, we also get that $v(k'') \not\sim v(k')$ for any k'' between k' and k. Let m denote any index such that $P_{k'}^2 + 1 \leq m \leq P_k^2$. Using condition 3 in Section 5.2 and Lemma 13, we obtain,

$$\left|\sum_{j=\langle m;v(k')\rangle+1}^{\langle m+1;v(k')\rangle} e(v(k')^{j-1}t\xi)\right| \leq \delta m(\langle m+1\rangle - \langle m\rangle)^{1-\beta'_m}.$$

for every t with $|t| \leq T(2^{-(k'+1)}, v(k'), k')$. Since condition 2 from Section 5.2 is valid at the end of the second substage of stage k', we have,

$$\left|(F^2_{k'})^{-1} \sum_{i=1}^{F^2_{k'}} e(tv(k')^{(j-1)}\xi)\right| < \gamma(2^{-(k'+1)}, v(k'), k')/2$$

for every t with $|t| \leq T(2^{-(k'+1)}, v(k'), k')$. By combining the above inequalities and Lemma 16 (for an appropriately defined sequence $\langle c_i \rangle$), precise bounds are obtained for the exponential averages in base $v(k')$ along stage k. The proof of the lemma now follows by invoking Lemma 17 using these new bounds. ◀

▶ **Lemma 21.** *For every $k \geq 1$, the requirement $\mathcal{T}_{k,2}$ is met by the construction.*

Proof Sketch. We assume that there exists $k' < k$ such that $v(k') = v(k+1)$. Otherwise, $\mathcal{T}_{k,2}$ is vacuously satisfied. Let $F^2_k = \langle P^2_k; v(k+1) \rangle$ denote the index of the final digit fixed during the second substage of stage k in the base $v(k+1)$ expansion of ξ. Let $n \geq F^2_k$ denote the index of any digit that is fixed during the first substage of stage $k+1$ in the base $v(k+1)$ expansion of ξ. For every $w \in \Sigma^*_s$, let $\mathbb{P}^{k+1}(w, j_1, j_2)$ denote the fraction of $|w|$-length blocks containing w among the digits in the base-$v(k+1)$ expansion of ξ with indices in the range j_1 to j_2. Then, $\mathbb{P}^{k+1}(w, 1, n)$ is equal to

$$\frac{F^2_k - |w| + 1}{n - |w| + 1} \mathbb{P}^{k+1}(w, 1, F^2_k) + \frac{n - F^2_k - |w| + 1}{n - |w| + 1} \mathbb{P}^{k+1}(w, F^2_k + 1, n) + o(1/n). \tag{6}$$

The above error term is negligible for $n \geq F^2_k$. If $n - F^2_k \leq L_{v(k+1),v^*(k+1)}(k, \delta_{k+1}(2^{-k})/2)$ then second term above is easily verified to be negligible using inequality (5). Also, since there exists $k' < k$ such that $v(k') = v(k+1)$ and the requirement $\mathcal{S}_{k,2}$ is satisfied, we get,

$$\left|(l \log v(k+1))^{-1} \quad H(\mathbb{P}^{k+1}_l(\cdot, 1, F^2_k)) - 1\right| \leq 2^{-k}. \tag{7}$$

Hence, the required conclusion follows. Otherwise, let $n - F^2_k > L_{v(k+1),v^*(k+1)}(k, \delta_{k+1}(2^{-k})/2)$. Observe that $\log v^*(k+1)/\log v(k+1) = q_{r_{k+1}}$. From Lemma 14 and the definition of $\delta_{k+1}(2^{-k})$ we obtain that,

$$\left|(l \log v(k+1))^{-1} H(\mathbb{P}^{k+1}_l(\cdot, F^2_k + 1, n)) - q_{r_{k+1}}\right| \leq 2^{-k}. \tag{8}$$

for any $l \leq k$. Now, from the concavity of the Shannon entropy function (see [9]) and (6) we get that for any $l \leq k$,

$$H(\mathbb{P}^{k+1}_l(\cdot, 1, n)) \geq \frac{F^2_k - l + 1}{n - l + 1} H(\mathbb{P}^{k+1}_l(\cdot, 1, F^2_k)) + \frac{n - F^2_k - l + 1}{n - l + 1} H(\mathbb{P}^{k+1}_l(\cdot, F^2_k + 1, n)).$$

Combining inequalities (7) and (8) with the above lower bound, we get,

$$\frac{1}{l \log v(k+1)} H(\mathbb{P}^{k+1}_l(\cdot, 1, n)) \geq \lambda_{l,k} \times 1 + (1 - \lambda_{l,k}) \times q_{r_{k+1}} - \frac{1}{2^{k-1}} \geq q_{r_k} - \frac{1}{2^{k-1}}$$

where $\lambda_{l,k} = (F^2_k - l + 1)/(n - l + 1)$. Therefore, the requirement $\mathcal{T}_{k,2}$ is satisfied for every $k \geq 1$. ◀

▶ **Lemma 22.** *For every $k \geq 1$, the requirement $\mathcal{T}_{k,1}$ is met by the construction.*

Lemma 22 is proved using the same techniques as in the proof of Lemma 21. The fact that the requirement $\mathcal{F}_k$ is satisfied (as shown in Lemma 19) is used along with inequality (3) and the *concavity* of the Shannon entropy, to ensure that the block entropies in base $v(k)$ does not fall below $q_{r_k} - 2^{-(k-1)}$ during the transition between the first and second substages of stage k.

7 Discussion and Open Questions

It is open whether our main results are true in the setting of finite-state strong dimension ([2, 7]). In particular: *Does there exist an absolutely strong dimensioned number with finite-state strong dimension strictly between* 0 *and* 1*?*. The strong dimension of ξ is 1. It is unclear how to modify our construction to bound the limit superior of the block entropies away from 1. This is because while we control the block entropies in base $v(k)$ during stage k, the block entropies in all bases with $k' < k$ and $v(k') \not\sim v(k)$ are converging to 1 (since the exponential averages in these bases are *getting smaller* after every individual step within stage k). This behavior seems to be an essential feature of constructions based on Schmidt's method [23]. Hence, the question regarding *absolute strong dimension*, if such a number exists, may require new construction techniques that are capable of stabilizing block entropies in non-equivalent bases simultaneously around values strictly between 0 and 1.

References

1 Christoph Aistleitner, Verónica Becher, Adrian-Maria Scheerer, and Theodore A. Slaman. On the construction of absolutely normal numbers. *Acta Arith.*, 180(4):333–346, 2017. doi:10.4064/aa170213-5-8.

2 Krishna B Athreya, John M Hitchcock, Jack H Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM journal on computing*, 37(3):671–705, 2007.

3 Verónica Becher, Yann Bugeaud, and Theodore A. Slaman. On simply normal numbers to different bases. *Math. Ann.*, 364(1-2):125–150, 2016. doi:10.1007/s00208-015-1209-9.

4 Verónica Becher, Pablo Ariel Heiber, and Theodore A. Slaman. A computable absolutely normal Liouville number. *Math. Comp.*, 84(296):2939–2952, 2015. doi:10.1090/mcom/2964.

5 Verónica Becher and Theodore A. Slaman. On the normality of numbers to different bases. *Journal of the London Mathematical Society*, 90(2):472–494, 2014. doi:10.1112/jlms/jdu035.

6 Patrick Billingsley. *Probability and measure*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, Inc., New York, third edition, 1995. A Wiley-Interscience Publication.

7 Chris Bourke, John M. Hitchcock, and N. V. Vinodchandran. Entropy rates and finite-state dimension. *Theoretical Computer Science*, 349(3):392–406, 2005.

8 D. G. Champernowne. Construction of decimals normal in the scale of ten. *J. London Math. Soc.*, 2(8):254–260, 1933.

9 T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New York, N.Y., 1991.

10 Jack J. Dai, James I. Lathrop, Jack H. Lutz, and Elvira Mayordomo. Finite-state dimension. *Theoretical Computer Science*, 310(1-3):1–33, 2004.

11 Lance Fortnow, John M. Hitchcock, A. Pavan, N. V. Vinodchandran, and Fengming Wang. Extracting Kolmogorov complexity with applications to dimension zero-one laws. *Inform. and Comput.*, 209(4):627–636, 2011. doi:10.1016/j.ic.2010.09.006.

12 S. Gaal and L. Gál. The discrepancy of the sequence $\{(2^n x)\}$. *Nederl. Akad. Wetensch. Proc. Ser. A 67 = Indag. Math.*, 26:129–143, 1964.

13 John M. Hitchcock. Fractal dimension and logarithmic loss unpredictability. *Theoretical Computer Science*, 304(1):431–441, 2003. doi:10.1016/S0304-3975(03)00138-5.

14 A. Ya. Khinchin. *Mathematical Foundations of Information Theory*. Dover Publications, 1957.

15 J. F. Koksma. *Diophantische Approximationen*. Springer-Verlag, Berlin-New York, 1974. Reprint.

16 Alexander Kozachinskiy and Alexander Shen. Automatic Kolmogorov complexity, normality, and finite-state dimension revisited. *Journal of Computer and System Sciences*, 118:75–107, 2021. doi:10.1016/j.jcss.2020.12.003.

17 L. Kuipers and H. Niederreiter. *Uniform distribution of sequences*. Pure and Applied Mathematics. Wiley-Interscience [John Wiley & Sons], New York-London-Sydney, 1974.

18 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003. doi:10.1137/S0097539701417723.

19 Jack H. Lutz. Alan turing in the twenty-first century: normal numbers, randomness, and finite automata. *CCR 2012: 7th conference on computability, complexity and randomnes*, July 2012. URL: https://www.newton.ac.uk/seminar/2265/.

20 Jack H. Lutz and Elvira Mayordomo. Computing absolutely normal numbers in nearly linear time. *Inform. and Comput.*, 281:Paper No. 104746, 12, 2021. doi:10.1016/j.ic.2021.104746.

21 Walter Philipp. Limit theorems for lacunary series and uniform distribution mod 1. *Acta Arith.*, 26(3):241–251, 1974/75. doi:10.4064/aa-26-3-241-251.

22 Wolfgang M. Schmidt. On normal numbers. *Pacific J. Math.*, 10:661–672, 1960. URL: http://projecteuclid.org/euclid.pjm/1103038420.

23 Wolfgang M. Schmidt. Über die Normalität von Zahlen zu verschiedenen Basen. *Acta Arith.*, 7:299–309, 1961/62. doi:10.4064/aa-7-3-299-309.

24 C. P. Schnorr and H. Stimm. Endliche automaten und zufallsfolgen. *Acta Informatica*, 1:345–359, 1972.

25 D. Sheinwald. On the Ziv-Lempel proof and related topics. *Proceedings of the IEEE*, 82(6):866–871, 1994. doi:10.1109/5.286190.

26 Elias M. Stein and Rami Shakarchi. *Complex analysis*, volume 2 of *Princeton Lectures in Analysis*. Princeton University Press, Princeton, NJ, 2003.

27 Hermann Weyl. Über die Gleichverteilung von Zahlen mod. Eins. *Math. Ann.*, 77(3):313–352, 1916. doi:10.1007/BF01475864.

28 J. Ziv and A. Lempel. Compression of individual sequences via variable rate coding. *IEEE Transaction on Information Theory*, 24:530–536, 1978.

A Proofs from Section 2

A.1 Proofs from Subsection 2.2

Proof of Lemma 7. Theorem 6 implies that for almost every x, there exists a minimum number N_x such that,

$$R^b(x, n, \alpha_1, \alpha_2) < C_b \cdot \frac{\sqrt{\log\log N}}{\sqrt{n}}$$

for any $\alpha_1 < \alpha_2$ and every $n \geq N_x$. For every $i \in \mathbb{N}$, define $U_i = \{x : N_x \geq i\}$. U_i's are a monotonically decreasing sequence of sets such that $\mu(\bigcap_{i\in\mathbb{N}} U_i) = 0$. Using the continuity of measure from above (see [6]), there exists a large enough $N_b(\epsilon)$ such that $\mu(U_i) < \epsilon$ for every $i \geq N_b(\epsilon)$. This completes the proof of the lemma. ◀

Proof of Corollary 8. For any $z \in \Sigma^*$, $I_z^b = [v_b(z), v_b(z) + b^{-|z|})$ represents the interval containing exactly those real numbers in $[0, 1]$ whose base-b expansion starts with the string z. Setting $\alpha_1 = v_b(z)$ and $\alpha_2 = v_b(z)+b^{-|z|}$, it follows from Lemma 7 that for $x = 0.X_1X_2X_3\ldots$ outside a set of measure at most ϵ,

$$\left|\frac{N(z, X_1^{n+|z|-1})}{n} - \frac{1}{b^{|z|}}\right| < C_b\frac{\sqrt{\log\log n}}{\sqrt{n}} \tag{9}$$

for $n' \geq N_b(\epsilon)$, string z of length less than $n' - N_b(\epsilon) + 1$ and n ranging from $N_b(\epsilon)$ to $n' - |z| + 1$. This implies that the number of strings of length $n' \geq N_b(\epsilon)$ such that,

$$\left|\frac{N(z, X_1^{n+|z|-1})}{n} - \frac{1}{b^{|z|}}\right| \geq C_b\frac{\sqrt{\log\log n}}{\sqrt{n}} \tag{10}$$

for some string z of length less than $n' - N_b(\epsilon) + 1$ and some n between $N_b(\epsilon)$ to $n' - |z| + 1$ is at most $\epsilon \cdot b^n$. If this is not the case, then the union of the cylinder sets corresponding to the strings violating (9) is a set of measure greater than ϵ in which each element violates (9). Since, this leads to a contradiction, the number of strings of length $n' \geq N_b(\epsilon)$ such that (10) holds, is at most $\epsilon \cdot b^n$. This completes the proof of the corollary. ◀

A.2 Proofs from Subsection 2.3

Proof of Lemma 9. For every m, $\xi_m \geq \xi_{m-1}$ and thus we have,

$$\begin{aligned} |\xi_m - \xi_{m-1}| &= \xi_m - \xi_{m-1} \\ &= \xi_m - \eta_m(\xi_{m-1}) + \eta_m(\xi_{m-1}) - \xi_{m-1} \\ &\leq \frac{1}{u(m)^{a_m}} + \frac{1}{u(m)^{a_m}} = \frac{2}{u(m)^{a_m}}. \end{aligned}$$

Then,

$$\sum_{i=m+1}^{\infty} \frac{1}{u(i)^{a_i}} \leq \frac{1}{e^{\langle m+1 \rangle}}(1 + \frac{1}{e^2} + \frac{1}{e^4} + \dots) < \frac{3}{2}\frac{1}{e^{\langle m+1 \rangle}} < \frac{3}{2}\frac{1}{u(m)^{b_m - 1}} \leq \frac{1}{2}\frac{1}{u(m)^{b_m - 2}}.$$

Therefore, the limit ξ satisfies,

$$0 \leq \xi - \xi_m = |\xi - \xi_m| < \frac{1}{u(m)^{b_m - 2}}.$$ ◀

B Proofs from Section 4

In order to prove Lemma 12, we need the following inequality.

▶ **Lemma 23.** *For every $n \geq 2$ and x with $|x| < 1$,*

$$\frac{\sin(nx)}{n\sin(x)} \geq 1 - \frac{(n^2-1)x^2}{6}.$$

Proof of Lemma 23. We prove the statement using induction on n. Consider the base case when $n = 2$. We have,

$$\frac{\sin(2x)}{2\sin(x)} = \cos(x) \geq 1 - \frac{x^2}{2} = 1 - \frac{(2^2-1)x^2}{6}.$$

The inequality $\cos(x) \geq 1 - x^2/2$ follows easily from the Taylor series expansion of $\cos(x)$ since $|x| < 1$.

Assume that the statement in the conclusion holds for arbitrary n. We now show that this implies the required conclusion for $n + 1$. We have,

$$\begin{aligned} \frac{\sin((n+1)x)}{(n+1)\sin(x)} &= \frac{\sin(nx)\cos(x) + \cos(nx)\sin(x)}{(n+1)\sin(x)} \\ &= \frac{\sin(nx)}{n\sin(x)} \cdot \frac{n}{n+1} \cdot \cos(x) + \frac{1}{n+1} \cdot \cos(nx). \end{aligned}$$

Using the induction hypothesis and the inequality $\cos(x) \geq 1 - x^2/2$, we get,

$$\begin{aligned}
\frac{\sin((n+1)x)}{(n+1)\sin(x)} &\geq \left(1 - \frac{(n^2-1)x^2}{6}\right) \cdot \frac{n}{n+1} \cdot \left(1 - \frac{x^2}{2}\right) + \frac{1}{n+1} \cdot \left(1 - \frac{n^2x^2}{2}\right) \\
&> \frac{n}{n+1} - \frac{1}{n+1} \cdot \frac{(n^3 - n + 3n)x^2}{6} + \frac{1}{n+1} - \frac{1}{n+1} \cdot \frac{n^2x^2}{2} \\
&= 1 - \frac{1}{n+1} \cdot \frac{(n^3 + 3n^2 + 2n)x^2}{6} \\
&= 1 - \frac{1}{n+1} \frac{(n+1)(n^2+2n)x^2}{6} \\
&= 1 - \frac{(n^2+2n)x^2}{6} \\
&= 1 - \frac{((n+1)^2 - 1)x^2}{6}
\end{aligned}$$

The lemma now follows due to induction. ◀

Now, we prove Lemma 12.

Proof of Lemma 12. From Lemma 23, we get that,

$$\left|\frac{\sin(nx)}{n\sin(x)} - 1\right| \leq \frac{(n^2-1)x^2}{6}.$$

Now,

$$\left|\frac{\sin(nx)}{n\sin(x)} - 1\right| \leq \frac{(n^2-1)x^2}{6}.$$

This implies that,

$$\sum_{i=1}^{\infty} \left|\frac{\sin(n\pi/2^{i+1})}{n\sin(\pi/2^{i+1})} - 1\right| < \infty.$$

Now, using Proposition 3.1 from [26], it follows that the infinite product is convergent. The convergence of $\prod_{i=1}^{\infty} \left|\cos(\pi/2^{i+1})\right|$ also follows from this argument since,

$$\cos\left(\pi/2^{i+1}\right) = \frac{\sin\left(2\pi/2^{i+1}\right)}{2\sin\left(\pi/2^{i+1}\right)}.$$ ◀

C Proofs from Section 6

Full Proof of Lemma 20. Let k' be any stage number below k such that $v(k') \not\sim v(k)$. Without loss of generality, let us assume that there does not exist any k'' between k' and k such that $v(k'') = v(k')$. Since the equivalence class of base $v(k')$ has a unique representative in the sequence $\langle r_k \rangle$, we also get that $v(k'') \not\sim v(k')$ for any k'' between k' and k. Since condition 2 from Section 5.2 is valid at the end of the second substage of stage k', we have,

$$\left|\frac{1}{F_{k'}^2} \sum_{i=1}^{F_{k'}^2} e(tv(k')^{(j-1)}\xi)\right| < \frac{\gamma(2^{-(k'+1)}, v(k'), k')}{2} \tag{11}$$

for every t with $|t| \leq T(2^{-(k'+1)}, v(k'), k')$. From the validity of condition 3 from Section 5.2 at the end of the second substage of stage k', we have,

$$P^2_{k'} \geq \max\{M(\gamma(2^{-(k'+1)}, v(k'), k')/2, v(k')), T(2^{-(k'+1)}, v(k'), k')\}.$$

Let m denote any index such that $P^2_{k'} + 1 \leq m \leq P^2_k$. From Lemma 13, we get that,

$$\sum_{\substack{t=-m \\ t\neq 0}}^{m} \sum_{\substack{h=1 \\ u(h)\not\sim u(m)}}^{m} \left| \sum_{j=\langle m;u(h)\rangle+1}^{\langle m+1;u(h)\rangle} e(u(h)^{j-1}t\xi) \right|^2 \leq \delta m^2 (\langle m+1\rangle - \langle m\rangle)^{2-\beta_m}$$

for every t with $|t| \leq m$. Since for every $k'' \in [k'+1, k]$, $v(k'') \not\sim v(k')$, from the above we obtain,

$$\left| \sum_{j=\langle m;v(k')\rangle+1}^{\langle m+1;v(k')\rangle} e(v(k')^{j-1}t\xi) \right| \leq \delta m (\langle m+1\rangle - \langle m\rangle)^{1-\beta'_m}.$$

Define the sequence $\langle c_i\rangle_{i=1}^{\infty}$ as $c_i = \beta'_i$. It is easily verified that $\langle c_i\rangle$ is a non-increasing sequence satisfying $c_1 = 1/4$ and $c_i \geq c_1/\sqrt[4]{i}$ for every $i \geq 1$. Consider any m satisfying $P^2_{k'} + 1 \leq m \leq P^2_k$ and $l \leq \langle m+1; v(k')\rangle - \langle m; v(k')\rangle$. Now, for any t with $|t| \leq T(2^{-(k'+1)}, v(k'), k') \leq m$, we have,

$$\begin{aligned}
&\frac{1}{\langle m; v(k')\rangle + l} \left| \sum_{j=1}^{\langle m;v(k')\rangle+l} e(v(k')^{j-1}t\xi) \right| \\
&= \frac{1}{\langle m; v(k')\rangle + l} \left| \sum_{i=1}^{\langle P^2_{k'};v(k')\rangle} e(tv(k')^{(j-1)}\xi) \right| \\
&+ \frac{1}{\langle m; v(k')\rangle + l} \left| \sum_{j=\langle P^2_{k'};v(k')\rangle+1}^{\langle m;v(k')\rangle+l} e(v(k')^{j-1}t\xi) \right| \\
&= \frac{1}{\langle m; v(k')\rangle + l} \left| \sum_{i=1}^{F^2_{k'}} e(tv(k')^{(j-1)}\xi) \right| + \frac{1}{\langle m; v(k')\rangle + l} \left| \sum_{j=\langle P^2_{k'};v(k')\rangle+1}^{\langle m;v(k')\rangle+l} e(v(k')^{j-1}t\xi) \right| \\
&\leq \left| \frac{1}{F^2_{k'}} \sum_{i=1}^{F^2_{k'}} e(tv(k')^{(j-1)}\xi) \right| + \frac{\sum_{i=P^2_{k'}}^{m-1} (\langle i+1\rangle - \langle i\rangle)^{1-\beta'_i} + l}{\langle m; v(k')\rangle + l} \\
&\leq \frac{\gamma(2^{-(k'+1)}, v(k'), k')}{2} + \frac{\sum_{i=1}^{m-1} (\langle i+1\rangle - \langle i\rangle)^{1-c_i} + l}{\langle m; v(k')\rangle + l} \\
&\leq \frac{\gamma(2^{-(k'+1)}, v(k'), k')}{2} + \frac{\gamma(2^{-(k'+1)}, v(k'), k')}{2} \\
&= \gamma(2^{-(k'+1)}, v(k'), k').
\end{aligned}$$

The second last inequality above follows from Lemma 16 since,

$$m \geq P^2_{k'} \geq M(\gamma(2^{-(k'+1)}, v(k'), k')/2, v(k')).$$

Finally, from the above inequalities and Lemma 17 we obtain that, $|H_l^{v(k')}(X(k')_1^n) - 1| \leq 2^{-(k'+1)}$ for every $l \leq k'$ when $\langle P^2_{k-1} + 1; v(k')\rangle + 1 \leq n \leq \langle P^2_k + 1; v(k')\rangle$. ◀

Gap Preserving Reductions Between Reconfiguration Problems

Naoto Ohsaka
CyberAgent, Inc., Tokyo, Japan

Abstract

Combinatorial reconfiguration is a growing research field studying problems on the transformability between a pair of solutions for a search problem. For example, in SAT Reconfiguration, for a Boolean formula φ and two satisfying truth assignments σ_{s} and σ_{t} for φ, we are asked to determine whether there is a sequence of satisfying truth assignments for φ starting from σ_{s} and ending with σ_{t}, each resulting from the previous one by flipping a single variable assignment. We consider the approximability of *optimization variants* of reconfiguration problems; e.g., Maxmin SAT Reconfiguration requires to maximize the minimum fraction of satisfied clauses of φ during transformation from σ_{s} to σ_{t}. Solving such optimization variants approximately, we may be able to obtain a reasonable reconfiguration sequence comprising almost-satisfying truth assignments.

In this study, we prove a series of *gap-preserving reductions* to give evidence that a host of reconfiguration problems are **PSPACE**-hard to approximate, under some plausible assumption. Our starting point is a new working hypothesis called the *Reconfiguration Inapproximability Hypothesis* (RIH), which asserts that a gap version of Maxmin CSP Reconfiguration is **PSPACE**-hard. This hypothesis may be thought of as a reconfiguration analogue of the PCP theorem [3,4]. Our main result is **PSPACE**-hardness of approximating Maxmin 3-SAT Reconfiguration of *bounded occurrence* under RIH. The crux of its proof is a gap-preserving reduction from Maxmin Binary CSP Reconfiguration to itself of *bounded degree*. Because a simple application of the degree reduction technique using expander graphs due to Papadimitriou and Yannakakis (J. Comput. Syst. Sci., 1991) [40] does not preserve the *perfect completeness*, we modify the alphabet as if each vertex could take a pair of values simultaneously. To accomplish the soundness requirement, we further apply an explicit family of near-Ramanujan graphs and the expander mixing lemma. As an application of the main result, we demonstrate that under RIH, optimization variants of popular reconfiguration problems are **PSPACE**-hard to approximate, including Nondeterministic Constraint Logic due to Hearn and Demaine (Theor. Comput. Sci., 2005) [24,25], Independent Set Reconfiguration, Clique Reconfiguration, and Vertex Cover Reconfiguration.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases combinatorial reconfiguration, hardness of approximation, gap reduction

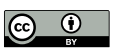
Digital Object Identifier 10.4230/LIPIcs.STACS.2023.49

Related Version *Full Version*: `https://arxiv.org/abs/2212.04207` [38]

Acknowledgements I wish to thank the anonymous referees for their suggestions which help improve the presentation of this paper.

1 Introduction

Combinatorial reconfiguration is a growing research field studying the following problem over the solution space: Given a pair of feasible solutions for a particular search problem, find a step-by-step transformation from one to the other, called a *reconfiguration sequence*. Since the establishment of the unified framework of reconfiguration due to Ito, Demaine, Harvey, Papadimitriou, Sideri, Uehara, and Uno [28], numerous reconfiguration problems have been derived from search problems; e.g., in SAT Reconfiguration [23], for a Boolean formula φ and two satisfying truth assignments σ_{s} and σ_{t} for φ, we seek a reconfiguration


© Naoto Ohsaka;
licensed under Creative Commons License CC-BY 4.0
40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 49; pp. 49:1–49:18


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

sequence of satisfying truth assignments from σ_{s} to σ_{t}, each resulting from the previous one by flipping a single variable assignment. Of particular importance is to reveal their computational complexity. Most reconfiguration problems are classified as either **P** (e.g., 3-Coloring Reconfiguration [14]), **NP**-complete, or **PSPACE**-complete (e.g., Independent Set Reconfiguration [24]), and recent studies dig into the fine-grained analysis using restricted graph classes and parameterized complexity [20, 21]. See surveys by van den Heuvel [41] and Nishimura [37]. One promising aspect has, however, been still less explored: *approximability*.

Just like an **NP** optimization problem derived from an **NP** search problem (e.g., Max SAT is a generalization of SAT), an *optimization variant* can be defined for a reconfiguration problem. For instance, in Maxmin SAT Reconfiguration [28] – an optimization variant of SAT Reconfiguration – we wish to maximize the minimum fraction of clauses of φ satisfied by any truth assignment during transformation from σ_{s} to σ_{t}. Such optimization variants naturally arise when we are faced with the nonexistence of a reconfiguration sequence for the original decision version, or when we already know a problem of interest to be **PSPACE**-complete. Solving them approximately, we may be able to obtain a reasonable reconfiguration sequence, e.g., that comprising *almost-satisfying* truth assignments, each violating at most 1% of the clauses.

Indeed, in their seminal work, Ito et al. [28] proved inapproximability results of Maxmin SAT Reconfiguration and Maxmin Clique Reconfiguration, and posed **PSPACE**-hardness of approximation as an open problem. Their results rely on **NP**-hardness of the corresponding search problem, which, however, does not bring us **PSPACE**-hardness. The significance of showing **PSPACE**-hardness is that it not only refutes a polynomial-time algorithm under $\mathbf{P} \neq \mathbf{PSPACE}$, but further disproves the existence of a witness (especially a reconfiguration sequence) of *polynomial length* under $\mathbf{NP} \neq \mathbf{PSPACE}$. The present study aims to reboot the study on **PSPACE**-hardness of approximation for reconfiguration problems, assuming some plausible hypothesis.

Our Approach. Since no **PSPACE**-hardness of approximation for natural reconfiguration problems are known (to the best of our knowledge), we assert a new working hypothesis called the *Reconfiguration Inapproximability Hypothesis* (RIH), concerning a gap version of Maxmin q-CSP Reconfiguration, and use it as a starting point.

▶ **Hypothesis 1.1** (informal; see Hypothesis 2.4)**.** *Given a constraint graph G and two satisfying assignments ψ_{s} and ψ_{t} for G, it is* **PSPACE***-hard to distinguish between* YES *instances, in which ψ_{s} can be transformed into ψ_{t} by repeatedly changing the value of a single vertex at a time, while ensuring every intermediate assignment satisfying G, and* NO *instances, in which any such transformation induces an assignment violating ε-fraction of the constraints.*

This hypothesis may be thought of as a reconfiguration analogue of the PCP theorem [3, 4], and it already holds as long as "**PSPACE**-hard" is replaced by "**NP**-hard" [28]. Moreover, if the gap version of some reconfiguration problem, e.g., Maxmin SAT Reconfiguration, is **PSPACE**-hard, RIH directly follows. Our contribution is to prove that the converse is also true: Starting from RIH, we present a series of (polynomial-time) *gap-preserving reductions* to give evidence that a host of reconfiguration problems are **PSPACE**-hard to approximate.

Our Results. Figure 1 presents an overall picture of the gap-preserving reductions introduced in this paper, all of which preserve the *perfect completeness*; i.e., YES instances have a solution to the decision version. Our main result is **PSPACE**-hardness of approximating Maxmin E3-SAT Reconfiguration of *bounded occurrence* under RIH (Theorem 3.1). Here, "bounded

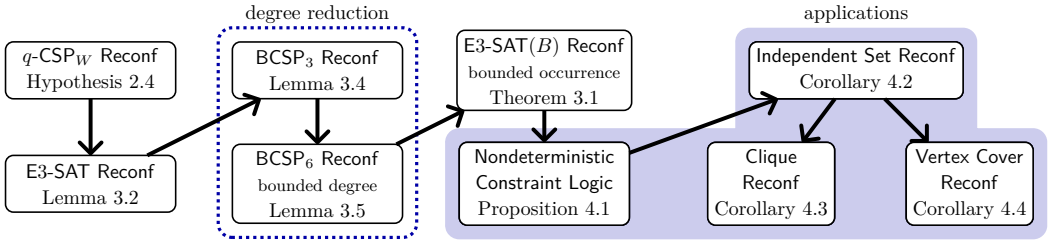


Figure 1 A series of gap-preserving reductions starting from Reconfiguration Inapproximability Hypothesis used in this paper. Here, q-CSP$_W$ Reconf and BCSP$_W$ Reconf denote q-CSP Reconfiguration and Binary CSP Reconfiguration whose alphabet size is restricted to W, respectively; E3-SAT(B) Reconf denotes 3-SAT Reconfiguration in which every clause has exactly 3 literals and each variable occurs in at most B clauses. Note that all reductions preserve the perfect completeness.

occurrence" is critical to further reduce to Nondeterministic Constraint Logic, which requires the number of clauses to be proportional to the number of variables. Toward that end, we first reduce from Maxmin q-CSP Reconfiguration to Maxmin Binary CSP Reconfiguration in a gap-preserving manner *via* Maxmin E3-SAT Reconfiguration (Lemmas 3.2 and 3.4), which employs a reconfigurable SAT encoding.

We then proceed to a gap-preserving reduction from Maxmin Binary CSP Reconfiguration to itself of *bounded degree* (Lemma 3.5), which is the most technical step in this paper. Recall shortly the degree reduction technique due to Papadimitriou and Yannakakis [40], also used by Dinur [19] to prove the PCP theorem [3, 4]: Each (high-degree) vertex is replaced by an expander graph called a cloud, and equality constraints are imposed on the intra-cloud edges so that the assignments in the cloud behave like a single assignment. Observe easily that a simple application of this technique to Binary CSP Reconfiguration fails to preserve the perfect completeness. This is because we have to change the value of the vertices in the cloud *one by one*, breaking many equality constraints. To bypass this issue, we modify the alphabet as if each vertex could take a pair of values simultaneously; e.g., if the original alphabet is $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, the new one is $\Sigma' = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{ab}, \mathsf{bc}, \mathsf{ca}\}$. Having a vertex to be assigned ab represents that it has value a *and* b. With this interpretation in mind, we redefine equality-like constraints for the intra-cloud edges so as to preserve the perfect completeness.

Unfortunately, this modification causes another issue, which renders the proof of soundness nontrivial. Example 3.9 illustrated in Figure 2 tells us that our reduction is neither a Karp reduction of Binary CSP Reconfiguration nor a PTAS reduction [16, 17] of Maxmin Binary CSP Reconfiguration. One particular reason is that assigning conflicting values to vertices in a cloud may not break any equality-like constraints. Thankfully, we are "promised" that at least ε-fraction of constraints are violated for some $\varepsilon \in (0, 1)$. We therefore use the following machinery to accomplish the soundness requirement:

- Use an explicit family of near-Ramanujan graphs [1, 36] so that the second largest eigenvalue λ is $\mathcal{O}(\sqrt{d})$; the degree d is determined based on the value of ε.
- Apply the *expander mixing lemma* [2] to bound the number of violated edges, whereas Papadimitriou and Yannakakis [40] used the vertex expansion property, which is not applicable as shown in Example 3.9.

By applying this degree reduction step, we come back to Maxmin E3-SAT Reconfiguration, wherein, but this time, each variable appears in a constant number of clauses, completing the proof of the main result.

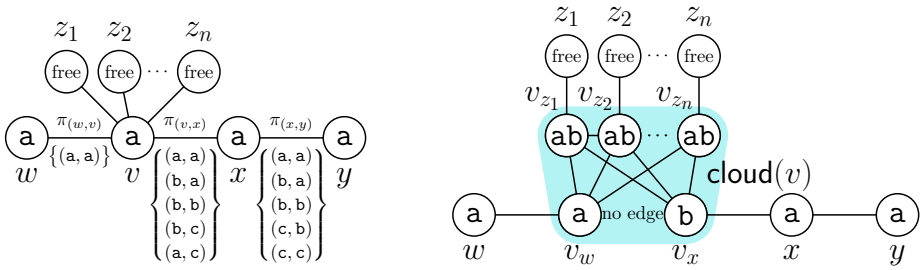


Figure 2 A drawing of Example 3.9. The left side shows an instance G of BCSP Reconfiguration, where we cannot transform from $\psi_{\mathsf{s}}(w, v, x, y) = (\mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a})$ to $\psi_{\mathsf{t}}(w, v, x, y) = (\mathsf{a}, \mathsf{a}, \mathsf{c}, \mathsf{c})$. The right side shows the resulting instance by applying the degree reduction step on v of G. We can now assign conflicting values to v_w and v_x because edge (v_w, v_x) does not exist; in particular, we can transform from $\psi'_{\mathsf{s}}(w, v_w, v_x, x, y) = (\mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{a})$ into $\psi'_{\mathsf{t}}(w, v_w, v_x, x, y) = (\mathsf{a}, \mathsf{a}, \mathsf{a}, \mathsf{c}, \mathsf{c})$.

Once we have established gap-preserving reducibility from RIH to Maxmin E3-SAT Reconfiguration of bounded occurrence, we can apply it to devise conditional **PSPACE**-hardness of approximation for an optimization variant of Nondeterministic Constraint Logic (Proposition 4.1). Nondeterministic Constraint Logic is a **PSPACE**-complete problem proposed by Hearn and Demaine [24, 25] that has been used to prove **PSPACE**-hardness of many games, puzzles, and other reconfiguration problems [5, 9, 11, 30]. We show that under RIH, it is **PSPACE**-hard to distinguish whether an input is a YES instance, or has a property that every transformation must violate more than ε-fraction of nodes. The proof makes a modification to the existing gadgets [24, 25]. As a consequence of Proposition 4.1, we demonstrate that assuming RIH, optimization variants of popular reconfiguration problems on a graph are **PSPACE**-hard to approximate, including Independent Set Reconfiguration, Clique Reconfiguration, and Vertex Cover Reconfiguration (Corollaries 4.2–4.4), whose proofs are almost immediate from existing work [9, 24, 25].

Owing to space limitations, proofs marked with $*$ are omitted and can be found in the full version of this paper [38].

Additional Related Work. Other reconfiguration problems whose approximability was analyzed include Set Cover Reconfiguration [28], which is 2-factor approximable, Subset Sum Reconfiguration [27], which admits a PTAS, Shortest Path Reconfiguration [22], and Submodular Reconfiguration [39]. We note that approximability of reconfiguration problems frequently refers to that of *the shortest sequence* [7, 8, 10, 12, 13, 26, 29, 35, 42], which is of independent interest. The objective value of optimization variants is sometimes called the *reconfiguration index* [32] or *reconfiguration threshold* [18]. A different type of optimization variant, called *incremental optimization under the reconfiguration framework* [6, 31, 43] has recently been studied; e.g., starting from an initial independent set, we want to transform into a maximum possible independent set without touching those smaller than the specified size.

2 Preliminaries

Notations. For two integers $m, n \in \mathbb{N}$ with $m \leqslant n$, let $[n] \triangleq \{1, 2, \ldots, n\}$ and $[m \,..\, n] \triangleq \{m, m+1, \ldots, n-1, n\}$. A *sequence* $\mathcal{E}$ of a finite number of elements $e^{(0)}, e^{(1)}, \ldots, e^{(\ell)}$ is denoted by $\langle e^{(0)}, e^{(1)}, \ldots, e^{(\ell)} \rangle$, and we write $e^{(i)} \in \mathcal{E}$ to indicate that $e^{(i)}$ appears in $\mathcal{E}$. We briefly recapitulate Ito et al.'s reconfiguration framework [28]. Suppose we are given a "definition" of feasible solutions for some combinatorial search problem and a symmetric

"adjacency relation" over a pair of feasible solutions.[1] For two feasible solutions e_{s} and e_{t}, a *reconfiguration sequence from* e_{s} *to* e_{t} is a sequence of feasible solutions, $\mathcal{E} = \langle e^{(0)}, \dots, e^{(\ell)} \rangle$, starting from e_{s} (i.e., $e^{(0)} = e_{\mathsf{s}}$) and ending with e_{t} (i.e., $e^{(\ell)} = e_{\mathsf{t}}$) such that all consecutive solutions $e^{(i-1)}$ and $e^{(i)}$ are adjacent. In a reconfiguration problem, we wish to decide if there exists a reconfiguration sequence between a pair of feasible solutions.

Boolean Satisfiability. We use the standard terminology and notation of Boolean satisfiability. Truth values are denoted by `T` or `F`. A *Boolean formula* φ consists of variables $x_1, \dots, x_n$ and the logical operators, AND ($\wedge$), OR ($\vee$), and NOT ($\neg$). A *truth assignment* $\sigma \colon \{x_1, \dots, x_n\} \to \{\mathtt{T}, \mathtt{F}\}$ for φ is a mapping that assigns a truth value to each variable. A Boolean formula φ is said to be *satisfiable* if there exists some assignment σ such that φ evaluates to `T` when each variable x_i is assigned the truth value specified by $\sigma(x_i)$. A *literal* is either a variable or its negation; a *clause* is a disjunction of literals. A Boolean formula is said to be in *conjunctive normal form* (CNF) if it is a conjunction of clauses. A *k-CNF* formula is a CNF formula in which every clause contains at most k literals. Hereafter, the prefix "Ek-" means that every clause has exactly k distinct literals, while the suffix "(B)" indicates that the number of occurrences of each variable is bounded by $B \in \mathbb{N}$. We say that two truth assignments for a Boolean formula are *adjacent* if one is obtained from the other by flipping a single variable assignment; i.e., they differ in exactly one variable. In the **PSPACE**-complete k-SAT Reconfiguration problem [23], for a k-CNF formula φ and two satisfying truth assignments σ_{s} and σ_{t} for φ, we are asked to decide if there exists a reconfiguration sequence of satisfying truth assignments for φ from σ_{s} to σ_{t}. Since we are concerned with approximability of k-SAT Reconfiguration, we formulate its optimization variant [28], where we are allowed to go through *non-satisfying* truth assignments. For a CNF formula φ consisting of m clauses $C_1, \dots, C_m$ and a truth assignment σ for φ, let $\mathsf{val}_\varphi(\sigma)$ denote the fraction of clauses in φ satisfied by σ; namely,

$$\mathsf{val}_\varphi(\sigma) \triangleq \frac{|\{j \in [m] : \sigma \text{ satisfies } C_j\}|}{m}. \tag{1}$$

For a reconfiguration sequence $\boldsymbol{\sigma} = \langle \sigma^{(0)}, \dots, \sigma^{(\ell)} \rangle$ of truth assignments, let $\mathsf{val}_\varphi(\boldsymbol{\sigma})$ denote the *minimum* fraction of satisfied clauses over all $\sigma^{(i)}$'s in $\boldsymbol{\sigma}$; i.e.,

$$\mathsf{val}_\varphi(\boldsymbol{\sigma}) \triangleq \min_{\sigma^{(i)} \in \boldsymbol{\sigma}} \mathsf{val}_\varphi(\sigma^{(i)}). \tag{2}$$

Then, Maxmin k-SAT Reconfiguration is defined as a problem of maximizing $\mathsf{val}_\varphi(\boldsymbol{\sigma})$ subject to $\boldsymbol{\sigma} = \langle \sigma_{\mathsf{s}}, \dots, \sigma_{\mathsf{t}} \rangle$, where σ_{s} and σ_{t} are not necessarily satisfying. For two truth assignments σ_{s} and σ_{t} for φ, let $\mathsf{val}_\varphi(\sigma_{\mathsf{s}} \rightsquigarrow \sigma_{\mathsf{t}})$ denote the maximum value of $\mathsf{val}_\varphi(\boldsymbol{\sigma})$ over all possible reconfiguration sequences $\boldsymbol{\sigma}$ from σ_{s} to σ_{t}; namely,

$$\mathsf{val}_\varphi(\sigma_{\mathsf{s}} \rightsquigarrow \sigma_{\mathsf{t}}) \triangleq \max_{\boldsymbol{\sigma} = \langle \sigma_{\mathsf{s}}, \dots, \sigma_{\mathsf{t}} \rangle} \mathsf{val}_\varphi(\boldsymbol{\sigma}) = \max_{\boldsymbol{\sigma} = \langle \sigma_{\mathsf{s}}, \dots, \sigma_{\mathsf{t}} \rangle} \min_{\sigma^{(i)} \in \boldsymbol{\sigma}} \mathsf{val}_\varphi(\sigma^{(i)}). \tag{3}$$

Note that $\mathsf{val}_\varphi(\sigma_{\mathsf{s}} \rightsquigarrow \sigma_{\mathsf{t}}) \leqslant \min\{\mathsf{val}_\varphi(\sigma_{\mathsf{s}}), \mathsf{val}_\varphi(\sigma_{\mathsf{t}})\}$. If $\mathsf{val}_\varphi(\sigma_{\mathsf{s}} \rightsquigarrow \sigma_{\mathsf{t}}) \geqslant \rho$ for some ρ, we can transform σ_{s} into σ_{t} while ensuring that *every* intermediate truth assignment satisfies at least ρ-fraction of the clauses of φ. We finally define the "gap version" of Maxmin k-SAT Reconfiguration as follows.

[1] An adjacency relation can also be defined in terms of a "reconfiguration step," which specifies how a solution can be transformed, e.g., a flip of a single variable assignment.

▶ **Problem 2.1.** *For every $k \in \mathbb{N}$ and $0 \leqslant s \leqslant c \leqslant 1$,* $\mathsf{Gap}_{c,s}$ k-SAT Reconfiguration *requests to determine for a k-CNF formula φ and two truth assignments σ_{s} and σ_{t} for φ, whether $\mathsf{val}_{\varphi}(\sigma_{\mathsf{s}} \rightsquigarrow \sigma_{\mathsf{t}}) \geqslant c$ (the input is a* YES *instance) or $\mathsf{val}_{\varphi}(\sigma_{\mathsf{s}} \rightsquigarrow \sigma_{\mathsf{t}}) < s$ (the input is a* NO *instance). Here, c and s denote completeness and soundness, respectively.*

Problem 2.1 is a *promise problem*, in which we are allowed to output anything when $s \leqslant \mathsf{val}_{\varphi}(\sigma_{\mathsf{s}} \rightsquigarrow \sigma_{\mathsf{t}}) < c$. The present problem definition does not request an actual reconfiguration sequence. Note that we can assume σ_{s} and σ_{t} to be satisfying ones whenever $c = 1$, and the case of $s = c = 1$ particularly reduces to k-SAT Reconfiguration.

Constraint Satisfaction Problem. Subsequently, we introduce reconfiguration problems on constraint satisfaction. First, we define the notion of *constraint graphs*.

▶ **Definition 2.2** (Constraint graph)**.** *A q-ary constraint graph is defined as a tuple $G = (V, E, \Sigma, \Pi)$ such that (V, E) is a q-uniform hypergraph called the underlying graph of G, Σ is a finite set called the alphabet, and $\Pi = (\pi_e)_{e \in E}$ is a collection of q-ary constraints, where each $\pi_e \subseteq \Sigma^e$ is a set of q-tuples of acceptable values that q vertices in e can take. The degree $d_G(v)$ of each vertex v in G is defined as the number of hyperedges including v.*

An *assignment* is a mapping $\psi \colon V \to \Sigma$ that assigns a value of Σ to each vertex of V. We say that ψ *satisfies* hyperedge $e = \{v_1, \ldots, v_q\} \in E$ (or constraint π_e) if $\psi(e) \triangleq (\psi(v_1), \ldots, \psi(v_q)) \in \pi_e$, and ψ *satisfies* G if it satisfies all hyperedges of G. We say that G is *satisfiable* if some assignment that satisfies G exists. Two assignments are said to be *adjacent* if they differ in exactly one vertex. In q-CSP Reconfiguration, for a q-ary constraint graph G and two satisfying assignments ψ_{s} and ψ_{t} for G, we are asked to decide if there is a reconfiguration sequence of satisfying assignments for G from ψ_{s} to ψ_{t}. Hereafter, BCSP stands for 2-CSP, q-CSP_W designates the restricted case that the alphabet size $|\Sigma|$ is some $W \in \mathbb{N}$, and q-$\mathsf{CSP}(\Delta)$ for some $\Delta \in \mathbb{N}$ means that the maximum degree of the constraint graph is bounded by Δ. In an analogous way to the case of Boolean satisfiability, we define $\mathsf{val}_G(\psi) \triangleq \frac{|\{e \in E : \psi \text{ satisfies } e\}|}{|E|}$ for assignment $\psi \colon V \to \Sigma$, $\mathsf{val}_G(\boldsymbol{\psi}) \triangleq \min_{\psi^{(i)} \in \boldsymbol{\psi}} \mathsf{val}_G(\psi^{(i)})$ for reconfiguration sequence $\boldsymbol{\psi} = \langle \psi^{(i)} \rangle_{i \in [0..\ell]}$, and $\mathsf{val}_G(\psi_{\mathsf{s}} \rightsquigarrow \psi_{\mathsf{t}}) \triangleq \max_{\boldsymbol{\psi} = \langle \psi_{\mathsf{s}}, \ldots, \psi_{\mathsf{t}} \rangle} \mathsf{val}_G(\boldsymbol{\psi})$ for $\psi_{\mathsf{s}}, \psi_{\mathsf{t}} \colon V \to \Sigma$. In Maxmin q-CSP Reconfiguration, we wish to maximize $\mathsf{val}_G(\boldsymbol{\psi})$ subject to $\boldsymbol{\psi} = \langle \psi_{\mathsf{s}}, \ldots, \psi_{\mathsf{t}} \rangle$. The corresponding gap version is defined as follows.

▶ **Problem 2.3.** *For every $q \in \mathbb{N}$ and $0 \leqslant s \leqslant c \leqslant 1$,* $\mathsf{Gap}_{c,s}$ q-CSP Reconfiguration *requests to determine for a q-ary constraint graph G and two (not necessarily satisfying) assignments ψ_{s} and ψ_{t} for G, whether $\mathsf{val}_G(\psi_{\mathsf{s}} \rightsquigarrow \psi_{\mathsf{t}}) \geqslant c$ or $\mathsf{val}_G(\psi_{\mathsf{s}} \rightsquigarrow \psi_{\mathsf{t}}) < s$.*

Reconfiguration Inapproximability Hypothesis. We now present a formal description of our working hypothesis, which serves as a starting point for **PSPACE**-hardness of approximation.

▶ **Hypothesis 2.4** (Reconfiguration Inapproximability Hypothesis, RIH)**.** *There exist universal constants $q, W \in \mathbb{N}$, $\varepsilon \in (0, 1)$ such that* $\mathsf{Gap}_{1,1-\varepsilon}$ q-CSP_W Reconfiguration *is* **PSPACE***-hard.*

3 Hardness of Approximation for Maxmin E3-SAT(B) Reconfiguration

In this section, we prove the main result of this paper; that is, Maxmin E3-SAT Reconfiguration of bounded occurrence is **PSPACE**-hard to approximate under RIH.

▶ **Theorem 3.1.** *Under Hypothesis 2.4, there exist universal constants $\varepsilon \in (0,1)$ and $B \in \mathbb{N}$ such that* $\mathsf{Gap}_{1,1-\varepsilon}$ E3-SAT(B) Reconfiguration *is* **PSPACE***-hard.*

The remainder of this section is devoted to the proof of Theorem 3.1 and organized as follows. In Section 3.1, we reduce Maxmin q-CSP$_W$ Reconfiguration to Maxmin BCSP$_3$ Reconfiguration; Section 3.2 presents the degree reduction of Maxmin BCSP Reconfiguration.

3.1 Maxmin q-CSP$_W$ Reconfiguration to Maxmin BCSP$_3$ Reconfiguration

We first reduce from Maxmin q-CSP$_W$ Reconfiguration to Maxmin E3-SAT Reconfiguration.

▶ **Lemma 3.2** (∗). *For every $q, W \geqslant 2$ and $\varepsilon \in (0,1)$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ q-CSP$_W$ Reconfiguration *to* $\mathsf{Gap}_{1,1-\frac{\varepsilon}{W^q \cdot 2^{qW}(qW-2)}}$ E3-SAT Reconfiguration. *Moreover, if the maximum degree of the constraint graph in the former problem is Δ, then the number of occurrences of each variable in the latter problem is bounded by $W^q \cdot 2^{qW}\Delta$.*

The proof of Lemma 3.2 consists of a reduction from Maxmin q-CSP$_W$ Reconfiguration to Maxmin Ek-SAT Reconfiguration, where the clause size k depends solely on q and W, and that from Maxmin Ek-SAT Reconfiguration to Maxmin E3-SAT Reconfiguration. In the first reduction, we apply a slightly sophisticated SAT encoding, described below. In the second reduction, we use an established Karp reduction from k-SAT to 3-SAT, previously used by Gopalan, Kolaitis, Maneva, and Papadimitriou [23] in the context of reconfiguration.

Reconfigurable SAT Encoding. For the proof of the first reduction, we introduce an encoding of the alphabet of a constraint graph into a string of truth values. Hereafter, we denote $\Sigma \triangleq [W]$ for some integer $W \in \mathbb{N}$. Consider an encoding $\mathsf{enc}\colon \{\mathtt{T},\mathtt{F}\}^\Sigma \to \Sigma$ of a binary string $\mathbf{s} \in \{\mathtt{T},\mathtt{F}\}^\Sigma$ to Σ defined as follows:

$$\mathsf{enc}(\mathbf{s}) \triangleq \begin{cases} 1 & \text{if } s_\alpha = \mathtt{F} \text{ for all } \alpha \in \Sigma, \\ \alpha & \text{if } s_\alpha = \mathtt{T} \text{ and } s_\beta = \mathtt{F} \text{ for all } \beta > \alpha. \end{cases} \tag{4}$$

enc exhibits the following property concerning reconfigurability:

▷ Claim 3.3 (∗). For any two strings $\mathbf{s}$ and $\mathbf{t}$ in $\{\mathtt{T},\mathtt{F}\}^\Sigma$ with $\alpha \triangleq \mathsf{enc}(\mathbf{s})$ and $\beta \triangleq \mathsf{enc}(\mathbf{t})$, we can transform $\mathbf{s}$ into $\mathbf{t}$ by repeatedly flipping one entry at a time while preserving every intermediate string mapped to α or β by enc.

We use enc to encode each q-tuple of unacceptable values $(\alpha_1,\dots,\alpha_q) \in \Sigma^e \setminus \pi_e$ for hyperedge $e = \{v_1,\dots,v_q\} \in E$.

We then reduce Maxmin E3-SAT Reconfiguration to Maxmin BCSP$_3$ Reconfiguration in a gap-preserving manner, whose proof uses the *place encoding* due to Järvisalo and Niemelä [33].

▶ **Lemma 3.4** (∗). *For every $\varepsilon \in (0,1)$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ E3-SAT Reconfiguration *to* $\mathsf{Gap}_{1,1-\frac{\varepsilon}{3}}$ BCSP$_3$ Reconfiguration. *Moreover, if the number of occurrences of each variable in the former problem is B, then the maximum degree of the constraint graph in the latter problem is bounded by* $\max\{B,3\}$.

Reduction. We here describe a reduction from Maxmin E3-SAT Reconfiguration to Maxmin BCSP_3 Reconfiguration. Let $(\varphi, \sigma_{\mathsf{s}}, \sigma_{\mathsf{t}})$ be an instance of Maxmin E3-SAT Reconfiguration, where φ is a 3-CNF formula consisting of m clauses $C_1, \ldots, C_m$ over n variables $x_1, \ldots, x_n$ and σ_{s} and σ_{t} satisfy φ. Using the place encoding due to Järvisalo and Niemelä [33], we construct a binary constraint graph $G = (V, E, \Sigma, \Pi)$ as follows. The underlying graph of G is a *bipartite graph* with a bipartition $(\{x_1, \ldots, x_n\}, \{C_1, \ldots, C_m\})$, and there is an edge between variable x_i and clause C_j in E if x_i or $\overline{x_i}$ appears in C_j. For the sake of notation, we use Σ_v to denote the alphabet assigned to vertex $v \in V$. We then express $\Sigma_{x_i} \triangleq \{\mathtt{T}, \mathtt{F}\}$ for each variable x_i, and $\Sigma_{C_j} \triangleq \{\ell_1, \ell_2, \ell_3\}$ for each clause $C_j = (\ell_1 \vee \ell_2 \vee \ell_3)$. For each edge $(x_i, C_j) \in E$ with $C_j = (\ell_1 \vee \ell_2 \vee \ell_3)$, the constraint $\pi_{(x_i, C_j)} \subseteq \Sigma_{x_i} \times \Sigma_{C_j}$ is defined as follows:

$$\pi_{(x_i, C_j)} \triangleq \begin{cases} (\Sigma_{x_i} \times \Sigma_{C_j}) \setminus \{(\mathtt{F}, x_i)\} & \text{if } x_i \text{ appears in } C_j, \\ (\Sigma_{x_i} \times \Sigma_{C_j}) \setminus \{(\mathtt{T}, \overline{x_i})\} & \text{if } \overline{x_i} \text{ appears in } C_j. \end{cases} \tag{5}$$

For an assignment ψ for G, $\psi(x_i)$ represents the truth value assigned to x_i, and $\psi(C_j)$ specifies which literal should evaluate to $\mathtt{T}$. Note that $|V| = n + m$, $|E| = 3m$, and the maximum degree of (V, E) is $\max\{B, 3\}$. For a satisfying truth assignment σ for φ, let $\psi_\sigma : V \to \Sigma$ be an assignment for G defined as follows: $\psi_\sigma(x_i) \triangleq \sigma(x_i)$ for each variable x_i, and $\psi_\sigma(C_j) \triangleq \ell_i$ for each clause C_j whenever ℓ_i appears in C_j and evaluates to $\mathtt{T}$ by σ.[2] Obviously, ψ_σ satisfies G. Constructing ψ_{s} from σ_{s} and ψ_{t} from σ_{t} according to this procedure, we obtain an instance $(G, \psi_{\mathsf{s}}, \psi_{\mathsf{t}})$ of Maxmin BCSP_3 Reconfiguration, which completes the reduction. It is not hard to see that the above reduction is a gap-preserving reduction, whose proof can be found in the full version [38].

3.2 Degree Reduction of Maxmin BCSP Reconfiguration

We now present a gap-preserving reduction from Maxmin BCSP Reconfiguration to itself of *bounded degree*, which is the most technical step in this paper.

▶ **Lemma 3.5.** *For every $\varepsilon \in (0, 1)$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ BCSP_3 Reconfiguration *to* $\mathsf{Gap}_{1,1-\overline{\varepsilon}}$ $\mathsf{BCSP}_6(\Delta)$ Reconfiguration*, where $\overline{\varepsilon} \in (0, 1)$ and $\Delta \in \mathbb{N}$ are some computable functions dependent only on the value of ε. In particular, the constraint graph in the latter problem is of bounded degree for fixed ε.*

We first introduce the notion of *expander graphs*.

▶ **Definition 3.6** (Expander graph)**.** *For every $n \in \mathbb{N}$, $d \in \mathbb{N}$, and $\lambda > 0$, an (n, d, λ)-expander graph is a d-regular graph G on n vertices such that $\max\{\lambda_2(G), |\lambda_n(G)|\} \leqslant \lambda < d$, where $\lambda_i(G)$ is the i-th largest (real-valued) eigenvalue of the adjacency matrix of G.*

An (n, d, λ)-expander graph is called *Ramanujan* if $\lambda \leqslant 2\sqrt{d-1}$. There exists an *explicit construction* (i.e., a polynomial-time algorithm) for near-Ramanujan graphs.

▶ **Theorem 3.7** (Explicit construction of near-Ramanujan graphs [1, 36])**.** *For every constant $d \geqslant 3$, $\varepsilon > 0$, and all sufficiently large $n \geqslant n_0(d, \varepsilon)$, where nd is even, there is a deterministic $n^{\mathcal{O}(1)}$-time algorithm that outputs an (n, d, λ)-expander graph with $\lambda \leqslant 2\sqrt{d-1} + \varepsilon$.*

In this paper, we rely only on the special case of $\varepsilon = 2\sqrt{d} - 2\sqrt{d-1}$ so that $\lambda \leqslant 2\sqrt{d}$; thus, we let $n_0(d) \triangleq n_0(d, 2\sqrt{d} - 2\sqrt{d-1})$. We can assume $n_0(\cdot)$ to be computable as $2\sqrt{d} - 2\sqrt{d-1} \geqslant \frac{1}{\sqrt{d}}$. The crucial property of expander graphs that we use in the proof of Lemma 3.5 is the following expander mixing lemma [2].

[2] Such ℓ_i always exists as σ satisfies C_j, and ties are broken arbitrarily.

▶ **Lemma 3.8** (Expander mixing lemma; e.g., Alon and Chung [2]). *Let G be an (n, d, λ)-expander graph. Then, for any two sets S, T of vertices, it holds that*

$$\left| e(S,T) - \frac{d|S| \cdot |T|}{n} \right| \leqslant \lambda \sqrt{|S| \cdot |T|}, \tag{6}$$

where $e(S,T)$ counts the number of edges between S and T.

This lemma states that $e(S,T)$ of an expander graph G is concentrated around its expectation if G were a *random* d-regular graph. The use of near-Ramanujan graphs enables us to make an additive error (i.e., $\lambda\sqrt{|S| \cdot |T|}$) acceptably small.

Reduction. We then explain our gap-preserving reduction, which *does* depend on ε. Redefine $\varepsilon \leftarrow \lceil 1/\varepsilon \rceil^{-1}$ so that $1/\varepsilon$ is a positive integer, which does not increase the value of ε; i.e., $\mathsf{val}_G(\psi_\mathsf{s} \rightsquigarrow \psi_\mathsf{t}) < 1 - \varepsilon$ implies $\mathsf{val}_G(\psi_\mathsf{s} \rightsquigarrow \psi_\mathsf{t}) < 1 - \lceil 1/\varepsilon \rceil^{-1}$. Let $(G = (V, E, \Sigma, \Pi = (\pi_e)_{e \in E}), \psi_\mathsf{s}, \psi_\mathsf{t})$ be an instance of $\mathsf{Gap}_{1,1-\varepsilon}\ \mathsf{BCSP}_3\ \mathsf{Reconfiguration}$, where ψ_s and ψ_t satisfy G. For the sake of notation, we denote $\Sigma \triangleq \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. We then create a new instance $(G' = (V', E', \Sigma', \Pi' = (\pi'_{e'})_{e' \in E'}), \psi'_\mathsf{s}, \psi'_\mathsf{t})$ of $\mathsf{Maxmin}\ \mathsf{BCSP}_6\ \mathsf{Reconfiguration}$, which turns out to meet the requirement of completeness and soundness. The constraint graph G' is defined as follows:

Vertex set: For each vertex v of V, let

$$\mathsf{cloud}(v) \triangleq \Big\{ (v, e) : e \in E \text{ is incident on } v \Big\}. \tag{7}$$

Define $V' \triangleq \bigcup_{v \in V} \mathsf{cloud}(v)$.

Edge set: For each vertex v of V, let X_v be a $(d_G(v), d_0, \lambda)$-expander graph on $\mathsf{cloud}(v)$ using Theorem 3.7 if $d_G(v) \geqslant n_0(d_0)$, or a complete graph on $\mathsf{cloud}(v)$ if $d_G(v) < n_0(d_0)$. Here, $\lambda \leqslant 2\sqrt{d_0}$ and $d_0 = \Theta(\varepsilon^{-2})$, whose precise value will be determined later. Define

$$E' \triangleq \bigcup_{v \in V} E(X_v) \cup \Big\{ ((v, e), (w, e)) : e = (v, w) \in E \Big\}. \tag{8}$$

Alphabet: Define $\Sigma' \triangleq \{\{\mathtt{a}\}, \{\mathtt{b}\}, \{\mathtt{c}\}, \{\mathtt{a}, \mathtt{b}\}, \{\mathtt{b}, \mathtt{c}\}, \{\mathtt{c}, \mathtt{a}\}\}$. By abuse of notation, we write each value of Σ' as if it were an element (e.g., $\mathtt{ab} \in \Sigma'$, $\mathtt{a} \subset \mathtt{ab}$, and $\mathtt{b} \not\subseteq \mathtt{ca}$).

Constraints: The constraint $\pi'_{e'} \subseteq \Sigma'^{e'}$ for each edge $e' \in E'$ is defined as follows:

- If $e' \in E(X_v)$ for some $v \in V$ (i.e., e' is an intra-cloud edge), define[3]

$$\pi'_{e'} \triangleq \Big\{ (\alpha, \beta) : \alpha, \beta \in \Sigma', \alpha \subseteq \beta \text{ or } \beta \subseteq \alpha \Big\}. \tag{9}$$

- If $e' = ((v, e), (w, e))$ such that $e = (v, w) \in E$ (i.e., e' is an inter-cloud edge), define

$$\pi'_{e'} \triangleq \Big\{ (\alpha, \beta) : \alpha \times \beta \subseteq \pi_e \Big\}. \tag{10}$$

Although the underlying graph (V', E') is the same as that in [19] (except for the use of Theorem 3.7), the definitions of Σ' and Π' are somewhat different, which is essential to ensure the perfect completeness, while making the proof of the soundness nontrivial. Intuitively, having vertex $v' \in V'$ be $\psi(v') = \mathtt{ab}$ represents that v' has values $\mathtt{a}$ *and* $\mathtt{b}$ simultaneously;

[3] Eq. (9) can be explicitly expanded as $\pi'_{e'} = \{(\mathtt{a}, \mathtt{a}), (\mathtt{b}, \mathtt{b}), (\mathtt{c}, \mathtt{c}), (\mathtt{ab}, \mathtt{a}), (\mathtt{ab}, \mathtt{b}), (\mathtt{bc}, \mathtt{b}), (\mathtt{bc}, \mathtt{c}), (\mathtt{ca}, \mathtt{c}), (\mathtt{ca}, \mathtt{a}), (\mathtt{a}, \mathtt{ab}), (\mathtt{b}, \mathtt{ab}), (\mathtt{b}, \mathtt{bc}), (\mathtt{c}, \mathtt{bc}), (\mathtt{c}, \mathtt{ca}), (\mathtt{a}, \mathtt{ca}), (\mathtt{ab}, \mathtt{ab}), (\mathtt{bc}, \mathtt{bc}), (\mathtt{ca}, \mathtt{ca})\}$.

e.g., if $\psi'(v') = \mathsf{ab}$ and $\psi'(w') = \mathsf{c}$ for some $v' \in \mathsf{cloud}(v)$ and $w' \in \mathsf{cloud}(w)$ with $v \neq w$, then ψ' satisfies $\pi'_{(v',w')}$ if both (a,b) and (a,c) are in $\pi_{(v,w)}$ owing to Eq. (10). Construct two assignments $\psi'_{\mathsf{s}}\colon V' \to \Sigma'$ from ψ_{s} and $\psi'_{\mathsf{t}}\colon V' \to \Sigma'$ from ψ_{t} such that $\psi'_{\mathsf{s}}(v,e) \triangleq \{\psi_{\mathsf{s}}(v)\}$ and $\psi'_{\mathsf{t}}(v,e) \triangleq \{\psi_{\mathsf{t}}(v)\}$ for all $(v,e) \in V'$. Observe that both ψ'_{s} and ψ'_{t} satisfy G', thereby completing the reduction. Note that $|V'| = 2|E|$, $|E'| \leqslant n_0(d_0) \cdot |E|$, $|\Sigma'| = 6$, and the maximum degree of G' is $\Delta \leqslant n_0(d_0)$, which is constant for fixed ε.

Using an example illustrated in Figure 2, we demonstrate that our reduction may map a NO instance of BCSP Reconfiguration to a YES instance; namely, it is neither a Karp reduction of BCSP Reconfiguration nor a PTAS reduction of Maxmin BCSP Reconfiguration.

▶ **Example 3.9.** We construct a constraint graph $G = (V, E, \Sigma, \Pi = (\pi_e)_{e\in E})$ such that $V \triangleq \{w, v, x, y, z_1, \ldots, z_n\}$ for some large integer n, $E \triangleq \{(w,v),(v,x),(x,y),(v,z_1),\ldots,(v,z_n)\}$, $\Sigma \triangleq \{\mathsf{a},\mathsf{b},\mathsf{c}\}$, and each π_e is defined as follows: $\pi_{(w,v)} \triangleq \{(\mathsf{a},\mathsf{a})\}$, $\pi_{(v,x)} \triangleq \{(\mathsf{a},\mathsf{a}),(\mathsf{b},\mathsf{a}),(\mathsf{b},\mathsf{b}),(\mathsf{b},\mathsf{c}),(\mathsf{a},\mathsf{c})\}$, $\pi_{(x,y)} \triangleq \{(\mathsf{a},\mathsf{a}),(\mathsf{b},\mathsf{a}),(\mathsf{b},\mathsf{b}),(\mathsf{c},\mathsf{b}),(\mathsf{c},\mathsf{c})\}$, and $\pi_{(v,z_1)} = \cdots = \pi_{(v,z_n)} \triangleq \Sigma \times \Sigma$. Define $\psi_{\mathsf{s}}, \psi_{\mathsf{t}}\colon V \to \Sigma$ such that $\psi_{\mathsf{s}}(u) \triangleq \mathsf{a}$ for all $u \in V$, $\psi_{\mathsf{t}}(x) = \psi_{\mathsf{t}}(y) \triangleq \mathsf{c}$, and $\psi_{\mathsf{t}}(u) \triangleq \mathsf{a}$ for all other u. Then, it is *impossible* to transform ψ_{s} into ψ_{t} without any constraint violation: As the values of w and v cannot change from a, we can only change the value of x to c, violating (x,y). In particular, $\mathsf{val}_G(\psi_{\mathsf{s}} \rightsquigarrow \psi_{\mathsf{t}}) < 1$.

Consider applying our reduction to v *only*. Create $\mathsf{cloud}(v) \triangleq \{v_w, v_x, v_{z_1}, \ldots, v_{z_n}\}$ with the shorthand notation $v_u \triangleq (v,(v,u))$, and let X_v be an expander graph on $\mathsf{cloud}(v)$. We then construct a new constraint graph $G' = (V', E', \Sigma', \Pi' = (\pi'_e)_{e\in E'})$, where $V' \triangleq \{w, x, y, z_1, \ldots, z_n\} \cup \mathsf{cloud}(v)$, $E' \triangleq E(X_v) \cup \{(w,v_w),(v_x,x),(x,y),(v_{z_1},z_1),\ldots,(v_{z_n},z_n)\}$, $\Sigma' \triangleq \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{ab},\mathsf{bc},\mathsf{ca}\}$, and each π'_e is defined according to Eqs. (9) and (10). Construct $\psi'_{\mathsf{s}}, \psi'_{\mathsf{t}}\colon V' \to \Sigma'$ from $\psi_{\mathsf{s}}, \psi_{\mathsf{t}}$ using the procedure described above. Suppose now "by chance" $(v_w, v_x) \notin E(X_v)$. The crucial observation is that we can assign a to v_w, b to v_x, and ab to $v_{z_1}, \ldots, v_{z_n}$ to do some "cheating." Consequently, ψ'_{s} can be transformed into ψ'_{t} without sacrificing any constraint: Assign ab to $v_{z_1}, \ldots, v_{z_n}$ in arbitrary order; assign b to v_x, x, and y in this order; assign c to x and y in this order; assign a to v_x; assign a to $v_{z_1}, \ldots, v_{z_n}$ in arbitrary order. In particular, $\mathsf{val}_{G'}(\psi'_{\mathsf{s}} \rightsquigarrow \psi'_{\mathsf{t}}) = 1$.

Correctness. The proof of the completeness is immediate from the definition of Σ' and Π'.

▶ **Lemma 3.10** (∗). *If* $\mathsf{val}_G(\psi_{\mathsf{s}} \rightsquigarrow \psi_{\mathsf{t}}) = 1$, *then* $\mathsf{val}_{G'}(\psi'_{\mathsf{s}} \rightsquigarrow \psi'_{\mathsf{t}}) = 1$.

Then, in the remainder of this subsection, we prove the soundness.

▶ **Lemma 3.11.** *If* $\mathsf{val}_G(\psi_{\mathsf{s}} \rightsquigarrow \psi_{\mathsf{t}}) < 1 - \varepsilon$, *then* $\mathsf{val}_{G'}(\psi'_{\mathsf{s}} \rightsquigarrow \psi'_{\mathsf{t}}) < 1 - \overline{\varepsilon}$, *where* $\overline{\varepsilon} = \overline{\varepsilon}(\varepsilon)$ *is some computable function such that* $\overline{\varepsilon} \in (0,1)$ *if* $\varepsilon \in (0,1)$.

For an assignment $\psi'\colon V' \to \Sigma'$ for G', let $\mathsf{PLR}(\psi')\colon V \to \Sigma$ denote an assignment for G such that $\mathsf{PLR}(\psi')(v)$ for $v \in V$ is determined based on the *plurality vote* of $\psi'(v')$ over $v' \in \mathsf{cloud}(v)$; namely,

$$\mathsf{PLR}(\psi')(v) \triangleq \operatorname*{argmax}_{\alpha\in\Sigma} \left|\left\{v' \in \mathsf{cloud}(v) : \alpha \in \psi'(v')\right\}\right|, \tag{11}$$

where ties are arbitrarily broken according to any prefixed ordering over Σ (e.g., $\mathsf{a} \prec \mathsf{b} \prec \mathsf{c}$). Suppose we have a reconfiguration sequence $\boldsymbol{\psi}' = \langle \psi'^{(0)} = \psi'_{\mathsf{s}}, \ldots, \psi'^{(\ell)} = \psi'_{\mathsf{t}}\rangle$ for $(G', \psi'_{\mathsf{s}}, \psi'_{\mathsf{t}})$ with the maximum value. Construct then a sequence of assignments, $\boldsymbol{\psi} \triangleq \langle \psi^{(i)}\rangle_{i\in[0..\ell]}$, such that $\psi^{(i)} \triangleq \mathsf{PLR}(\psi'^{(i)})$ for all $i \in [0\,..\,\ell]$. Observe that $\boldsymbol{\psi}$ is a valid reconfiguration sequence for $(G, \psi_{\mathsf{s}}, \psi_{\mathsf{t}})$, and we thus must have $\mathsf{val}_G(\boldsymbol{\psi}) < 1 - \varepsilon$; in particular, there exists

some $\psi'^{(i)}$ such that $\mathsf{val}_G(\mathsf{PLR}(\psi'^{(i)})) = \mathsf{val}_G(\psi^{(i)}) < 1-\varepsilon$. We would like to show that $\mathsf{val}_{G'}(\psi'^{(i)}) < 1-\bar{\varepsilon}$ for some $\bar{\varepsilon} \in (0,1)$. Hereafter, we denote $\psi \triangleq \psi^{(i)}$ and $\psi' \triangleq \psi'^{(i)}$ for notational simplicity.

For each vertex $v \in V$, we define D_v as the set of vertices in $\mathsf{cloud}(v)$ whose values *disagree* with the plurality vote $\psi(v)$; namely,

$$D_v \triangleq \Big\{ v' \in \mathsf{cloud}(v) : \psi(v) \notin \psi'(v') \Big\}. \tag{12}$$

Consider any edge $e = (v,w) \in E$ violated by ψ (i.e., $(\psi(v),\psi(w)) \notin \pi_e$), and let $e' = (v',w') \in E'$ be a unique (inter-cloud) edge such that $v' \in \mathsf{cloud}(v)$ and $w' \in \mathsf{cloud}(w)$. By definition of $\pi'_{e'}$, either of the following must hold: (1) edge e' is violated by ψ' (i.e., $(\psi'(v'),\psi'(w')) \notin \pi'_{e'}$), or (2) $\psi(v) \notin \psi'(v')$ (i.e., $v' \in D_v$) or $\psi(w) \notin \psi'(w')$ (i.e., $w' \in D_w$). Consequently, the number of edges in E violated by ψ is bounded by the sum of the number of edges in E' violated by ψ' and the number of vertices in V' who disagree with the plurality vote; namely,

$$\varepsilon|E| < (\#\text{ edges violated by } \psi') + \sum_{v \in V} |D_v|. \tag{13}$$

Then, one of the two terms on the right-hand side should be greater than $\frac{\varepsilon}{2}|E|$. If the number of edges violated by ψ' is more than $\frac{\varepsilon}{2}|E|$, then we have done because

$$\mathsf{val}_{G'}(\psi') \leqslant \frac{|E'| - (\#\text{ edges violated by } \psi')}{|E'|} < 1 - \frac{\varepsilon}{2}\frac{|E|}{|E'|} \leqslant 1 - \frac{\varepsilon}{2 \cdot n_0(d_0)}. \tag{14}$$

We now consider the case that $\sum_{v \in V} |D_v| > \frac{\varepsilon}{2}|E|$. Define $x_v \triangleq |D_v|/d_G(v)$ for each $v \in V$, which is the fraction of vertices in $\mathsf{cloud}(v)$ who disagree with $\psi(v)$. We also define $\delta \triangleq \frac{\varepsilon}{8}$. We first show that the total size of $|D_v|$ *conditioned on* $x_v \geqslant \delta$ is $\Theta(\varepsilon|E|)$.

▷ Claim 3.12 (∗). $\displaystyle\sum_{v \in V : x_v \geqslant \delta} |D_v| > \frac{\varepsilon}{4}|E|$, where $\delta = \frac{\varepsilon}{8}$.

We then discover a pair of disjoint subsets of $\mathsf{cloud}(v)$ for every $v \in V$ such that their size is $\Theta(|D_v|)$ and they are mutually conflicting under ψ', where the fact that $|\Sigma| = 3$ somewhat simplifies the proof by cases.

▷ Claim 3.13 (∗). For each vertex v of V, there exists a pair of disjoint subsets S, T of $\mathsf{cloud}(v)$ such that $|S| \geqslant \frac{|D_v|}{3}$, $|T| \geqslant \frac{|D_v|}{3}$, and ψ' violates all constraints between S and T.

Consider a vertex $v \in V$ such that $x_v \geqslant \delta$; that is, at least δ-fraction of vertices in $\mathsf{cloud}(v)$ disagree with $\psi(v)$. Letting S and T be two disjoint subsets of $\mathsf{cloud}(v)$ obtained by Claim 3.13, we wish to bound the number of edges between S and T (i.e., $e(S,T)$) using the expander mixing lemma. Hereafter, we determine the value of d_0 by $d_0 \triangleq \left(\frac{12}{\delta}\right)^2 = \frac{9216}{\varepsilon^2}$, which is a positive *even* integer (so that Theorem 3.7 is applicable) and depends only on the value of ε. Suppose first $d_G(v) \geqslant n_0(d_0)$; i.e., X_v is an expander graph.

▶ **Lemma 3.14.** *For a vertex v of V such that $x_v \geqslant \delta$ and $d_G(v) \geqslant n_0(d_0)$, let S and T be a pair of disjoint subsets of $\mathsf{cloud}(v)$ obtained by Claim 3.13. Then, $e(S,T) \geqslant \frac{8}{\delta}|D_v|$.*

Proof sketch. Recall that X_v is a $(d_G(v), d_0, \lambda)$-expander graph, where $\lambda \leqslant 2\sqrt{d_0}$. By applying the expander mixing lemma on S and T, we obtain

$$e(S,T) \geqslant \frac{d_0|S|\cdot|T|}{d_G(v)} - \lambda\sqrt{|S|\cdot|T|} \geqslant \underbrace{\frac{|S|\cdot|T|}{d_G(v)}\left(\frac{12}{\delta}\right)^2 - \frac{2\cdot 12}{\delta}\sqrt{|S|\cdot|T|}}_{=\underline{e}(S,T)}. \tag{15}$$

It is easy to see that $\underline{e}(S,T)$ is monotonically increasing in $\sqrt{|S|\cdot|T|}$ when $\sqrt{|S|\cdot|T|} > \frac{\delta}{12}d_G(v)$. Observing that $\sqrt{|S|\cdot|T|} \geqslant \frac{\delta}{3}d_G(v)$ since $|S| \geqslant \frac{x_v}{3}d_G(v)$, $|T| \geqslant \frac{x_v}{3}d_G(v)$, and $x_v \geqslant \delta$ by assumption, we derive

$$e(S,T) \geqslant \underline{e}(S,T) \geqslant \frac{1}{d_G(v)}\left(\frac{x_v\cdot d_G(v)}{3}\right)^2\left(\frac{12}{\delta}\right)^2 - \frac{2\cdot 12}{\delta}\frac{x_v\cdot d_G(v)}{3}$$
$$\underbrace{\geqslant}_{\text{use } x_v\geqslant\delta} \frac{1}{d_G(v)}\left(\frac{x_v\cdot d_G(v)}{3}\right)\left(\frac{\delta\cdot d_G(v)}{3}\right)\left(\frac{12}{\delta}\right)^2 - \frac{2\cdot 12}{\delta}\frac{x_v\cdot d_G(v)}{3} = \frac{8}{\delta}|D_v|. \blacktriangleleft$$

Suppose then $d_G(v) < n_0(d_0)$. Since X_v forms a complete graph over $d_G(v)$ vertices, $e(S,T)$ is exactly equal to $|S|\cdot|T|$, which is evaluated as

$$e(S,T) = |S|\cdot|T| \underbrace{\geqslant}_{\text{Claim 3.13}} \left(\frac{|D_v|}{3}\right)^2 = \frac{x_v\cdot d_G(v)}{9}|D_v| \underbrace{\geqslant}_{\text{use } d_G(v)\geqslant 1} \frac{\delta}{9}|D_v|. \tag{16}$$

By Lemma 3.14 and Eq. (16), for every vertex $v \in V$ such that $x_v \geqslant \delta$, the number of violated intra-cloud edges within X_v is at least $\min\{\frac{8}{\delta}, \frac{\delta}{9}\}|D_v| \geqslant \frac{\delta}{9}|D_v|$. Simple calculation using Claim 3.12 bounds the total number of edges violated from below as

$$\sum_{v\in V}(\#\text{ violated edges in } X_v) \geqslant \sum_{v:x_v\geqslant\delta}\frac{\delta}{9}|D_v| \underbrace{>}_{\text{Claim 3.12}} \frac{\varepsilon}{72}\frac{\varepsilon}{4}|E| \geqslant \frac{\varepsilon^2\cdot|E'|}{288\cdot n_0(d_0)}. \tag{17}$$

Consequently, from Eqs. (14) and (17), we conclude that

$$\mathsf{val}_{G'}(\boldsymbol{\psi}') \leqslant \mathsf{val}_{G'}(\psi') < \max\left\{1-\frac{\varepsilon}{2\cdot n_0(d_0)}, 1-\frac{\varepsilon^2}{288\cdot n_0(d_0)}\right\} = 1-\frac{\varepsilon^2}{288\cdot n_0\left(\frac{9216}{\varepsilon^2}\right)}.$$

Setting $\overline{\varepsilon} \triangleq \frac{\varepsilon^2}{288\cdot n_0\left(\frac{9216}{\varepsilon^2}\right)}$ accomplishes the proof of Lemma 3.11 and thus Lemma 3.5. ◀

4 Applications

Here, we apply Theorem 3.1 to devise conditional **PSPACE**-hardness of approximation for Nondeterministic Constraint Logic and popular reconfiguration problems on graphs.

4.1 Optimization Variant of Nondeterministic Constraint Logic

First, we review Nondeterministic Constraint Logic developed by Hearn and Demaine [24, 25]. An AND/OR *graph* is defined as an undirected graph $G = (V, E)$, where each edge of E is colored *red* or *blue* and has weight 1 or 2, respectively, and each node of V is one of two types:[4]

- AND *node*, which has two incident red edges and one incident blue edge, or
- OR *node*, which has three incident blue edges.

An orientation (i.e., an assignment of direction to each edge) for G *satisfies* a particular node of G if the total weight of incoming edges is at least 2, and *satisfies* G if all nodes are satisfied. AND and OR nodes behave like the corresponding logical gates: the blue edge of an AND node can be directed outward if and only if both two red edges are directed inward; a particular

[4] We refer to vertices of an AND/OR graph as *nodes* to distinguish from those of a standard graph.

blue edge of an OR node can be directed outward if and only if at least one of the other two blue edges is directed inward. Thus, a direction of each edge can be considered a *signal*. In the **PSPACE**-complete Nondeterministic Constraint Logic problem, for an AND/OR graph G and two satisfying orientations O_{s} and O_{t} for G, we are asked if O_{s} can be transformed into O_{t} by repeating the reversal of a single edge while ensuring that every intermediate orientation satisfies G.[5]

We now formulate an optimization variant of Nondeterministic Constraint Logic, where we are allowed to use an orientation that does *not* satisfy some nodes. Once more, we define $\mathsf{val}_G(\cdot)$ analogously: Let $\mathsf{val}_G(O)$ denote the fraction of nodes satisfied by orientation O, let

$$\mathsf{val}_G(\mathcal{O}) \triangleq \min_{O^{(i)} \in \mathcal{O}} \mathsf{val}_G(O^{(i)}) \tag{18}$$

for reconfiguration sequence $\mathcal{O} = \langle O^{(i)} \rangle_{i \in [0..\ell]}$, and let

$$\mathsf{val}_G(O_{\mathsf{s}} \rightsquigarrow O_{\mathsf{t}}) \triangleq \max_{\mathcal{O} = \langle O_{\mathsf{s}}, \dots, O_{\mathsf{t}} \rangle} \mathsf{val}_G(\mathcal{O}) \tag{19}$$

for two orientations O_{s} and O_{t} for G. In Maxmin Nondeterministic Constraint Logic, we wish to maximize $\mathsf{val}_G(\mathcal{O})$ subject to $\mathcal{O} = \langle O_{\mathsf{s}}, \dots, O_{\mathsf{t}} \rangle$. $\mathsf{Gap}_{c,s}$ Nondeterministic Constraint Logic requests to distinguish whether $\mathsf{val}_G(O_{\mathsf{s}} \rightsquigarrow O_{\mathsf{t}}) \geqslant c$ or $\mathsf{val}_G(O_{\mathsf{s}} \rightsquigarrow O_{\mathsf{t}}) < s$. In what follows, RIH implies **PSPACE**-hardness of approximation for Maxmin Nondeterministic Constraint Logic.

▶ **Proposition 4.1.** *For every $B \in \mathbb{N}$ and $\varepsilon \in (0,1)$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ E3-SAT(B) Reconfiguration *to* $\mathsf{Gap}_{1,1-\Theta(\frac{\varepsilon}{B})}$ Nondeterministic Constraint Logic.

Our proof makes a modification to the CNF network [24, 25]. To this end, we borrow special nodes that can be simulated by an AND/OR subgraph, including CHOICE, RED–BLUE, FANOUT nodes, and free-edge terminators, which are described below; see also Hearn and Demaine [24, 25] for more details.

- CHOICE *node*: This node has three red edges and is satisfied if at least two edges are directed inward; i.e., only one edge may be directed outward.
- RED–BLUE *node*: This is a degree-2 node incident to one red edge and one blue edge, which acts as transferring a signal between them; i.e., one edge may be directed outward if and only if the other is directed inward.
- FANOUT *node*: This node is equivalent to an AND node from a different interpretation: two red edges may be directed outward if and only if the blue edge is directed inward. Accordingly, a FANOUT node plays a role in *splitting* a signal.
- *Free-edge terminator*: This is an AND/OR subgraph of constant size used to connect the loose end of an edge. The connected edge is free in a sense that it can be directed inward or outward.

Reduction. Given an instance $(\varphi, \sigma_{\mathsf{s}}, \sigma_{\mathsf{t}})$ of Maxmin E3-SAT(B) Reconfiguration, where φ is an E3-CNF formula consisting of m clauses $C_1, \dots, C_m$ over n variables $x_1, \dots, x_n$, and σ_{s} and σ_{t} satisfy φ, we construct an AND/OR graph G_φ as follows. For each variable x_i of φ, we create a CHOICE node, denoted v_{x_i}, called a *variable node*. Of the three red edges incident

[5] A variant of Nondeterministic Constraint Logic, called *configuration-to-edge* [24], requires to decide if a specified edge can be eventually reversed by a sequence of edge reversals. From a point of view of approximability, this definition does not seem to make much sense.

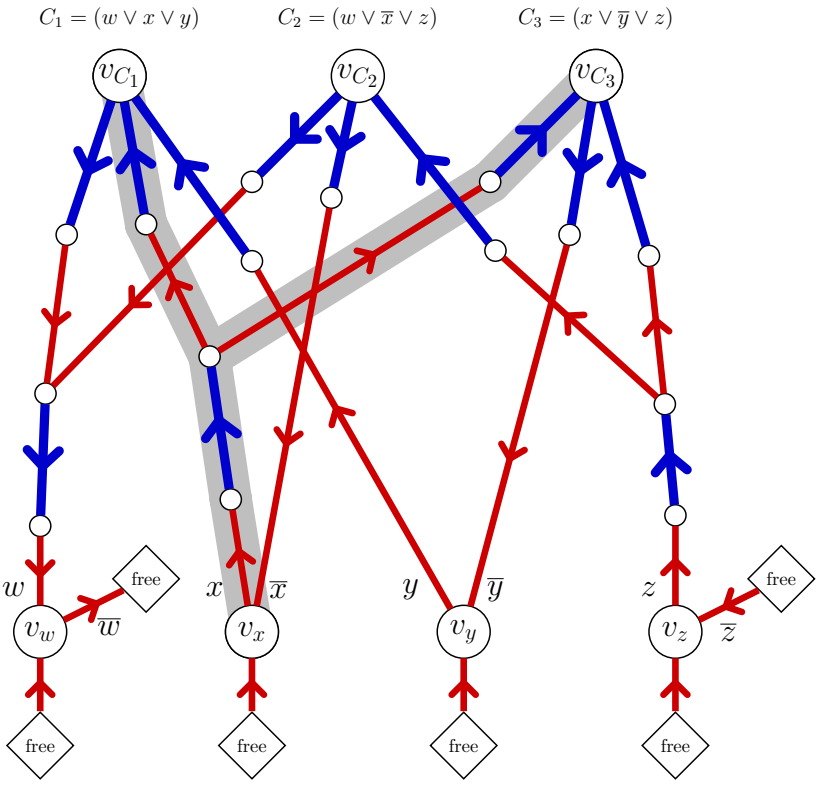


Figure 3 An AND/OR graph G_φ corresponding to an E3-CNF formula $\varphi = (w \vee x \vee y) \wedge (w \vee \overline{x} \vee z) \wedge (x \vee \overline{y} \vee z)$, taken and modified from [25, Figure 5.1]. Here, thicker blue edges have weight 2, thinner red edges have weight 1, and the square node denotes a free edge terminator. The orientation of G_φ shown above is given by O_{ψ_s} such that $\psi_\mathsf{s}(w, x, y, z) = (\mathtt{F}, \mathtt{T}, \mathtt{T}, \mathtt{T})$. If ψ_t is defined as $\psi_\mathsf{t}(w, x, y, z) = (\mathtt{F}, \mathtt{F}, \mathtt{T}, \mathtt{T})$, we can transform O_{ψ_s} into O_{ψ_t}; in particular, edges in the subtree rooted at x, denoted the gray area, can be made directed downward.

to v_{x_i}, one is connected to a free edge terminator, whereas the other two are labeled "x_i" and "$\overline{x_i}$". Thus, either of x_i or $\overline{x_i}$ can be directed outward without breaking v_{x_i}. For each clause C_j of φ, we create an OR node, denoted v_{C_j}, called a *clause node*. The output signals of variable nodes are sent toward the corresponding clause nodes. Specifically, if literal ℓ appears in multiple clauses of φ, we first make a desired number of copied signals of edge ℓ using RED–BLUE and FANOUT nodes; if ℓ does not appear in any clause, we connect the edge ℓ to a free edge terminator. Then, for each clause $C_j = (\ell_1 \vee \ell_2 \vee \ell_3)$ of φ, the clause node v_{C_j} is connected to three edges corresponding to the (copied) signals of ℓ_1, ℓ_2, ℓ_3. This completes the construction of G_φ. See Figure 3 for an example.

Observe that G_φ is satisfiable if and only if φ is satisfiable [24, 25]. In fact, a satisfying orientation O_σ for G_φ can be obtained from any satisfying truth assignment σ for φ. Here, the trick is that if a literal x_i or $\overline{x_i}$ evaluates to $\mathtt{T}$ by σ and appears in clause C_j, we can safely orient *every* edge on the unique path between v_{x_i} and v_{C_j} toward v_{C_j}. Constructing O_s from σ_s and O_t from σ_t according to this procedure, we obtain an instance $(G, O_\mathsf{s}, O_\mathsf{t})$ of Maxmin Nondeterministic Constraint Logic, which completes the reduction. The proof of the correctness shown below relies on the fact that for fixed $B \in \mathbb{N}$, the number of nodes $|V(G_\varphi)|$ is proportional to the number of variable nodes n as well as that of clause nodes m.

Proof sketch of Proposition 4.1. We begin with a few remarks on the construction of G_φ. For each clause C_j that includes x_i or $\overline{x_i}$, there is a *unique path* between v_{x_i} and v_{C_j} without passing through any other variable or clause node, which takes the following form:

Output signal of a variable node v_{x_i}
$\to$ a RED–BLUE node
$\to$ any number of (a FANOUT node $\to$ a RED–BLUE node)
$\to$ a clause node v_{C_j}.

Therefore, every node except variable and clause nodes is uniquely associated with a particular literal ℓ of φ. Hereafter, the *subtree rooted at literal* ℓ is defined as a subgraph of G_φ induced by the unique paths between the corresponding variable node and v_{C_j}'s for all clauses C_j including ℓ (see also Figure 3).

Since the completeness is almost immediate from the above observation, we next prove the soundness; i.e., $\mathsf{val}_\varphi(\sigma_\mathsf{s} \rightsquigarrow \sigma_\mathsf{t}) < 1-\varepsilon$ implies $\mathsf{val}_{G_\varphi}(O_\mathsf{s} \rightsquigarrow O_\mathsf{t}) < 1 - \Theta(\frac{\varepsilon}{B})$. Let $\mathcal{O} = \langle O^{(0)} = O_\mathsf{s}, \dots, O^{(\ell)} = O_\mathsf{t} \rangle$ be any reconfiguration sequence for $(G_\varphi, O_\mathsf{s}, O_\mathsf{t})$. Construct then a sequence of truth assignments, $\boldsymbol{\sigma} = \langle \sigma^{(i)} \rangle_{i \in [0..\ell]}$, such that each $\sigma^{(i)}(x_j)$ for variable x_j is defined to be `T` if edge x_i is directed outward from v_{x_i} and edge $\overline{x_i}$ is directed inward to v_{x_i}, and to be `F` otherwise. Since $\boldsymbol{\sigma}$ is a valid reconfiguration sequence for $(\varphi, \sigma_\mathsf{s}, \sigma_\mathsf{t})$, it holds that $\mathsf{val}_\varphi(\boldsymbol{\sigma}) < 1-\varepsilon$; in particular, there exists some $\sigma^{(i)}$ such that $\mathsf{val}_\varphi(\sigma^{(i)}) < 1-\varepsilon$. However, the number of clause nodes satisfied by $O^{(i)}$ may not be less than $m(1-\varepsilon)$ because other nodes may be violated in lieu of them (e.g., both of x_i and $\overline{x_i}$ may be directed outward). Thus, we compare $O^{(i)}$ with an orientation $O_{\sigma^{(i)}}$ constructed from $\sigma^{(i)}$ by the procedure described in the reduction paragraph. Note that $O_{\sigma^{(i)}}$ satisfies every non-clause node, while more than εm clause nodes are not satisfied. Transforming $O_{\sigma^{(i)}}$ into $O^{(i)}$ by reversing the directions of edges one by one, we can see that each time a non-clause node is violated, we would be able to make at most B clause nodes satisfied. Consequently, we derive

$$\begin{aligned}
&\varepsilon m - B \cdot (\#\text{ non-clause nodes violated by } O^{(i)}) < (\#\text{ clause nodes violated by } O^{(i)}) \\
&\implies (\#\text{ violated nodes by } O^{(i)}) > \frac{\varepsilon}{B} m \\
&\implies \mathsf{val}_{G_\varphi}(\mathcal{O}) \leqslant \mathsf{val}_{G_\varphi}(O^{(i)}) < \frac{|V(G_\varphi)| - \frac{\varepsilon}{B} m}{|V(G_\varphi)|} = 1 - \Theta\left(\frac{\varepsilon}{B}\right), \qquad (20)
\end{aligned}$$

where we used the fact that $|V(G_\varphi)| = \Theta(m+n) = \Theta(m)$, completing the proof. ◀

4.2 Reconfiguration Problems on Graphs

Independent Set Reconfiguration. Denote by $\alpha(G)$ the size of a maximum independent set of a graph G. For a pair of independent sets I_s and I_t of G, Independent Set Reconfiguration asks if there is a sequence of independent sets of G from I_s to I_t, each resulting from the previous one by either adding or removing a single vertex of G,[6] without going through an independent set of size less than $\min\{|I_\mathsf{s}|, |I_\mathsf{t}|\} - 1$. For a reconfiguration sequence $\mathcal{I} = \langle I^{(i)} \rangle_{i \in [0..\ell]}$ of independent sets of a graph G, we define $\mathsf{val}_G(\mathcal{I}) \triangleq \min_{I^{(i)} \in \mathcal{I}} \frac{|I^{(i)}|}{\alpha(G)-1}$. Here, division by $\alpha(G)-1$ is derived from the nature that we must remove at least one vertex whenever $|I_\mathsf{s}| = |I_\mathsf{t}| = \alpha(G)$ and $I_\mathsf{s} \neq I_\mathsf{t}$. We then define $\mathsf{val}_G(I_\mathsf{s} \rightsquigarrow I_\mathsf{t}) \triangleq \max_{\mathcal{I} = \langle I_\mathsf{s}, \dots, I_\mathsf{t} \rangle} \mathsf{val}_G(\mathcal{I})$. In Maxmin Independent Set

[6] Such a model of reconfiguration is called *token addition and removal* [28]. We do not consider token jumping [34] or token sliding [24] since they do not change the size of an independent set.

Reconfiguration, we wish to maximize $\mathsf{val}_G(\mathcal{I})$ subject to $\mathcal{I} = \langle I_\mathsf{s}, \ldots, I_\mathsf{t} \rangle$, which is **NP**-hard to approximate within any constant factor [28]. $\mathsf{Gap}_{c,s}$ Independent Set Reconfiguration requests to distinguish whether $\mathsf{val}_G(I_\mathsf{s} \rightsquigarrow I_\mathsf{t}) \geqslant c$ or $\mathsf{val}_G(I_\mathsf{s} \rightsquigarrow I_\mathsf{t}) < s$. The proof of the following corollary is based on a Karp reduction due to [24, 25].

▶ **Corollary 4.2** (∗). *For every $\varepsilon \in (0, 1)$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ Nondeterministic Constraint Logic *to* $\mathsf{Gap}_{1,1-\Theta(\varepsilon)}$ Independent Set Reconfiguration.

As an immediate corollary, Maxmin Clique Reconfiguration is **PSPACE**-hard to approximate under Hypothesis 2.4.

▶ **Corollary 4.3.** *For every $\varepsilon \in (0, 1)$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ Nondeterministic Constraint Logic *to* $\mathsf{Gap}_{1,1-\Theta(\varepsilon)}$ Clique Reconfiguration.

Vertex Cover Reconfiguration. We conclude this section with a conditional inapproximability result of Minmax Vertex Cover Reconfiguration, which is 2-factor approximable [28]. Refer to the full version [38] for the formal definition. The proof uses a gap-preserving reduction from Maxmin Independent Set Reconfiguration on restricted graphs due to Bonsma and Cereceda [9].

▶ **Corollary 4.4** (∗). *For every $\varepsilon \in (0, 1)$, there exists a gap-preserving reduction from* $\mathsf{Gap}_{1,1-\varepsilon}$ Nondeterministic Constraint Logic *to* $\mathsf{Gap}_{1,1+\Theta(\varepsilon)}$ Vertex Cover Reconfiguration.

5 Conclusions

We gave a series of gap-preserving reductions to demonstrate **PSPACE**-hardness of approximation for optimization variants of popular reconfiguration problems *assuming* Reconfiguration Inapproximability Hypothesis (RIH). An immediate open question is to verify RIH. One approach is to prove it directly, e.g., by using gap amplification of Dinur [19]. Some steps may be more difficult to prove, as we are required to preserve reconfigurability. Another way entails a reduction from some problems already known to be **PSPACE**-hard to approximate, such as True Quantified Boolean Formula due to Condon, Feigenbaum, Lund, and Shor [15]. We are currently uncertain whether we can "adapt" a Karp reduction from True Quantified Boolean Formula to Nondeterministic Constraint Logic [24, 25].

References

1 Noga Alon. Explicit expanders of every degree and size. *Comb.*, 41(4):447–463, 2021.
2 Noga Alon and Fan R. K. Chung. Explicit construction of linear sized tolerant networks. *Discret. Math.*, 72(1-3):15–19, 1988.
3 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
4 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.
5 Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory Comput. Syst.*, 65(4):662–686, 2021.
6 Alexandre Blanché, Haruka Mizuta, Paul Ouvrard, and Akira Suzuki. Decremental optimization of dominating sets under the reconfiguration framework. In *IWOCA*, pages 69–82, 2020.
7 Marthe Bonamy, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Moritz Mühlenthaler, Akira Suzuki, and Kunihiro Wasa. Shortest reconfiguration of colorings under Kempe changes. In *STACS*, pages 35:1–35:14, 2020.

8 Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski. Complexity of token swapping and its variants. *Algorithmica*, 80(9):2656–2682, 2018.

9 Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theor. Comput. Sci.*, 410(50):5215–5226, 2009.

10 Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito, and Moritz Mühlenthaler. Shortest reconfiguration of matchings. In *WG*, pages 162–174, 2019.

11 Nicolas Bousquet, Takehiro Ito, Yusuke Kobayashi, Haruka Mizuta, Paul Ouvrard, Akira Suzuki, and Kunihiro Wasa. Reconfiguration of spanning trees with degree constraint or diameter constraint. In *STACS*, pages 15:1–15:21, 2022.

12 Nicolas Bousquet and Alice Joffard. Approximating shortest connected graph transformation for trees. In *SOFSEM*, pages 76–87, 2020.

13 Nicolas Bousquet and Arnaud Mary. Reconfiguration of graphs with connectivity constraints. In *WAOA*, pages 295–309, 2018.

14 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *J. Graph Theory*, 67(1):69–82, 2011.

15 Anne Condon, Joan Feigenbaum, Carsten Lund, and Peter W. Shor. Probabilistically checkable debate systems and nonapproximability of PSPACE-hard functions. *Chic. J. Theor. Comput. Sci.*, 1995, 1995.

16 Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *CCC*, pages 262–273, 1997.

17 Pierluigi Crescenzi and Luca Trevisan. On approximation scheme preserving reducibility and its applications. *Theory Comput. Syst.*, 33(1):1–16, 2000.

18 Mark de Berg, Bart M. P. Jansen, and Debankur Mukherjee. Independent-set reconfiguration thresholds of hereditary graph classes. *Discret. Appl. Math.*, 250:165–182, 2018.

19 Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

20 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 2012.

21 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

22 Kshitij Gajjar, Agastya Vibhuti Jha, Manish Kumar, and Abhiruk Lahiri. Reconfiguring shortest paths in graphs. In *AAAI*, pages 9758–9766, 2022.

23 Parikshit Gopalan, Phokion G. Kolaitis, Elitza Maneva, and Christos H. Papadimitriou. The connectivity of Boolean satisfiability: Computational and structural dichotomies. *SIAM J. Comput.*, 38(6):2330–2355, 2009.

24 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor. Comput. Sci.*, 343(1-2):72–96, 2005.

25 Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A K Peters, Ltd., 2009.

26 Lenwood S. Heath and John Paul C. Vergara. Sorting by short swaps. *J. Comput. Biol.*, 10(5):775–789, 2003.

27 Takehiro Ito and Erik D. Demaine. Approximability of the subset sum reconfiguration problem. *J. Comb. Optim.*, 28(3):639–654, 2014.

28 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theor. Comput. Sci.*, 412(12-14):1054–1065, 2011.

29 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Shortest reconfiguration of perfect matchings via alternating cycles. *SIAM J. Discret. Math.*, 36(2):1102–1123, 2022.

30 Takehiro Ito, Marcin Kamiński, and Erik D. Demaine. Reconfiguration of list edge-colorings in a graph. *Discret. Appl. Math.*, 160(15):2199–2207, 2012.

31 Takehiro Ito, Haruka Mizuta, Naomi Nishimura, and Akira Suzuki. Incremental optimization of independent sets under the reconfiguration framework. *J. Comb. Optim.*, 43(5):1264–1279, 2022.

32 Takehiro Ito, Hiroyuki Nooka, and Xiao Zhou. Reconfiguration of vertex covers in a graph. *IEICE Trans. Inf. Syst.*, 99-D(3):598–606, 2016.

33 Matti Järvisalo and Ilkka Niemelä. A compact reformulation of propositional satisfiability as binary constraint satisfaction. In *Third International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 111–124, 2004.

34 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012.

35 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. In *ESA*, pages 66:1–66:15, 2016.

36 Sidhanth Mohanty, Ryan O'Donnell, and Pedro Paredes. Explicit near-Ramanujan graphs of every degree. *SIAM J. Comput.*, 51(3):STOC20–1–STOC20–23, 2021.

37 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

38 Naoto Ohsaka. Gap preserving reductions between reconfiguration problems. *CoRR*, abs/2212.04207, 2022.

39 Naoto Ohsaka and Tatsuya Matsuoka. Reconfiguration problems on submodular functions. In *WSDM*, pages 764–774, 2022.

40 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

41 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409, pages 127–160. Cambridge University Press, 2013.

42 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theor. Comput. Sci.*, 586:81–94, 2015.

43 Yusuke Yanagisawa, Akira Suzuki, Yuma Tamura, and Xiao Zhou. Decremental optimization of vertex-coloring under the reconfiguration framework. In *COCOON*, pages 355–366, 2021.

Dynamic Data Structures for Parameterized String Problems

Jędrzej Olkowski
Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland

Mateusz Rychlicki
School of Computing, University of Leeds, UK

Karol Węgrzycki
Saarland University, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarbrücken, Germany

Anna Zych-Pawlewicz
Institute of Informatics, University of Warsaw, Poland

Abstract

We revisit classic string problems considered in the area of parameterized complexity, and study them through the lens of dynamic data structures. That is, instead of asking for a static algorithm that solves the given instance efficiently, our goal is to design a data structure that efficiently maintains a solution, or reports a lack thereof, upon updates in the instance.

We first consider the Closest String problem, for which we design randomized dynamic data structures with amortized update times $d^{\mathcal{O}(d)}$ and $|\Sigma|^{\mathcal{O}(d)}$, respectively, where Σ is the alphabet and d is the assumed bound on the maximum distance. These are obtained by combining known static approaches to Closest String with color-coding.

Next, we note that from a result of Frandsen et al. [J. ACM'97] one can easily infer a meta-theorem that provides dynamic data structures for parameterized string problems with worst-case update time of the form $\mathcal{O}_k(\log \log n)$, where k is the parameter in question and n is the length of the string. We showcase the utility of this meta-theorem by giving such data structures for problems Disjoint Factors and Edit Distance. We also give explicit data structures for these problems, with worst-case update times $\mathcal{O}(k2^k \log \log n)$ and $\mathcal{O}(k^2 \log \log n)$, respectively. Finally, we discuss how a lower bound methodology introduced by Amarilli et al. [ICALP'21] can be used to show that obtaining update time $\mathcal{O}(f(k))$ for Disjoint Factors and Edit Distance is unlikely already for a constant value of the parameter k.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability; Theory of computation → Predecessor queries

Keywords and phrases Parameterized algorithms, Dynamic data structures, String problems, Closest String, Edit Distance, Disjoint Factors, Predecessor problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.50

Related Version *Full Version*: https://arxiv.org/abs/2205.00441

Funding This work is the result of research conducted within research project number 2017/26/D/ST6/00264 financed by National Science Centre, Poland (Jędrzej Olkowski and Anna Zych-Pawlewicz). This work is a part of projects BOBR (Michał Pilipczuk) and TIPEA (Karol Węgrzycki) that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreements no. 948057 and 850979, respectively). Mateusz Rychlicki acknowledges support from the Engineering and Physical Sciences Research Council (EPSRC, project EP/V00252X/1).

© Jędrzej Olkowski, Michał Pilipczuk, Mateusz Rychlicki, Karol Węgrzycki, and Anna Zych-Pawlewicz;
licensed under Creative Commons License CC-BY 4.0

40th International Symposium on Theoretical Aspects of Computer Science (STACS 2023).
Editors: Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté;
Article No. 50; pp. 50:1–50:22

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

1 Introduction

The field of parameterized complexity is based on the principle of *parameterization*: measuring the usage of resources not only in terms of the total input size, but also in terms of auxiliary complexity measures called *parameters*. Traditionally, the principle is applied to static algorithms and their running times, but the idea can be – and has been – used within essentially every algorithmic paradigm. Among these, a recent line of research has identified the area of dynamic data structures as one where the application of the parameterized approach leads to new and interesting results, see e.g. [3, 15, 17, 18, 23, 30]. In this work, we continue this promising direction by investigating classic string problems considered in parameterized complexity.

Arguably, the most widely known parameterized string problem is Closest String.

Closest String
Input: Integer d and words $s_1, s_2, \ldots, s_n \in \Sigma^L$ over an alphabet Σ, each of length L
Task: Decide whether there exists a word $c \in \Sigma^L$ such that for every $i \in \{1, \ldots, n\}$, the Hamming distance between s_i and c is at most d.

Closest String has several natural parameters: n, d, L, $|\Sigma|$. For the parameterization by d and Σ, Gramm et al. [22] gave a $d^{\mathcal{O}(d)} \cdot (nL)^{\mathcal{O}(1)}$-time algorithm, while Ma and Sun [29] gave a $|\Sigma|^{\mathcal{O}(d)} \cdot (nL)^{\mathcal{O}(1)}$-time algorithm. By now, these are literally textbook examples of the technique of branching [16, Theorem 3.14 and Exercise 3.25], and their running time dependence on d is known to be asymptotically optimal under the Exponential Time Hypothesis (ETH) [28]. For the parameterization by n, the classic algorithm of Gramm et al. [22, Section 4] solves the problem in time $2^{n^{\mathcal{O}(n)}} \cdot L^{\mathcal{O}(1)}$ by a reduction to integer programming in dimension $n^{\mathcal{O}(n)}$. Recently, Koutecký et al. [25] improved this running time to $n^{\mathcal{O}(n^2)} \cdot L^{\mathcal{O}(1)}$ using exciting developments in parameterized algorithms for block-structured integer programs. Kernelization algorithms for Closest String were studied in [8].

We study the *dynamic variant* of Closest String, which is to design a dynamic data structure supporting the following operations:

- *Initialize* the data structure for a given instance of Closest String.
- *Update* the data structure upon modification of a single symbol in a single string s_i.
- *Query* whether the current instance is a yes-instance of Closest String.

Note that parameters n, d, L, and Σ are fixed on the initialization and do not change over the life of the data structure; only the strings $s_1, \ldots, s_n$ can be modified, and by one symbol at the time. Also, we assume that upon query, the data structure is only required to answer *yes* or *no*, and does not need to provide the solution c.

For this variant we give randomized dynamic data structures whose update times match the parametric factors in the runtimes of the algorithms of Gramm et al. [22] and of Ma and Sun [29].

▶ **Theorem 1.** *The dynamic variant of* Closest String *admits a randomized data structure with initialization time* $2^{\mathcal{O}(d)} \cdot nL|\Sigma|^{1+o(1)}$*, amortized update time* $2^{\mathcal{O}(d)}$*, and worst-case query time* $d^{\mathcal{O}(d)}$ *or* $|\Sigma|^{\mathcal{O}(d)}$*, whichever is smaller. The answer to each query may result with a false positive with probability at most* $2^{-\Omega(d)}$*; there are no false negatives.*

In the proof of Theorem 1 we combine the classic approach to Closest String, originating in [22, 29], with an interesting application of color-coding. The randomization comes from the color-coding; we can dispose of it using standard derandomization techniques (see [16, Section 5.6]), but at the cost of introducing an additional $\mathcal{O}(\log(nL))$ factor in the update time. Also, note that by the results of [28], under ETH one cannot expect to improve the query time to $d^{o(d)}$ or $|\Sigma|^{o(d)}$, even in the amortized sense.

Next, we turn attention to other problems. First, we note that combining a result of Frandsen et al. [19] on the dynamic word problem for aperiodic semigroups with the classic Schützenberger-McNaughton-Papert Theorem [32, 40] yields the following meta-theorem [1].

▶ **Theorem 2.** *Suppose Σ is a finite alphabet and $\mathcal{L} \subseteq \Sigma^\star$ is a language definable in logic $\mathsf{FO}[\Sigma, <]$. Then there exists a data structure that for a given word $w \in \Sigma^\star$, which can be updated over time by replacing single symbols, maintains whether $w \in \mathcal{L}$. The data structure can be initialized on a given word w in time $\mathcal{O}(n)$ where $n = |w|$, and then every update takes worst-case time $\mathcal{O}(\log \log n)$.*

Theorem 2 follows immediately from the combination explained above, so we consider it an essentially known result (though we could not find this precise formulation in the literature). What is new is the observation that this result is a very convenient tool for obtaining dynamic data structures in the parameterized setting. We showcase this by considering the following two text problems.

Disjoint Factors
Input: A word $w \in \{1, \ldots, k\}^\star$, where k is an integer **Task:** Decide whether there exist k pairwise disjoint (non-overlapping) substrings $w_1, w_2, \ldots, w_k$ of w such that for each $i \in \{1, \ldots, k\}$, w_i has length at least 2 and begins and ends with symbol i.

Edit Distance
Input: Integer k and two words $u, v \in \Sigma^\star$, where Σ is an alphabet **Task:** Decide whether $\mathsf{ed}(u, v) \leqslant k$, that is, whether v can be obtained from u by a sequence of at most k *edits*, each consisting of a deletion, insertion, or substitution of a single symbol.

Disjoint Factors has been introduced in [11] as a stepping stone for kernelization hardness of the Disjoint Cycles and Disjoint Paths problems. We choose to use it in this work as an example, because its simple combinatorial structure makes many basic ideas clearly visible. On the other hand, Edit Distance is a problem of immense importance with multiple applications (see for survey [34]). It can be solved in time $\mathcal{O}(n^2)$ by standard dynamic programming (where n is the total length of the words). The best currently known algorithm for Edit Distance runs in $\mathcal{O}(n^2/(\log n)^2)$ time [31] and under the Strong ETH, there is no strongly subquadratic algorithm [7, 1, 13, 2]. Here, we focus on parameterization by the size of the solution k. In terms of this parametrization Edit Distance can be solved in $\mathcal{O}(n + k^2)$ by the celebrated Landau and Vishkin algorithm [27] and even in sublinear time when approximation is allowed [9, 6, 21].

We observe that both for Disjoint Factors and for Edit Distance, the language of yes-instances can be defined in $\mathsf{FO}[\Sigma, <]$ using a sentence of length bounded in terms of the parameters. Therefore, by simply applying Theorem 2, we obtain data structures for the dynamic variants of Disjoint Factors and Edit Distance (defined similarly as for Closest String) with worst-case update times $f(k) \cdot \log \log n$ for some computable function $f(k)$. As usual with meta-theorems, the parametric dependence in these complexity guarantees is not explicit. For this reason, we also design explicit data structures for both problems.

[1] See Section A for formal definition of $\mathsf{FO}[\Sigma, <]$

▶ **Theorem 3.** *The dynamic variant of* Disjoint Factors *admits a data structure with initialization time* $\mathcal{O}(k2^k + kn)$*, worst-case query time* $\mathcal{O}(1)$*, and worst-case update time* $\mathcal{O}(k2^k \log\log n)$*.*

▶ **Theorem 4.** *The dynamic variant of* Edit Distance *admits a data structure with initialization time* $\mathcal{O}(kn)$*, worst-case query time* $\mathcal{O}(1)$*, and worst-case update time* $\mathcal{O}(k^2 \log\log n)$*.*

Our key component are van Emde Boas trees [42]. This is not surprising, as van Emde Boas trees are also the main tool underlying the proof of Theorem 2 (see [19]). In both cases, we heavily build upon known static algorithms [27, 11]. We point out that these results serve mainly as a demonstration that one can improve the dependence on the parameter guaranteed by Theorem 2 for concrete problems. We are not aware of any previous works on Disjoint Factors in exactly this dynamic setting. On the other hand Edit Distance was considered in the dynamic setting for unbounded values of k and only polynomial in n updates are known [24, 14, 5]. Landau et al. [26] considered parameterization by k of Edit Distance in the incremental setting. Moreover it is folklore that dynamic Edit Distance (even with insertions and deletions) can be maintained in $k^2 \operatorname{polylog}(n)$ update/query time (by combining [27, 36]). We reiterate that Theorems 3 and 4 mainly serve here as examples that dependence on k in the general framework in Theorem 2 can be improved for concrete problems.

Finally, we observe that we can use the hardness methodology proposed by Amarilli et al. [4] to establish conditional lower bounds against improving the update time in Theorems 3 and 4. More precisely, we prove that already for constant values of the parameters, the problems Disjoint Factors and Edit Distance are *prefix-U_1 hard*, which means that finding a data structure for them is at least as hard as designing a data structure for the following problem: for a dynamic word w over $\{0,1\}^\star$, support queries of the form "given i, is the first/leftmost occurence of the symbol 1 in w at position $\leqslant i$". Amarilli et al. [4] conjectured that no data structure for this problem achieves update time $\mathcal{O}(1)$, and our reduction carries this hardness over to the dynamic variants of Disjoint Factors and Edit Distance. Let us point out that the two discussed problems are just examples, and the obtained hardness methodology can be potentially applied to a other dynamic string problems.

Organization. In the Section 2 we give a short preliminaries. In Section 3 we present a proof of Theorem 1 for large alphabets. In Section 4 we prove Theorem 1 for small alphabets. The remaining proofs are deferred to the appendix. In Appendix A we prove Theorem 2. Appendix B contains omitted proofs.

In the full-version of this paper [37] we include we give dynamic data structures for Disjoint Factors and Edit Distance and show lower bounds for them.

2 Preliminaries

For a parameter ℓ, we write $\mathcal{O}_\ell(\cdot)$ to hide factors depending only on ℓ. The $\operatorname{poly}(n_1, n_2)$ denotes $(n_1 n_2)^{\mathcal{O}(1)}$. We use a shorthand notation $[n] \coloneqq \{1, \ldots, n\}$. For two sets X, Y, $X \triangle Y$ denotes their symmetric difference $(X \setminus Y) \cup (Y \setminus X)$. For two words $u, v \in \Sigma^L$, where $L \in \mathbb{N}$, by $\operatorname{dist}(u, v)$ we denote the Hamming distance between u and v. For a word $u \in \Sigma^L$ and a set $X \subseteq [L]$, we write $u[X] \in \Sigma^{|X|}$ for the word obtained from u by removing all positions outside of X. For $1 \leqslant i \leqslant j \leqslant m$, we write $u[i:j] \in \Sigma^{j-i+1}$ for $u[\{i, \ldots, j\}]$.

Computation Model. In this paper we work in the standard word-RAM model. In all our results the $\mathcal{O}(\log\log n)$ factors come exclusively from application of van Emde Boas trees that solve the Predecessor problem, where one needs to maintain a set S of $n \in \mathbb{N}$ integers with $w \in \mathbb{N}$ bits. In update one can add/remove integers to/from set S. During query, for a given integer x one should returns the largest integer $y \in S$ such that $x \geqslant y$. The Predecessor problem is a well-studied problem both in terms of lower and upper bounds (see the recent survey [35]). In the word-RAM model the complexity of Predecessor operations is well understood to be

$$\Theta\left(\max\left[1, \min\left\{\log_w(n), \frac{\log \frac{w}{\log w}}{\log\left(\log \frac{w}{\log w} / \log \frac{\log n}{\log w}\right)}, \log \frac{\log(2^w - n)}{\log w}\right\}\right]\right) \tag{1}$$

The upper and lower bounds were given by Pătraşcu and Thorup [38], see also [10, 20]. This means that strictly speaking $\mathcal{O}(\log\log n)$ factors in our paper, could be replaced with Equation 1 in the word-RAM model depending on word size. We are using the worse $\mathcal{O}(\log\log n)$ bound in order to keep the results transparent. Note that the $\mathcal{O}(\log\log n)$ bound for the Predecessor is tight in more restricted computation models (see, e.g., [33]).

3 Closest String

In this section, we show the first half of Theorem 1 by proving the following theorem.

▶ **Theorem 5.** *The dynamic variant of* Closest String *admits a randomized data structure with initialization time $2^{\mathcal{O}(d)}nL$, amortized update time $2^{\mathcal{O}(d)}$, and worst-case query time $d^{\mathcal{O}(d)}$. The answer to each query may result with a false positive with probability at most $2^{-\Omega(d)}$; there are no false negatives.*

Throughout this section we fix the parameter $d \in \mathbb{N}$ and denote $\mathcal{S} \coloneqq \{s_1, \ldots, s_n\}$ for brevity, and call it a *dictionary*. Then updates on such a dictionary consist of replacing one symbol in one word with another symbol. Our data structure is based on the static algorithm for Closest String due to Gramm et al. [22].

3.1 Branching for Closest String

Algorithm 1 presents a pseudocode for a $(3d)^d\mathrm{poly}(n, L)$ time algorithm for Closest String loosely based on [22]. We first check if every pair of words of $\mathcal{S}$ are at distance at most $2d$ from each other; otherwise, by triangle inequality, we can safely terminate and return that there is no solution. Following this, we run a recursive search that maintains a candidate q for a solution, together with an upper bound x on how far from q, in terms of Hamming distance, we allow the sought solution to be. Candidate q is initially set to be any word in $\mathcal{S}$ and upper bound x is initially set to d. Within the search, we first verify whether q is already a solution. If yes, then we can terminate, this time yielding a positive answer. Otherwise, there is some $s \in \mathcal{S}$ at distance more than d from q (and at most $2d$). Observe that due to the initial check and the fact that during recursion we modify q at most d times, it will be always the case that s and q differ on at most $3d$ positions. Hence, we can branch over one of at most $3d$ possibilities of modifying q by a single letter so that q gets closer to s. The nontrivial observation is that if there exists a solution, one of the modifications will take us closer to it in terms of the Hamming distance.

Algorithm 1 Pseudocode of static $\mathcal{O}((3d)^d \text{poly}(n, L))$ time algorithm for Closest String. To get a dynamic data structure, use Lemma 6 to perform manipulations on q.

```
Algorithm ClosestString(S, d)
1   if there exist s_i, s_j ∈ S such that dist(s_i, s_j) > 2d then
2       return False
3   Set q to be any word from S
4   return ClosestStringRec(S, q, d)

Procedure ClosestStringRec(S, q, x)
5   if x < 0 then
6       return False
7   if there exists s ∈ S such that dist(s, q) > d then
8       Find P := {i ∈ [L] | s[i] ≠ q[i]}          // Observe that |P| ⩽ 3d
9       for i ∈ P do
10          Set q' = q
11          Replace q'[i] = s[i]
12          if ClosestString(S, q', x − 1) then
13              return True
14      return False
15  return True
```

For the running time, observe that in each call we can make at most $|P| \leqslant 3d$ guesses. Moreover, through the execution of the algorithm we can only modify at most d letters in q. This means that the total size of the recursion tree is $\mathcal{O}((3d)^d)$.[2]

Let us take a closer look at the polynomial factors of the Algorithm 1 and discuss problems with dynamization. In line 1 we need to check if there exist words $s_i, s_j \in \mathcal{S}$ with $\text{dist}(s_i, s_j) > 2d$. Naively, one needs to iterate over every pair of words in $\mathcal{S}$ and compute the distance exactly which already requires n^2 iterations, where $n = |\mathcal{S}|$. Even if somehow, this number could be decreased, observe that in order to compute a distance between a fixed pair of words one needs to at least read them in $\mathcal{O}(L)$ time, which is too slow. Later, manipulations on the candidate word q also require $\mathcal{O}(nL)$ time in each call of the recursive procedure `ClosestStringRec()`, as q is checked against all words in $\mathcal{S}$.

We remedy these problems by introducing a data structure that maintains the dictionary $\mathcal{S}$ and provides access to all operations needed in the Algorithm 1, including efficient manipulation of the candidate q. This data structure is described in the following lemma.

▶ **Lemma 6** (Far word data structure). *There exists a randomized data structure that maintains the dictionary $\mathcal{S}$ of n words in Σ^L with amortized $2^{\mathcal{O}(d)}$ time updates; the initialization time is $2^{\mathcal{O}(d)} nL|\Sigma|$. The data structure provides the following method:*

- `QueryFarPair()`*: Decide if there exist $s, s' \in \mathcal{S}$ with $\text{dist}(s, s') > 2d$. The query may also return a positive answer in case there are no s, s' as above, but then it is guaranteed that the answer to the instance $(\mathcal{S}, d)$ of Closest String is negative.*

Further, the data structure provides access to a auxiliary word $q \in \Sigma^L$ through the following methods:

[2] With clever optimizations, one can decrease the running time to be $\mathcal{O}((d+1)^d \text{poly}(n, L))$ [16, Section 3.5].

- `Reset`(): *Reset q to the first word in $\mathcal{S}$.*
- `UpdateCandidate`(i, a)*: Change the ith position of q to symbol a.*
- `QueryFarWord`(): *Query if there exists $s \in \mathcal{S}$ with $\operatorname{dist}(s, q) > d$, and if so, return the pointer to s and the set of positions where s and q differ.*

Usage of the above requires the following promises:

- *Usage of* `Reset`() *must be preceded by obtaining a negative answer to* `QueryFarPair`().
- *Following resetting q to $s \in \mathcal{S}$ through usage of* `Reset`(), *the user has to guarantee that the assertion $\operatorname{dist}(q, s) \leqslant d$ will hold at all times till the next usage of* `Reset`().
- *Every update to any word in $\mathcal{S}$ resets q to be undefined, so that* `Reset`() *needs to be invoked again to enable operations on q.*

Methods `QueryFarPair`(), `Reset`(), `UpdateCandidate`(), `QueryFarWord`() *work in worst-case time $2^{\mathcal{O}(d)}$. Queries* `QueryFarPair`() *and* `QueryFarWord`() *return a false negative with probability $2^{-\Omega(d)}$; there are no false positives.*

A few remarks are in place regarding the use of randomness in the data structure of Lemma 6. Namely, random bits are used solely in the initialization of the data structure, and the correctness of subsequent uses of query methods depends on those initial random bits. Hence, the events when algorithm returns correct answers are *not* independent. This means that the error probability cannot be improved in the standard way by repeating each query many times. Instead, one can improve the error probability by setting up multiple independent copies of the data structure of Lemma 6.

With Lemma 6 stated, we can show how to derive Theorem 5 from it.

Proof of Theorem 5 assuming Lemma 6. We initialize and maintain $\alpha \log d$ independent copies of the data structure provided by Lemma 6 for some large enough constant α, to be determined later. Each update and each query is accordingly relayed to all these data structures; the output of a query is the disjunction of outputs provided by the individual data structures. In this way, we may assume that we have one instance of the data structure of Lemma 6 where the probability of a false negative is reduced to $(2^{-\Omega(d)})^{\alpha \log d} = (d^{-\Omega(d)})^{\alpha}$. The cost for this is that the running times of all methods are increased by a multiplicative factor of $\mathcal{O}(\log d)$; this will be immaterial in the forthcoming complexity analysis.

It remains to implement the query: we look for a word c that is at Hamming distance at most d from all the words in $\mathcal{S}$. The idea is to run Algorithm 1 with all operations replaced by suitable invocations of methods of the data structure of Lemma 6. Lines 1 and 3 are replaced by invocations of methods `QueryFarPair`() and `Reset`(), respectively. In line 7, we invoke method `QueryFarWord`(). Finally, in line 11 we use one `UpdateCandidate`() operation before recursing, and we roll-back this update (using the `UpdateCandidate`() method again) when returning from the recursion. The running time and the correctness (assuming no false negatives from the data structure of Lemma 6) follow from the correctness of the original static algorithm and Lemma 6. It is also straightforward to verify that the assumptions of Lemma 6 hold.

It remains to bound the probability of a false positive. Clearly, a false positive might arise only if some invocation of a method of the data structure of Lemma 6 returns a false negative. Since the recursion tree of procedure `ClosestString`() has depth at most d and branching at most $3d$, it has at most $2(3d)^d$ nodes, hence in total there are at most $1 + 2(3d)^d$ invocations of methods of the data structure of Lemma 6. By setting α large enough, we have $(1 + 2(3d)^d) \cdot (d^{-\Omega(d)})^{\alpha} \leqslant 2^{-\Omega(d)}$. So by the union bound, the probability of an error is bounded by $2^{-\Omega(d)}$. ◀

Now, we discuss the technical ideas behind the proof of Lemma 6. The key idea is that we can efficiently maintain an approximate solution, as explained in the following lemma.

▶ **Lemma 7** (Approximate Closest String). *There exists a data structure that maintains a dictionary $\mathcal{S}$ of words in Σ^L with amortized update time $\mathcal{O}(|\Sigma|)$, as well as a word $o \in \Sigma^L$ with the following guarantee: if the answer to the* Closest String *instance $(\mathcal{S}, d)$ is positive, then $\operatorname{dist}(o, s) \leqslant 4|\Sigma| \cdot d$ for every $s \in \mathcal{S}$. The data structure can be initialized in $\mathcal{O}(nL)$ time.*

Moreover, the data structure also maintains a set $\Delta(o, s) \coloneqq \{i \in [L] \mid o[i] \neq s[i]\}$ for every $s \in \mathcal{S}$ and, upon request, can return $\Delta(o, s)$ in time $\mathcal{O}(|\Delta(o, s)|)$. Finally, the data structure can check whether $\operatorname{dist}(o, s) \leqslant 4|\Sigma| \cdot d$ for all $s \in \mathcal{S}$ in time $\mathcal{O}(1)$.

In Section 3.2 we prove Lemma 7. Next, in Section 3.3 we use an approach based on color coding to leverage Lemma 7 to maintain a dictionary $\mathcal{S}$ and implement query `QueryFarPair`(). Adding the functionality concerning the candidate word q uses similar arguments and is presented in Section 3.4. Looking at the statement of Lemma 7, the reader might be at this point worried that this plan involves complexities dependent also on $|\Sigma|$. However, in Section 3.3 we will show how to reduce $|\Sigma|$ to $\mathcal{O}(d)$ using color coding.

3.2 Approximate Closest String

In this section, we prove Lemma 7. The main idea is to define $o \in \Sigma^L$ through an approximate majority vote for every position, maintained in a lazy fashion. We formalize this through the following definition.

▶ **Definition 8** (Origin Word). *An* origin word *for a dictionary $\mathcal{S}$ of words in Σ^L is a word $o \in \Sigma^L$ such that*

$$|\{s \in \mathcal{S} \mid s[i] = o[i]\}| \geqslant \frac{|\mathcal{S}|}{2|\Sigma|} \text{ for every } i \in [L].$$

We say that the origin word o is good *if $\operatorname{dist}(o, s) \leqslant 4|\Sigma| \cdot d$ for every $s \in \mathcal{S}$.*

By definition, if an origin word is good, then it is a solution for the Closest String instance $(\mathcal{S}, 4|\Sigma|d)$. We now show a reverse "soundness" implication: if some origin word is not good, then for sure there is no solution for $(\mathcal{S}, d)$.

▶ **Lemma 9.** *If for an instance $(\mathcal{S}, d)$ there exists an origin word that is not good, then the answer to $(\mathcal{S}, d)$ is negative.*

Proof. For the sake of contradiction, let us assume that there exists $c \in \Sigma^L$ such that $\operatorname{dist}(c, s) \leqslant d$ for every $s \in \mathcal{S}$. Moreover, there exists some origin word $o \in \Sigma^L$ and a witness $w \in \mathcal{S}$ such that $\operatorname{dist}(o, w) > 4d|\Sigma|$.

Let $C_{o,w}$ be the total count of matches between o and all words in $\mathcal{S}$ at the positions where o and w differ. That is,

$$C_{o,w} \coloneqq |\{(i, s) \in [L] \times \mathcal{S} \text{ such that } w[i] \neq o[i] \text{ and } s[i] = o[i]\}|.$$

Let us show a lower bound on $C_{o,w}$. Observe that for a witness $w \in \mathcal{S}$ there are at least $\operatorname{dist}(o, w)$ positions i that are taken into account when computing $C_{o,w}$. Moreover, by the definition of origin word o, for every position $i \in [L]$ at least $|\mathcal{S}|/(2|\Sigma|)$ words match o on position i. Therefore,

$$\frac{|\mathcal{S}| \cdot \operatorname{dist}(o, w)}{2|\Sigma|} \leqslant C_{o,w}. \tag{2}$$

On the other hand, we assumed that there exists $c \in \Sigma^L$ such that $\mathrm{dist}(c, s) \leqslant d$ for every $s \in \mathcal{S}$. Since $w \in \mathcal{S}$, by triangle inequality we have $\mathrm{dist}(s, w) \leqslant 2d$ for every $s \in \mathcal{S}$. Hence

$$C_{o,w} \leqslant 2d \cdot |\mathcal{S}|. \tag{3}$$

By combining Inequalities (2) and (3) we conclude that $\mathrm{dist}(o, w) \leqslant 4|\Sigma|d$, a contradiction. ◀

Next, we argue that in $\mathcal{O}(|\Sigma|)$ time we can maintain some origin word for a given dictionary.

▶ **Lemma 10.** *In $\mathcal{O}(nL)$ time we can initialize a data structure that for a given dictionary $\mathcal{S}$ of words in Σ^L maintains some origin word $o \in \Sigma^L$ with amortized update time $\mathcal{O}(|\Sigma|)$. The data structure also maintains the set $\Delta(o, s) \coloneqq \{i \in [L] \mid s[i] \neq o[i]\}$ for every $s \in \mathcal{S}$ and upon request, can return each set $\Delta(o, s)$ in time $\mathcal{O}(|\Delta(o, s)|)$. Finally, the data structure can check whether o is good in time $\mathcal{O}(1)$.*

Proof. Upon initialization, we set $o \in \Sigma^L$ so that for every position $i \in [L]$, $o[i]$ is a symbol that occurs the most often among $s[i]$ for all $s \in \mathcal{S}$. Clearly, o constructed in this way is an origin word. We also compute the relevant sets $\Delta(o, s)$.

The data structure stores the following additional data. For every position $i \in [L]$ and every symbol $\beta \in \Sigma$, we maintain a counter indicating the number of words $s \in \mathcal{S}$ such that $s[i] = \beta$. Each set $\Delta(o, s)$ is stored as a linked list (with no assumption on the order), plus there is an array of length L whose ith entry is either null if $i \notin \Delta(o, s)$, or contains a pointer to the relevant object on the linked list representing $\Delta(o, s)$. Additionally, with each set $\Delta(o, s)$, we maintain its size. Additionally, we store a single counter indicating the number of words $s \in \mathcal{S}$ such that $|\Delta(o, s)| \geqslant 4|\Sigma|d$. This counter can be used to answer queries about the goodness of o in time $\mathcal{O}(1)$. Upon initialization, all of the above can be computed in time $\mathcal{O}(nL)$ in a straightforward way.

We now explain how the data structure behaves upon an update. Suppose position $s_j[i]$ is modified. We update the relevant counters for position i and update $\Delta(o, s_j)$ accordingly. Next, we check whether the counter for the symbol $o[i]$ at position i did not drop below $|\mathcal{S}|/(2|\Sigma|)$. If not, then o remains an origin word and there is no need to change o. Otherwise, we modify $o[i]$ as follows.

By iterating through all words in $\mathcal{S}$, we compute the most frequent symbol among $s[i]$ for $s \in \mathcal{S}$, and we set $o[i]$ to be this symbol. Moreover, we iterate over all $s \in \mathcal{S}$ and update $\Delta(o, s)$ accordingly, by adding or removing the position i if needed. These operations require total time $\mathcal{O}(|\mathcal{S}|)$.

We now argue that the amortized update time is $\mathcal{O}(|\Sigma|)$. By the pigeon-hole principle, when symbol $o[i]$ gets modified, it is replaced by the most frequent symbol that occurs at least $|\mathcal{S}|/|\Sigma|$ times on position i in words from $\mathcal{S}$. Also, this is true for the symbol placed as $o[i]$ upon initialization. Therefore, before an update triggers a change of $o[i]$, there were at least $|\mathcal{S}|/(2|\Sigma|)$ updates on position i. We can charge the running time $\mathcal{O}(|\mathcal{S}|)$ used when modifying $o[i]$ to those previous updates, thus obtaining amortized update time $\mathcal{O}(|\Sigma|)$. ◀

Now Lemma 7 follows by combining Lemmas 10 and 9.

3.3 Detecting dissimilar words

In this section, we present a data structure, that maintains the dictionary and implements the method `QueryFarPair`(), and for now we ignore the methods for handling q.

In the data structure, we will maintain hashes of all words in $\mathcal{S}$ to the binary alphabet. More precisely, upon initialization of the data structure, we uniformly at random sample a function $h\colon [L] \times \Sigma \to \{0,1\}$ that assigns a label 0 or 1 to every position and symbol in the alphabet. This function is fixed for the whole life of the data structure and stored in it. In notation, we shall use a natural lift of $h\colon \Sigma^L \to \{0,1\}^L$ that applies h position-wise. In the data structure we store, together with $\mathcal{S}$, the hashed dictionary $\widetilde{\mathcal{S}} \coloneqq \{h(s) \mid s \in \mathcal{S}\}$. Observe that upon every update to $\mathcal{S}$ we can also update $\widetilde{\mathcal{S}}$ in constant time.

We also maintain an approximate solution $\tilde{o} \in \{0,1\}^L$ for dictionary $\widetilde{\mathcal{S}}$ using the data structure of Lemma 7. Recall that we can query the data structure of Lemma 7 about whether $\mathrm{dist}(\tilde{o}, \tilde{s}) \leqslant 8d$ for all $\tilde{s} \in \widetilde{\mathcal{S}}$ and if this is not the case, then we know for sure that the instance $(\widetilde{\mathcal{S}}, d)$ of Closest String has a negative answer. Note that this conclusion implies that the original instance $(\mathcal{S}, d)$ also has a negative answer.

In addition to the approximate solution $\tilde{o}$, the data structure of Lemma 7 provides an access to the sets $\Delta(\tilde{o}, \tilde{s})$ of positions where $\tilde{o}$ and $\tilde{s}$ differ, for all $\tilde{s} \in \widetilde{\mathcal{S}}$.

Finally, we also hash positions as follows. Upon initialization, we sample uniformly at random a function $\pi\colon [L] \to [16d]$ which maps positions to a set of $16d$ colors (numbers from 1 to $16d$). Again, this function is fixed for the whole life of the data structure and stored in it. For a word $\tilde{s} \in \widetilde{\mathcal{S}}$, let $\mathsf{colors}_{\tilde{o},\pi}(s) = \{\pi(i) \mid i \in [L] \text{ and } s[i] \neq \tilde{o}[i]\}$ be the set of colors assigned to the symbols in $\tilde{s}$ that are on positions where $\tilde{s}$ does not match the origin word $\tilde{o} \in \{0,1\}^L$.

In the data structure we maintain, for every $C \subseteq [16d]$, the set $\Phi(C)$ defined as follows:

$$\Phi(C) = \{\tilde{s} \in \widetilde{\mathcal{S}} \mid \mathsf{colors}_{\tilde{o},\pi}(\tilde{s}) = C\}.$$

In other words, $\Phi(C)$ is the set of words from $\widetilde{\mathcal{S}}$ that get assigned the color set C. The next statement shows that the sets $\Phi(C)$ can be maintained in $2^{\mathcal{O}(d)}$ time per update.

▶ **Lemma 11.** *We can initialize in $2^{\mathcal{O}(d)} \cdot nL$ time a data structure that for every $C \subseteq [16d]$ maintains the set $\Phi(C)$ in amortized $2^{\mathcal{O}(d)}$ time per update to $\mathcal{S}$. When queried about any $C \subseteq [16d]$, the data structure in $\mathcal{O}(1)$ time either returns any element from $\Phi(C)$, or asserts that $\Phi(C)$ is empty.*

The proof of Lemma 11 is deferred to Appendix B. It is rather technical and builds on the data structure of Lemma 7 by additionally storing sets $\Phi(C)$ as doubly-linked lists. Every modification to $\widetilde{\mathcal{S}}$ and o triggers a number of modifications to lists representing $\Phi(C)$, consisting of moving some elements from one list to another. The same amortization argument as the one used in the proof of Lemma 7 shows that the amortized update time is $2^{\mathcal{O}(d)}$.

Algorithm 2 Pseudocode for the method `QueryFarPair`().

```
Method: QueryFarPair()
if exists s̃ ∈ S̃ with dist(õ, s̃) > 8d then
    return Answer to (S, d) is negative.
for every X, Y ⊆ [16d] with |X △ Y| > 2d do
    if Φ(X) and Φ(Y) are both nonempty then
        return True        // There are words s, s' ∈ S with dist(s, s') > 2d.
return False               // It holds that dist(s, s') ⩽ 2d for all s, s' ∈ S.
```

We now present an implementation of the query operation; see Algorithm 2 for a pseudocode. We first check whether $\mathrm{dist}(\tilde{o}, \tilde{s}) \leqslant 8d$ for all $\tilde{s} \in \widetilde{\mathcal{S}}$. As argued in Lemma 9, if this is not the case, then we can safely conclude that the answer to the instance $(\mathcal{S}, d)$

is negative. Otherwise, we iterate over every pair of sets $X, Y \subseteq [16d]$ with $|X \triangle Y| = |(X \setminus Y) \cup (Y \setminus X)| > 2d$. Then, we check whether both $\Phi(X)$ and $\Phi(Y)$ are nonempty. If that is the case, then (as we will argue) any pair $(\tilde{s}, \tilde{s}') \in \Phi(X) \times \Phi(Y)$ satisfies $\mathrm{dist}(\tilde{s}, \tilde{s}') > 2d$, implying that the original words $s, s' \in \mathcal{S}$ also satisfy $\mathrm{dist}(s, s') > 2d$. Otherwise, if for every such X and Y at least one of $\Phi(X)$ or $\Phi(Y)$ is empty, we conclude that there is no pair $s, s' \in \mathcal{S}$ with $\mathrm{dist}(s, s') > 2d$.

Because the number of pairs $X, Y \subseteq [16d]$ is $2^{\mathcal{O}(d)}$, the query algorithm runs in $2^{\mathcal{O}(d)}$ time in total. The next lemma shows that if the algorithm finds some pair of words and reports that they are at distance larger than $2d$, then this answer is correct.

▶ **Lemma 12.** *Suppose Algorithm 2 finds a pair $X, Y \subseteq [16d]$ with $|X \triangle Y| > 2d$ and $\Phi(X) \neq \emptyset$ and $\Phi(Y) \neq \emptyset$. Then there are $s, s' \in \mathcal{S}$ such that $\mathrm{dist}(s, s') > 2d$.*

Proof. Consider any pair $(\tilde{s}, \tilde{s}') \in \Phi(X) \times \Phi(Y)$. Observe that for every color $r \in X \setminus Y$, there is a position i with $\pi(i) = r$ such that $\tilde{s}[i] \neq \tilde{o}[i]$ (due to $r \in X$), and we have $\tilde{o}[i] = \tilde{s}'[i]$ (due to $r \notin Y$). So $\tilde{s}[i] \neq \tilde{s}'[i]$, implying $s[i] \neq s'[i]$. Similar statements hold for every $r \in Y \setminus X$. Positions i as above have to be pairwise different due to receiving different colors in π. Therefore, because the number of such positions is more than $2d$, we conclude that s and s' differ on more than $2d$ positions. ◀

To finish the proof, it remains to analyze the success probability of Algorithm 2.

▶ **Lemma 13.** *If there exists a pair $a, b \in \mathcal{S}$ with $\mathrm{dist}(a, b) > 2d$, then Algorithm 2 detects such a pair with the probability at least $2^{-\mathcal{O}(d)}$, or concludes that the answer to the instance $(\mathcal{S}, d)$ is negative.*

Note that in Lemma 6 we promised error probability bounded by $2^{-\Omega(d)}$, while Lemma 13 provides a bound of $1 - 2^{-\mathcal{O}(d)}$ on the error probability. This can be easily remedied by maintaining $2^{\Theta(d)}$ independent copies of the data structure. This increases the time of update and initialization by a $2^{\mathcal{O}(d)}$ factor.

Proof of Lemma 13. Let $a, b \in \mathcal{S}$ be a pair of words in $\mathcal{S}$ with $\mathrm{dist}(a, b) > 2d$. First, we argue that after hashing the alphabet, we still have $\mathrm{dist}(h(a), h(b)) > 2d$ with sufficiently high probability. Let $P \subseteq \{i \in [L] \mid a[i] \neq b[i]\}$ be any set of size exactly $2d + 1$ consisting of positions where a and b differ.

Let $\tilde{a} = h(a)$ and $\tilde{b} = h(b)$ for First, we claim that with probability at least $2^{-\mathcal{O}(d)}$ it holds that $\mathrm{dist}(\tilde{a}, \tilde{b}) > 2d$. Observe that for a fixed position $i \in P$, the probability that h assigns different symbols to $a[i]$ and to $b[i]$ is $1/2$. Since h is sampled on each position $i \in [L]$ independently, the probability that this happens for all positions in P is $2^{-|P|} = 2^{-\mathcal{O}(d)}$.

From now on, let us assume that $\mathrm{dist}(\tilde{a}, \tilde{b}) > 2d$. Moreover, by Line 1 we may assume that $\mathrm{dist}(\tilde{o}, \tilde{s}) \leqslant 8d$ for every $\tilde{s} \in \widetilde{\mathcal{S}}$. Consider the set

$$\Delta_{\tilde{o}}(\tilde{a}, \tilde{b}) \coloneqq \{i \in [L] \mid \tilde{a}[i] \neq \tilde{o}[i] \text{ or } \tilde{b}[i] \neq \tilde{o}[i]\}.$$

Observe that since $\mathrm{dist}(\tilde{o}, \tilde{a}) \leqslant 8d$ and $\mathrm{dist}(\tilde{o}, \tilde{b}) \leqslant 8d$, we have $k \coloneqq |\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})| \in [2d + 1, 16d]$. Now, we claim that with probability $2^{-\mathcal{O}(d)}$ the function π assigns different colors to all positions in $\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})$. There are $(16d)^k$ different colorings on $\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})$. However, only $\binom{16d}{k} k!$ of them assign different colors to $\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})$. Therefore the probability that π assigns different colors on $\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})$ is:

$$\Pr\left[|\{\pi(i) \mid i \in \Delta_{\tilde{o}}(\tilde{a}, \tilde{b})\}| = k\right] = \frac{\binom{16d}{k} k!}{(16d)^k} = \frac{(16d)}{(16d)} \cdots \frac{(16d - k + 1)}{(16d)} > \frac{(16d)!}{(16d)^{16d}} > e^{-16d}$$

where the last inequality follows from the well-known bound $n! > (n/e)^n$. Hence, the probability that π assigns different colors to all positions in $\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})$ is $2^{-\mathcal{O}(d)}$. Now, we claim that if that indeed happens, then Algorithm 2 detects a suitable pair.

Let X and Y be sets of colors such that $\tilde{a} \in \Phi(X)$ and $\tilde{b} \in \Phi(Y)$. It suffices to show that $|X \triangle Y| > 2d$. Observe that every position where $\tilde{a}$ and $\tilde{b}$ differ belongs to $\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})$, hence these (more than $2d$) positions receive different colors in π. Further, for every position where $\tilde{a}$ and $\tilde{b}$ differ, the color of this position belongs to $X \triangle Y$, for outside of positions of $\Delta_{\tilde{o}}(\tilde{a}, \tilde{b})$ the words $\tilde{o}, \tilde{a}, \tilde{b}$ all agree. It follows that $|X \triangle Y| > 2d$. ◀

3.4 Maintaining a candidate solution

In this section, we finish the proof of Lemma 6 by implementing the operations on the candidate word q. This proof builds upon the construction from Section 3.3 using the same ideas, so we only briefly discuss the additional elements that need to be maintained.

Observe that q is always reset to the first word $s_1 \in \mathcal{S}$ and operations on q are performed under the promise that $\mathrm{dist}(q, s_1) \leqslant d$ at all times. Therefore, we maintain q implicitly by remembering only at most d positions on which s_1 and q differ, and what are the symbols of q on those positions. This allows us to implement the reset and update operations for q in time $2^{\mathcal{O}(d)}$. (Recall here that in Section 3.3 we in fact maintained $2^{\mathcal{O}(d)}$ independent copies of the data structure in order to boost the error probability.) Also, we maintain the hashed version $\tilde{q} := h(q)$.

The method `QueryFarWord()` is implemented using a similar mechanism as was used in Section 3.3. For technical reasons, we extend the palette of colors used by π from $[16d]$ to $[17d]$. Then we maintain the sets $\Phi(C) \subseteq \widetilde{\mathcal{S}}$ for $C \subseteq [17d]$ as before. However, instead of iterating over all pairs $X, Y \subseteq [17d]$ with $|X \triangle Y| > 2d$, we first compute $Q := \mathsf{colors}_{\tilde{o},\pi}(\tilde{q})$ and then iterate over all $X \subseteq [17d]$ such that $|X \triangle Q| > d$ and check whether $\Phi(X)$ is nonempty. The same reasoning as in Section 3.3 shows that if there exists $s \in \mathcal{S}$ with $\mathrm{dist}(q, s) > d$, then with high enough probability we will find such an s as any element of $\Phi(X)$.

Note that in the description above we did not specify how the set $\mathsf{colors}_{\tilde{o},\pi}(\tilde{q})$ is computed. This can be done by first obtaining the set $\Delta(\tilde{o}, \tilde{s}_1)$ from the data structure of Lemma 7, and then inspecting all the positions of $\Delta(\tilde{o}, \tilde{s}_1) \cup \Delta(s_1, q)$, where $\Delta(s_1, q)$ are the at most d positions where s_1 and q differ. Note here that we may assume that $|\Delta(\tilde{o}, \tilde{s}_1)| \leqslant 8d$, for otherwise the data structure presented in Section 3.3 must have returned that the answer to $(\mathcal{S}, d)$ is negative when resetting q. Further, this reasoning shows that $|\Delta(\tilde{o}, \tilde{q})| \leqslant 9d$ at all times. Note that in the correctness argument presented in Section 3.3, we used the assumption that $\mathrm{dist}(\tilde{o}, \tilde{a}) \leqslant 8d$ and $\mathrm{dist}(\tilde{o}, \tilde{b}) \leqslant 8d$, and this is why we chose a palette of colors of size $16d$. Now, we have $\mathrm{dist}(\tilde{o}, \tilde{q}) \leqslant 9d$ and $\mathrm{dist}(\tilde{o}, \tilde{s}) \leqslant 8d$, so a palette of $17d$ colors suffices.

It remains to argue that if s with $\mathrm{dist}(q, s) > d$ is found, the set P of positions on which q and s differ can be reported in time $\mathcal{O}(d)$. But again, P can be constructed by inspecting all positions of $\Delta(\tilde{o}, \tilde{s}_1) \cup \Delta(q, s_1)$, and this set has size $\mathcal{O}(d)$ and can be obtained by a query to the data structure of Lemma 7. This finishes the proof of Lemma 6.

4 Closest String for small alphabets

In this section we analyse the complexity of Closest String for small alphabets and show that our techniques also apply in this setting. That is, we prove the second half of Theorem 1, presented below.

▶ **Theorem 14.** *The dynamic variant of* Closest String *admits a randomized data structure with initialization time* $2^{\mathcal{O}(d)}nL|\Sigma|^{1+o(1)}$*, amortized update time* $2^{\mathcal{O}(d)}$*, and worst-case query time* $(|\Sigma|-1)^d 2^{\mathcal{O}(d)}$*. The answer to each query may result with a false negative with probability at most* $2^{-\Omega(d)}$*; there are no false positives.*

The strategy is exactly the same as in Section 3. First, we present a static algorithm with running time $(|\Sigma|-1)^d \cdot 2^{\mathcal{O}(d)} \cdot (nL)^{\mathcal{O}(1)}$, which is essentially the algorithm proposed by Ma and Sun [22]. Next, we show how to use Lemma 6 to implement this static algorithm to the dynamic setting. The algorithm is presented using pseudocode as Algorithm 3. We present it somewhat differently than Ma and Sun in order to streamline the analysis of the dynamic variant.

Algorithm 3 Pseudocode of a $(|\Sigma|-1)^d \cdot 2^{\mathcal{O}(d)} \cdot (nL)^{\mathcal{O}(1)}$-time static algorithm for Closest String. To get a dynamic data structure with query time $(|\Sigma|-1) \cdot 2^{\mathcal{O}(d)}$, use Lemma 6 for operations on q.

Algorithm : `ClosestStringSmallAlphabet`$(\mathcal{S}, d)$

Set $F := \emptyset$
Set q to be the first word $s_1 \in \mathcal{S}$
Set $b := d$
while *exists* $s \in \mathcal{S}$ *such that* $\mathrm{dist}(s,q) > d$ **do**
  Find $P := \{i \in [L] \text{ such that } s[i] \neq q[i]\} \setminus F$
  if $\mathrm{dist}(s,q) > 2d$ *or* $P = \emptyset$ **then**
    return False
  Guess $Q = \{i \in P \text{ such that } c[i] \neq q[i]\}$ `// c ∈ Σ^L denotes the sought solution`
  if $|Q| > b$ *or* $Q = \emptyset$ **then**
    return False
  for $i \in Q$ **do**
    Guess $c[i] \in \Sigma \setminus \{q[i]\}$
    Set $q[i] := c[i]$
  $F := F \cup P$
  $b := \min(d - \mathrm{dist}(s,q), b - |Q|)$
  if $b < 0$ **then**
    return False
return True

The algorithm maintains three global values. The first one is a set $F \subseteq [L]$ of *fixed indices*. The second one is a word $q \in \Sigma^L$ that is a candidate for the solution, which at the start is set to be any word from $\mathcal{S}$. The third one is a *budget* $b \in \mathbb{N}$, initially set to d.

We imagine the algorithm as a nondeterministic procedure that, having in mind some solution $c \in \Sigma^L$, guesses parts of c along the execution and appropriately modifies q. The set F is used to keep track of the positions that are already assumed to be fixed as in F. As usual, nondeterminism is determined by branching over all possibilities, and the total number of branches determines the running time of the algorithm. At every point, even in branches where guesses were inconsistent with c, the algorithm maintains the following invariant:

($\diamondsuit$) There is $s \in \mathcal{S}$ such that $s[\overline{F}] = q[\overline{F}]$ and $b = d - \mathrm{dist}(q[F], s[F])$, where we denote $\overline{F} = [L] \setminus F$.

In this way, one may think of b as of the budget that is left for changing symbols positions in q outside of F: at most b of them can be still changed, for otherwise the solution would be too far from s.

Every step of the algorithm works as follows. First, we find a word $s \in \mathcal{S}$ with $\mathrm{dist}(s, q) > d$. If no such word exists, then the current candidate q is a solution and we can terminate the procedure claiming a positive answer. Otherwise, we compute the set P of positions where s and q differ, and we remove from it all positions that were fixed before.

Next, we check whether $\mathrm{dist}(s, q) > 2d$, which translates to the condition $\mathrm{dist}(s[F \cup P], q[F \cup P]) > 2d$. If this is the case, we terminate and provide a negative answer: there is no way to obtain a word at distance at most d from s by changing at most d positions in q. If $P = \emptyset$, we can also terminate and provide a negative answer: already on fixed positions, our candidate q and s differ by more than d. Otherwise, when $\mathrm{dist}(s, q) \leqslant 2d$ and $P \neq \emptyset$, we guess exactly the symbols in c at positions from P and we modify q to have $q[P] = c[P]$. This is done through a two-stage process: first we guess the set of positions $Q \subseteq P$ where q needs to be modified, and then we guess the symbols of c at positions of Q; for each there are $|\Sigma| - 1$ possibilities. Note that we may restrict attention to sets Q that are nonempty (for $\mathrm{dist}(s, q) > d$) and satisfy $|Q| \leqslant b$ (by invariant 18). Finally, we add P to the set F of fixed indices and update b to the minimum of the two values: $d - \mathrm{dist}(s, q)$ and $b - |Q|$. It is straightforward to verify that this way invariant 18 is still maintained: either s or the previous witness for 18 may serve as the new witness for 18. Clearly, if b became negative, it is safe to terminate the branch. Otherwise we continue the search until a candidate q at distance at most d from all strings in $\mathcal{S}$ is found.

This concludes the description of the algorithm. The correctness is clear from the description as we return True only if our candidate is at the distance at most d from all input strings.

It is now straightforward to turn this algorithm into a dynamic data structure just as we did in the proof of Theorem 5. Namely, we maintain the data structure of Lemma 6, and use it to operate on the candidate word q. All distance checks can be implemented in linear time by verifying the $\mathcal{O}(d)$-sized difference sets provided by this data structure. We will later show that the whole recursion tree of Algorithm 3 has total size at most $(|\Sigma| - 1)^d \cdot 2^{\mathcal{O}(d)}$. Hence, as the operations in the data structure of Lemma 6 take amortized time $2^{\mathcal{O}(d)}$, the complexity guarantees promised in Theorem 14 follow in the same way as it was the case for Theorem 5. As for the error probability, we can maintain $\alpha \cdot \log |\Sigma|$ independent copies of the data structure of Lemma 6 for some large constant α, so that the probability that this composite data structure returns a false negative is reduced to $(|\Sigma|^{-\Omega(d)})^\alpha$. Then, just as in the proof of Theorem 5, it follows from the union bound that the probability of a false negative in Algorithm 3 is at most $2^{-\Omega(d)}$.

We are left with bounding the running time of Algorithm 3, or more precisely, showing that the whole recursion tree has size at most $(|\Sigma| - 1)^d \cdot 2^{\mathcal{O}(d)}$. The argument conceptually follows the reasoning of Ma and Sun [29]; we present it for completeness.

Runtime. The key observation is the following lemma.

▶ **Lemma 15.** *Consider ith iteration of the while loop in Algorithm 3 (with any guesses made). Let b_i be the value of b before this iteration, and b_{i+1} be the value of b after this iteration. Then $b_{i+1} \leqslant b_i/2$.*

We first argue that the claimed runtime of Algorithm 3 follows from Lemma 15. Consider any root-to-leaf path in the recursion tree of the algorithm; this corresponds to a single run of Algorithm 3 treated as a nondeterministic procedure, with some guesses made along the

way. For iterations $i = 1, 2, \ldots, p$ of the while-loop, where p is the total number of iterations made, let b_i be the value of b at the beginning of the ith iteration, and let ℓ_i be the size of the set Q considered in the ith iteration. Observe the following:

- We have $b_i \leqslant d/2^{i-1}$ for all $i \in [p]$ (because $b_1 = d$ and, by Lemma 15, $b_{i+1} \leqslant b_i/2$ for all $i \in [p-1]$).
- We have $1 \leqslant \ell_i \leqslant b_i$ for all $i \in [p]$ (because in the algorithm we consider only nonempty sets Q satisfying $|Q| \leqslant b$).
- We have $\sum_{i=1}^{p} \ell_i \leqslant d$ (because b decreases by at least ℓ_i in the ith iteration, and the procedure terminates once b becomes negative).

Therefore, every root-to-leaf path in the recursion tree can be uniquely described by specifying the following data:

(a) Positive integers $\ell_1, \ldots, \ell_p$ satisfying $\sum_{i=1}^{p} \ell_i \leqslant d$.
(b) For each $i \in [p]$, a choice of a subset Q_i of size ℓ_i of the set P_i, where P_i, Q_i are the sets P, Q considered in the ith iteration.
(c) For each $i \in [p]$, a choice of symbols guessed to be fixed at the positions of Q_i.

For a, it is well-known that the number of representations of d as a sum of numbers $\ell_1, \ldots, \ell_p$ is bounded by $2^{\mathcal{O}(d)}$. For c, the total number of choices is bounded by

$$\prod_{i=1}^{p} (|\Sigma| - 1)^{\ell_i} \leqslant (|\Sigma| - 1)^d.$$

Finally, for b we shall use the following known bound.

▶ **Lemma 16** (cf. Lemma 124 in [39]). *If m, k are nonnegative integers, then $\binom{m+k}{k} \leqslant 2^{2\sqrt{mk}}$.*

Since we always have $|P_i| \leqslant 2d$, the number of choices for b is bounded as follows:

$$\prod_{i=1}^{p} \binom{2d}{\ell_i} \leqslant \prod_{i=1}^{p} 2^{2\sqrt{2d\ell_i}} \leqslant \prod_{i=1}^{p} 2^{2\sqrt{2d \cdot d/2^{i-1}}} = 2^{2\sqrt{2} \cdot d \cdot \sum_{i=0}^{\infty} 2^{-i/2}} = 2^{\mathcal{O}(d)}.$$

So all in all, the total number of root-to-leaf paths in the recursion tree is bounded by

$$2^{\mathcal{O}(d)} \cdot 2^{\mathcal{O}(d)} \cdot (|\Sigma| - 1)^d = (|\Sigma| - 1)^d \cdot 2^{\mathcal{O}(d)},$$

as claimed. It remains to prove the Lemma 15.

Proof of Lemma 15. Let q_i and q_{i+1} be the candidate word respectively at the beginning and at the end of the ith iteration, that is, after guessing is performed. Recall that there is $s \in \mathcal{S}$ with $\mathrm{dist}(s, q_i) > d$. Further, recall that

$$b_{i+1} = \min(d - \mathrm{dist}(s, q_{i+1}), b_i - |Q|).$$

To prove that $b_{i+1} \leqslant b_i/2$, it suffices to show that

$$(d - \mathrm{dist}(s, q_{i+1})) + (b_i - |Q|) \leqslant b_i,$$

or equivalently,

$$d \leqslant \mathrm{dist}(s, q_{i+1}) + |Q|. \tag{4}$$

By triangle inequality we have

$$d < \mathrm{dist}(s, q_i) \leqslant \mathrm{dist}(s, q_{i+1}) + \mathrm{dist}(q_i, q_{i+1}),$$

but we also have

$$\mathrm{dist}(q_i, q_{i+1}) = |Q|.$$

So this establishes (4) and finishes the proof. ◀

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.

2 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. 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 375–388. ACM, 2016. doi:10.1145/2897518.2897653.

3 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic parameterized problems and algorithms. *ACM Trans. Algorithms*, 16(4):45:1–45:46, 2020. doi:10.1145/3395037.

4 Antoine Amarilli, Louis Jachiet, and Charles Paperman. Dynamic membership for regular languages. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 116:1–116:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.116.

5 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020. doi:10.1007/s00453-020-00744-0.

6 Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 990–1001. IEEE, 2020. doi:10.1109/FOCS46700.2020.00096.

7 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

8 Manu Basavaraju, Fahad Panolan, Ashutosh Rai, M. S. Ramanujan, and Saket Saurabh. On the kernelization complexity of string problems. *Theor. Comput. Sci.*, 730:21–31, 2018. doi:10.1016/j.tcs.2018.03.024.

9 Tugkan Batu, Funda Ergün, Joe Kilian, Avner Magen, Sofya Raskhodnikova, Ronitt Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 316–324. ACM, 2003. doi:10.1145/780542.780590.

10 Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. Syst. Sci.*, 65(1):38–72, 2002. doi:10.1006/jcss.2002.1822.

11 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011. doi:10.1016/j.tcs.2011.04.039.

12 Mikołaj Bojańczyk. *Languages recognised by finite semigroups and their generalisations to objects such as trees and graphs, with an emphasis on definability in monadic second-order logic.* In preparation, 2020. URL: https://www.mimuw.edu.pl/~bojan/papers/algebra-26-aug-2020.pdf.

13 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

14 Panagiotis Charalampopoulos. *Data Structures for Strings in the Internal and Dynamic Settings.* PhD thesis, King's College London, 2021.

15 Jiehua Chen, Wojciech Czerwiński, Yann Disser, Andreas Emil Feldmann, Danny Hermelin, Wojciech Nadara, Marcin Pilipczuk, Michał Pilipczuk, Manuel Sorge, Bartlomiej Wróblewski, and Anna Zych-Pawlewicz. Efficient fully dynamic elimination forests with applications to detecting long paths and cycles. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 796–809. SIAM, 2021. doi:10.1137/1.9781611976465.50.

16 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.

17 Zdeněk Dvořák, Martin Kupec, and Vojtěch Tůma. A dynamic data structure for MSO properties in graphs with bounded tree-depth. In *Proceedings of the 22th Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 334–345. Springer, 2014. doi:10.1007/978-3-662-44777-2_28.

18 Zdeněk Dvořák and Vojtěch Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *Proceedings of the 13th International Symposium on Algorithms and Data Structures, WADS 2013*, volume 8037 of *Lecture Notes in Computer Science*, pages 304–315. Springer, 2013. doi:10.1007/978-3-642-40104-6_27.

19 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.

20 Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. Syst. Sci.*, 47(3):424–436, 1993. doi:10.1016/0022-0000(93)90040-4.

21 Elazar Goldenberg, Tomasz Kociumaka, Robert Krauthgamer, and Barna Saha. Gap edit distance via non-adaptive queries: Simple and optimal. *CoRR*, abs/2111.12706, 2021. arXiv:2111.12706.

22 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.

23 Alejandro Grez, Filip Mazowiecki, Michał Pilipczuk, Gabriele Puppis, and Cristian Riveros. Dynamic data structures for timed automata acceptance. In *Proceedings of the 16th International Symposium on Parameterized and Exact Computation, IPEC 2021*, volume 214 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.IPEC.2021.20.

24 Heikki Hyyrö, Kazuyuki Narisawa, and Shunsuke Inenaga. Dynamic edit distance table under a general weighted cost function. *Journal of Discrete Algorithms*, 34:2–17, 2015. doi:10.1016/j.jda.2015.05.007.

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 Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998.

27 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

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

29 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009. doi:10.1137/080739069.

30 Konrad Majewski, Michał Pilipczuk, and Marek Sokołowski. Maintaining $CMSO_2$ properties on dynamic structures with bounded feedback vertex number. *CoRR*, abs/2107.06232, 2021. arXiv:2107.06232.

31 William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

32 Robert McNaughton and Seymour Papert. *Counter-free automata*. MIT Press, 1971.

33 Kurt Mehlhorn, Stefan Näher, and Helmut Alt. A lower bound on the complexity of the union-split-find problem. *SIAM J. Comput.*, 17(6):1093–1102, 1988. doi:10.1137/0217070.

34 Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

35 Gonzalo Navarro and Javiel Rojas-Ledesma. Predecessor search. *ACM Comput. Surv.*, 53(5):105:1–105:35, 2020. doi:10.1145/3409371.

36 Takaaki Nishimoto, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, et al. Fully dynamic data structure for lce queries in compressed space. *arXiv preprint arXiv:1605.01488*, 2016.

37 Jędrzej Olkowski, Michał Pilipczuk, Mateusz Rychlicki, Karol Węgrzycki, and Anna Zych-Pawlewicz. Dynamic data structures for parameterized string problems. *CoRR*, abs/2205.00441, 2022. doi:10.48550/arXiv.2205.00441.

38 Mihai Patrascu and Mikkel Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 166–175. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.26.

39 Michał Pilipczuk. *Tournaments and optimality: New results in parameterized complexity.* PhD thesis, University of Bergen, 2013.

40 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control.*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

41 Larry J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic.* PhD thesis, MIT, 1974.

42 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Process. Lett.*, 6(3):80–82, 1977. doi:10.1016/0020-0190(77)90031-X.

A Applications of Meta-Theorems

In this section we first state a meta-theorem for string problems definable in first-order logic FO; this result follows easily from the work of Frandsen et al. [19] using the classic Schützenberger-McNaughton-Papert theorem [32, 40]. Then we explain how to use the meta-theorem for the following toy problems: Disjoint Factors and Edit Distance. As usual with meta-theorems, the parametric factor in the complexity guarantees of the obtained data structures is not explicit, and typically is much higher than if one constructs the data structure "by hand". Therefore, we next show how to derive concrete data structures with concrete complexity guarantees for Disjoint Factors and Edit Distance. Finally, we discuss a methodology for lower bounds introduced by Amarilli et al. [4], and we apply it to derive lower bounds for those two problems.

A.1 A meta-theorem

We first need to recall basic knowledge on different equivalent views on regular languages. This material is standard in the area of algebraic theory of languages, so we refer an interested reader to the book of Bojańczyk [12] for a broader introduction. In particular, we explain the contemporary understanding of the material, and for appropriate references and historical remarks, we refer to [12].

The first view is through the lens of logic. Fix a finite alphabet Σ. We consider the logic $\mathsf{MSO}[\Sigma, <]$ operating on words. In this logic there are variables for single positions (denoted with small letters) and subsets of positions (denoted with capital letters). The atomic formulas are of the following form:

- equality test $x = y$;
- test $x \in X$ checking that position x belongs to position subset X;

- for every $a \in \Sigma$, test $a(x)$ checking that at position x there is symbol a; and
- test $x < y$ checking that position x appears before position y.

Formulas of $\mathsf{MSO}[\Sigma, <]$ can be obtained from atomic formulas using standard boolean connectives and quantification (both universal and existential, and applicable to both types of variables). $\mathsf{FO}[\Sigma, <]$ is a fragment of $\mathsf{MSO}[\Sigma, <]$ where we disallow variables for subsets of positions.

A *sentence* is a formula without free variables. By $w \models \varphi$ we mean that the sentence φ is satisfied in the word w. For a sentence $\varphi \in \mathsf{MSO}[\Sigma, <]$, the language *defined* by φ consists of all words w in which φ is satisfied. A language $\mathcal{L} \subseteq \Sigma^\star$ is MSO-*definable* if $\mathcal{L}$ is defined by some $\varphi \in \mathsf{MSO}[\Sigma, <]$ as above, and FO-*definable* if it is defined by some $\varphi \in \mathsf{FO}[\Sigma, <]$. It appears that regular languages exactly coincide with ones definable in MSO.

▶ **Theorem 17.** *A language of finite words over a finite alphabet is regular if and only if it is* MSO*-definable.*

The next view is through semigroup homomorphisms. Consider a language $\mathcal{L} \subseteq \Sigma^\star$. By endowing $\Sigma^\star$ with the concatenation operation we can regard it as a semigroup. For another semigroup S and a (semigroup) homomorphism $h\colon \Sigma^\star \to S$, we say that h *recognizes* $\mathcal{L}$ if there exists $A \subseteq S$ such that $\mathcal{L} = h^{-1}(A)$; in other words, whether $w \in \mathcal{L}$ can be recognized by looking at $h(w)$ and determining whether it belongs to A. It turns out that regular languages are also exactly those that are recognized by homomorphisms to finite semigroups.

▶ **Theorem 18.** *A language of finite words over a finite alphabet is regular if and only if it is recognized by a homomorphism to a finite semigroup.*

Further, it is known that if $\mathcal{L}$ is regular, then there exists a unique minimal – in terms of cardinality – semigroup S such that there is a homomorphism from $\Sigma^\star$ to S recognizing $\mathcal{L}$. This semigroup is called the *syntactic semigroup* for S.

It turns out that FO-definable languages can be characterized in terms of algebraic properties of their syntactic semigroups. Here, a semigroup is *aperiodic* (or *group-free*) if it does not contain any non-trivial group.

▶ **Theorem 19** (Schützenberger-McNaughton-Papert Theorem, [32, 40])**.** *A regular language* $\mathcal{L}$ *is* FO*-definable if and only if its syntactic semigroup is aperiodic.*

With these standard tools recalled, we can proceed to the setting of dynamic data structures.

Fix a finite alphabet Σ and consider a language $\mathcal{L} \subseteq \Sigma^\star$. The *word problem* for $\mathcal{L}$ is to design a data structure that maintains a dynamic word $w \in \Sigma^\star$ and supports the following operations:

- $\mathsf{init}(w)$: Initialize the data structure with the given word w.
- $\mathsf{update}(i, a)$: Update w by replacing the symbol at position i by symbol $a \in \Sigma$.
- $\mathsf{query}()$: Determine whether $w \in \mathcal{L}$.

The complexity guarantees of such a data structure is typically measured in terms of $n := |w|$. Note that this value is fixed upon initialization and then stays the same throughout the life of the data structure.

We can also consider the word problem for semigroups. Suppose S is a semigroup. Then the word problem for S is defined as above for words over S (that is, words $w \in S^\star$), where query is redefined as follows: Output the (left-to-right) product of all the symbols in w.

Observe that the word problem for a regular language $\mathcal{L} \subseteq \Sigma^\star$ easily reduces to the word problem for its syntactic semigroup S. Indeed, if $h\colon \Sigma^\star \to S$ is the homomorphism recognizing $\mathcal{L}$, say $\mathcal{L} = h^{-1}(A)$ for some $A \subseteq S$, then in the reduction we can map symbols $a \in \Sigma$ to their images $h(a) \in S$, and whether $w \in \mathcal{L}$ can be deduced by checking whether $h(w) \in A$.

Frandsen et al. [19] proposed an efficient dynamic data structure for the word problem in aperiodic semigroups.

▶ **Theorem 20** ([19]). *Let S be a finite aperiodic semigroup. Then there is a data structure for the word problem for S with initialization time $\mathcal{O}(n)$, worst-case update time $\mathcal{O}(\log \log n)$, and worst-case query time $\mathcal{O}(1)$.*

By combining Theorems 19 and 20 using the reduction presented above, we obtain the following.

▶ **Theorem 21.** *Let Σ be a finite alphabet and suppose $\mathcal{L} \subseteq \Sigma^\star$ is FO-definable. Then there is a data structure for the word problem for $\mathcal{L}$ with initialization time $\mathcal{O}(n)$, worst-case update time $\mathcal{O}(\log \log n)$, and worst-case query time $\mathcal{O}(1)$.*

A few remarks are in order. First, the proof of Theorem 20 relies on induction on the Khron-Rhodes decomposition of the semigroup S, where in each step of the induction one applies van Emde Boas trees [42]. The induction has depth bounded by the size of S, so one can view this data structure as $\mathcal{O}(|S|)$ van Emde Boas trees stacked "on top of each other". Consequently, the constants hidden in the $\mathcal{O}(\cdot)$ notation in Theorem 20 depend on S, but not horribly: they are polynomial in $|S|$. However, there is a much more significant complexity blow-up hidden in Theorem 18. Specifically, if a regular language $\mathcal{L}$ is defined by an $\mathsf{FO}[\Sigma, <]$ sentence φ, then the syntactic semigroup of $\mathcal{L}$ has size bounded by a function of $|\varphi|$, but this function is in general non-elementary – it is basically a tower of height equal to the quantifier rank of φ. This non-elementary dependence is known to be unavoidable [41]. Therefore, whenever one applies Theorem 21 in order to obtain data structures for a problem based on its description in FO, one should bear in mind that the constants hidden in the $\mathcal{O}(\cdot)$ notation depend non-elementarily on the length of the description.

Second, recently Amarilli et al. [4] gave a characterization of regular languages for which data structures with guarantees as in Theorem 20 exist. This characterization renders the tractability region to be a bit broader than just FO-definability, for instance the languages "on every even position there is symbol a" or "in total there is an even number of symbols a" are not FO-definable, but admit data structures for the word problem with constant update time. The characterization is expressed in algebraic terms and we could not find natural examples of parameterized string problems that would not be FO-definable, but fall under the characterization. So we refrain from giving more details and point an interested reader to [4] instead.

We now explain how to use Theorem 21 in practice on two examples: Disjoint Factors and Edit Distance. In each case, the task boils down to defining the problem in $\mathsf{FO}[\Sigma, <]$ for an appropriate alphabet Σ. We start with Disjoint Factors.

▶ **Lemma 22.** *Let $k \in \mathbb{N}$ and $\Sigma_k = [k]$. There is a sentence $\varphi_k \in \mathsf{FO}[\Sigma_k, <]$, computable from k, such that for every $w \in \Sigma_k^\star$, w is a yes-instance of* Disjoint Factors *for parameter k if and only if $w \models \varphi_k$.*

Proof. In the sentence φ_k, we first make a disjunction over all permutations $\pi\colon [k] \to [k]$. For each such π, we verify that there exist positions $x_1 < y_1 < x_2 < y_2 < \ldots < x_k < y_k$ such that for each $i \in [k]$, both at position x_i and at y_i there is symbol $\pi(i)$. It is straightforward to express this condition using an $\mathsf{FO}[\Sigma_k, <]$ sentence. ◀

By applying Theorem 21 to the language defined by sentence φ_k provided by Lemma 22, we obtain the following.

▶ **Corollary 23.** *There is a data structure for the dynamic* Disjoint Factors *problem with initialization time $\mathcal{O}_k(n)$, worst-case update time $\mathcal{O}_k(\log\log n)$, and query time $\mathcal{O}(1)$.*

Note here that the query time can be a constant independent of k, as we can always recompute the answer to the query following every update.

For Edit Distance, the formula is more complicated. For two words $u, v \in \Sigma^\star$, by $u \otimes v$ we denote the word over $(\Sigma \cup \{\bot\})^2$, where $\bot$ is a symbol not present in Σ, defined as follows:

- The length of $u \otimes v$ is $\max(|u|, |v|)$.
- For each $1 \leqslant i \leqslant \min(|u|, |v|)$, we put $u \otimes v[i] = (u[i], v[i])$.
- For each $\min(|u|, |v|) < i \leqslant \max(|u|, |v|)$, we put $u \otimes v[i] = (u[i], \bot)$ or $u \otimes v[i] = (\bot, v[i])$, depending on whether $\max(|u|, |v|) = |u|$ or $\max(|u|, |v|) = |v|$.

▶ **Lemma 24.** *Let $k \in \mathbb{N}$ and Σ be a finite alphabet. There is a sentence $\psi_{k,\Sigma} \in \mathsf{FO}[(\Sigma \cup \{\bot\})^2, <]$, computable from k and Σ, such that for all $u, v \in \Sigma^\star$, we have $\mathsf{ed}(u, v) \leqslant k$ if and only if $u \otimes v \models \psi_{k,\Sigma}$.*

Proof. Denote $\Gamma = (\Sigma \cup \{\bot\})^2$ for brevity. Note that for two words $u', v' \in \Sigma^\star$ we have $\mathsf{ed}(u', v') \leqslant k$ if and only if there exist integers $a, b, c \geqslant 0$ with $a + b + c \leqslant k$ such that one can remove a positions from u' and b positions from v' so that the resulting strings have equal length and differ on exactly c positions. In such case, we will call the pair (u', v') (a, b, c)-*editable*.

For all $(a, b, c) \in \{0, 1, \ldots, k\}^3$ with $a+b+c \leqslant k$ and $s, t \in \{-k, \ldots, k\}$, we shall construct a formula $\alpha_{s,t,a,b,c}(x, y)$ that satisfies the following: for two positions $1 \leqslant x \leqslant y \leqslant \max(|u|, |v|)$, we have

$$u \otimes v \models \alpha_{s,t,a,b,c}(x, y) \qquad \text{if and only if} \qquad (u[x : y], v[x + s : y + t]) \text{ is } (a, b, c)\text{-editable.}$$

Here, we use the convention that if the specified range $[x : y]$ or $[x + s : y + t]$ does not fit into the corresponding word, or makes no sense due to $x > y + 1$ or $x + s > y + t + 1$, then $\alpha_{s,t,a,b,c}(x, y)$ should be false (this can be easily recognized in $\mathsf{FO}[\Gamma, <]$). If we achieve the above, the formula $\psi_{k,\Sigma}$ can be defined as the disjunction of all formulas $\alpha_{0,t,a,b,c}(1, n_u)$ for a, b, c as above, where 1 is the first position, n_u is the last position of u, and $t \in \{-k, \ldots, k\}$ is such that $n_u + t$ is the last position of v (all these are easily definable from $u \otimes v$ in $\mathsf{FO}[\Gamma, <]$).

The construction is by induction on $a + b$. For the base case $a = b = 0$, we may define $\alpha_{s,t,0,0,c}(x, y)$ as follows: if $s \neq t$ then the formula is always false, and otherwise it checks whether there are exactly c different positions z such that $x \leqslant z \leqslant y$ and $u[z] \neq v[z + s]$. This can be checked by comparing the first coordinate of $u \otimes v[z]$ with the second coordinate of $u \otimes v[z + s]$. Since $|s| \leqslant k$ by assumption, it is straightforward to formulate this assertion in $\mathsf{FO}[\Gamma, <]$.

We proceed to the induction step. So assume $a + b > 0$, say $a > 0$; the construction in the case $b > 0$ is analogous, so we omit it. The idea is that we guess, by existential quantification, the first position in $u[x : y]$ that gets removed, and use simpler formulas given by the induction assumption. More precisely, $\alpha_{s,t,a,b,c}(x, y)$ can be defined as the conjunction of formulas

$$\exists z.\, (x \leqslant z \leqslant y \wedge \alpha_{s,r,0,b_1,c_1}(x, z - 1) \wedge \alpha_{r-1,t,a-1,b_2,c_2}(z + 1, y))\,,$$

for all integers $b_1, b_2 \geqslant 0$ with $b_1 + b_2 = b$, $c_1, c_2 \geqslant 0$ with $c_1 + c_2 = c$, and $r \in \{-k+1, \ldots, k\}$. Here, $z - 1$ and $z + 1$ are a syntactic sugar for the predecessor and the successor of z, respectively, which are easily definable in $\mathsf{FO}[\Gamma, <]$. It is straightforward to see that the construction of $\alpha_{s,t,a,b,c}(x, y)$ as above satisfies the required properties. ◀

Similarly as before, by combining Theorem 21 with Lemma 24 we obtain the following.

▶ **Corollary 25.** *There is a data structure for the dynamic* Edit Distance *problem with initialization time $\mathcal{O}_{k,\Sigma}(n)$, worst-case update time $\mathcal{O}_{k,\Sigma}(\log \log n)$, and query time $\mathcal{O}(1)$.*

B Omitted proofs

Proof of Lemma 11. We assume the familiarity with the proof of Lemma 7, as our data structure will extend the one proposed there. Recall that in the context of Lemma 11, we maintain the data structure of Lemma 7 for the dictionary $\widetilde{\mathcal{S}}$ and $\tilde{o}$ is the maintained word.

The data structure of Lemma 7 stores, for every $\tilde{s} \in \widetilde{\mathcal{S}}$, the following set:

$$\Delta(\tilde{o}, \tilde{s}) = \{i \in [L] \mid \tilde{o}[i] \neq \tilde{s}[i]\}.$$

As, we have a fixed coloring $\pi \colon [L] \to [16d]$, we can maintain a table of counters

$$\mathcal{T}(\tilde{s}, c) \coloneqq |\{i \in [L] \mid \pi(i) = c \text{ and } i \in \Delta(\tilde{o}, \tilde{s})\}| \qquad \text{for all } \tilde{s} \in \widetilde{\mathcal{S}} \text{ and } c \in [16d].$$

Upon initialization, we explicitly compute $\mathcal{T}(\tilde{s}, c)$ for every $\tilde{s} \in \tilde{S}$ and $c \in [16d]$ in $\mathcal{O}(nL + nd)$ time. During an update to the position $i \in [L]$ of a word $\tilde{s} \in \widetilde{\mathcal{S}}$ we check if $\tilde{o}[i] \neq \tilde{s}[i]$ and update the number of colors in $\mathcal{T}(\tilde{s}, c)$ based on $\pi(i)$ accordingly. It may happen that the update to $\tilde{s}$ at position i triggered a modification of the word $\tilde{o}$ at position i. Then we need to change $\mathcal{T}(\tilde{s}, \pi(i))$ for every $\tilde{s} \in \widetilde{\mathcal{S}}$. Note that this alone requires $\mathcal{O}(|\mathcal{S}|)$ time. However, recall that in the proof of Lemma 7 we argued that before update when the position i of the word o changes, there were at least $|\widetilde{\mathcal{S}}|/4$ updates to that position where $\tilde{o}$ was not modified (recall here that we work over the binary alphabet). Therefore, as in the proof of Lemma 7, we may charge the running time $\mathcal{O}(|\mathcal{S}|)$ to those previous updates to argue that the amortized update time is $\mathcal{O}(1)$.

Observe that based on the table $\mathcal{T}$, we can compute the set $\mathsf{colors}_{\tilde{o},\pi}(\tilde{s}) = \{\pi(i) \mid i \in [L] \text{ and } \tilde{s}[i] = \tilde{o}[i]\}$ for any given $\tilde{s}$ in time $\mathcal{O}(d)$, because it is enough to iterate through all $c \in [16d]$ and check whether $\mathcal{T}(\tilde{s}, c) > 0$. Now we describe how to maintain the sets

$$\Phi(C) = \{\tilde{s} \in \widetilde{\mathcal{S}} \mid \mathsf{colors}_{\tilde{o},\pi}(\tilde{s}) = C\} \qquad \text{for every } C \subseteq [16d].$$

Each set $\Phi(C)$ is stored as a doubly-linked list of pointers to words from $\widetilde{\mathcal{S}}$. Upon initialization, we iterate over every $\tilde{s} \in \tilde{S}$, lookup the value of $C_{\tilde{s}} \coloneqq \mathsf{colors}_{\tilde{o},\pi}(\tilde{s})$ and add a pointer to $\tilde{s}$ to the list $\Phi(C_{\tilde{s}})$. Observe, that this operation can be implemented in total time $2^{\mathcal{O}(d)} + nL$ time, as we can compute sets $C_{\tilde{s}}$ for all $\tilde{s} \in \widetilde{\mathcal{S}}$ in total time $\mathcal{O}(nL)$.

Next, when a word $\tilde{s}$ is updated on some position and is changed to $\tilde{s}'$, we compute the previous set of colors $C \coloneqq \mathsf{colors}_{\tilde{o},\pi}(\tilde{s})$ and the new set of colors $C' \coloneqq \mathsf{colors}_{\tilde{o},\pi}(\tilde{s}')$. As argued, this operation can be done in $\mathcal{O}(d)$ time. Next, we delete the pointer to the word $\tilde{s}$ from the list $\Phi(C)$ and append a pointer to $\tilde{s}'$ to list $\Phi(C')$. Alongside $\tilde{s}$ we store the pointer to its list entry in the list $\Phi(C')$ in order to be able to remove it efficiently. Both of these operations can be implemented in $\mathcal{O}(1)$ time. During a query $C \subseteq [16d]$ we return any element from the list $\Phi(C)$ or assert that it is empty in $\mathcal{O}(1)$ time. ◀

An Algebraic Approach to Vectorial Programs

Charles Paperman
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Sylvain Salvati
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Claire Soyez-Martin
Univ. Lille, CNRS, INRIA, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract

Vectorial programming, the combination of SIMD instructions with usual processor instructions, is known to speed-up many standard algorithms. Simple regular languages have benefited from this technology. This paper is a first step towards pushing these benefits further. We take advantage of the inner algebraic structure of regular languages and produce high level representations of efficient *vectorial programs* that recognize certain classes of regular languages.

As a technical ingredient, we establish equivalences between classes of *vectorial circuits* and logical formalisms, namely unary temporal logic and first order logic. The main result is the construction of compilation procedures that turns *syntactic semigroups* into vectorial circuits. The circuits we obtain are *small* in that they improve known upper-bounds on representations of automata within the logical formalisms. The gain is mostly due to a careful sharing of sub-formulas based on algebraic tools.

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

Keywords and phrases Automata theory, Semigroups, Vectorisation

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.51

Related Version *Full Version*: `https://hal.archives-ouvertes.fr/hal-03831752v2`

Acknowledgements We would like to thank the anonymous referees for helping to improve the paper a lot, Howard Straubing for proof-reading the main algebraic proofs of this paper, Michaël Hauspie for his advice, his support and his knowledge about SIMD and compilation and Corentin Barloy to have helped proof-read the paper.

1 Introduction

Finite state machines abstract the simplest class of programs. They are used everywhere: basic string manipulation functions of the `C` standard library like `memchr`, `strlen` or `strstr` are based on simple finite state automata, but also text-processing related tasks; checking the validity of encodings; text-mining; etc. As finite state machines are pervasive, implementing them efficiently is key in many softwares. The string related functions of the `C` standard library we mentioned earlier have greatly benefited from SIMD instructions built into modern CPUs. These functions can now process several characters per CPU cycle.

Single Instruction, Multiple Data (SIMD) executes an operation on several data in parallel, offering a form of low-level parallelism akin to Lamport's [13]. A function like `memchr` searches the first occurrence of a character in a string. SIMD instructions can check whether this character appears among several consecutive characters of the string in one go: each individual character is compared in parallel with the others. Other *vectorized algorithms* (see [11] for examples) benefit from these instructions. In the context of text-processing, impressive handcrafted SIMD based implementations have been proposed for string pattern matching [12], classical regular expression matching [33], Json parsing [14], checking correctness UTF-8 encoding [10], or DNA alignement in bioinformatics [6].


© Charles Paperman, Sylvain Salvati, and Claire Soyez-Martin;
licensed under Creative Commons License CC-BY 4.0

Though many efforts have been put into compilers to solve the problem of *auto-vectorization* [18, 32, 19, 20, 8], these optimization methods rarely succeed in accelerating text algorithms with SIMD instructions. Finding auto-vectorization methods that deal with text algorithms would have a high impact. The challenge would be to produce, among others, the clever handcrafted code in the `C` standard library from the description of its underlying regular expression. Text processing, however, requires some form of sequentiality simply because, in many cases, information needs to be passed sideways. In this paper we consider two of them: prefix-or and addition.

The unreasonable power of binary addition. For sideways information passing, addition is very interesting: carry propagation can be used to compute long distance relations words. Moreover, *Big Int* instructions of modern processors compute it efficiently over large vectors.

Several papers already explored the use of addition in relation to regular languages: Myers [17] uses it to solve approximate string matching; Bergeron and Hamel [2, 1] show that counter-free automata can also benefit from addition; Serre [26] then characterizes counter-free languages in terms of addition; on the practical side, Cameron et al. [4] use explicitly addition when compiling parts of regular expressions of the form B^* when B is a set of letters.

This work starts with the definition of a notion of *vectorial circuit* that abstracts away from the details of CPU operations. Circuits make clear which operations are independent from one another and are objects of choice when it comes to introduce parallelism in some computation. We are confident that vectorial circuits can be compiled to obtain efficient programs that use SIMD instructions. As a first step in that direction, the main focus of the paper is then to construct small vectorial circuits from various presentations of counter-free regular languages.

The results of [2, 1] heavily rely on Krohn-Rhodes' Theorem. In this paper, the construction goes through Past-LTL logic and uses the equivalence between the Yesterday-Since operation and binary addition. The relation between these two operations is actually formalized in coq [21] (for technical reasons, in this formalization, we consider Forward-LTL and Next-Until instead of Past-LTL and Yesterday-Since). A consequence is that we can obtain concise vectorial circuits from Past-LTL formulae in the sense that they have the same size as the initial formula.

We also consider how to obtain concise vectorial circuits directly from automata. With Serre's result, it is theoretically possible to produce vectorial circuits for counter-free automata. However, only a double exponential upper bound in the size of the automaton is known on the size of Past-LTL formulas. This would make circuits far too large in practice. This upperbound comes from a construction of Wilke [35, Corollary 1] that builds formulas by induction over the structure of the *syntactic monoids* of the input automata. On other classes of automata, such as $\mathrm{FO}_2[<]$ [5, 31], known transformations of automata into formulas are indirect and not constructive. They are of no use to actually compile automata into formulas or circuits.

Main contributions. We study a notion of *vectorial circuits* as an abstraction of SIMD programs. Vectorial circuits describe the shape of actual circuits for arbitrary size of inputs. In particular they are a uniform fragment of AC^0 (constant depth polysize boolean circuits). As for circuits, their expressivity depends on the set of authorized gates. Our main goal is to produce small vectorial circuits for particular classes of regular languages: the class of counter-free or $\mathrm{FO}[<]$-languages and the class **DA** or $\mathrm{FO}_2[<]$-languages. We measure the

size of output circuits with respect to the size of the syntactic monoids. Were we to consider automata as inputs that we would need to exponentiate our bounds (e.g. the languages of the form A^kaA^* have syntactic monoids of size 2^k with minimal automata of size $O(k)$).

Concerning counter-free languages, we revisit Serre's result. We propose an algorithm that produces vectorial circuits (using bitwise boolean operations and addition) that are polynomial in the size of the input aperiodic monoid. Serre's result ensures here that the class of vectorial programs expressed with addition and bitwise boolean operations are equivalent to counter-free languages.

For the class **DA**, we replace addition with prefix-or to obtain a class of circuits that captures exactly that class. When transforming syntactic monoids of **DA** into vectorial circuits, the use of prefix-or makes circuits larger than they would be with addition. The transformation that we propose gives vectorial circuits which have exponential size in the $\mathcal{J}$-depth (see 3.1 for the definition of $\mathcal{J}$-depth) of the syntactic monoid.

We begin by presenting vectorial circuits in Section 2. We also show the links between vectorial circuits and fragments of logic. In Section 3, we introduce general strategies of evaluation of words. Then, in Section 4, we construct small programs that compute our strategies of evaluation. Sketches of all the main proofs can be found in the appendices. For the complete version, see [22].

2 Compiling regular languages into vectorial circuits

2.1 Algebraic preliminaries

We write $[n]$ for the set $\{0, \dots, n-1\}$. Given a set E, we denote by $|E|$ its cardinality. Given some finite set Σ, the *alphabet*, words on Σ are finite sequences of elements of Σ. We write $|x|$ for the length of the word x. We denote by Σ^* the set of words on Σ.

Semigroups. A *semigroup* is a pair consisting of a set S and an associative binary operation $\cdot_S$ on S, called the inner operation of S. We usually write that the set S is a semigroup. A *monoid* is a triple $(M, \cdot_M, 1)$, where $(M, \cdot_M)$ is a semigroup and $1 \in M$ is an identity (or a neutral element) of M. We usually write that the set M is a monoid. We only work with *finite* semigroups and monoids. We thus designate *finite semigroups* (resp. finite monoids) when we mention semigroups (resp. monoids). Given a semigroup S, any element e of S satisfying $e \cdot_S e = e$ is called an *idempotent*. In a finite semigroup S, any element s of S admits an *idempotent power*, which is an element s^n (where $n > 0$ is an integer) that is idempotent, where s^n denotes the iterated product of s by itself n times. We use the usual notation s^ω to denote the idempotent power of s (ω is the minimum integer such that, for any element s, s^ω is the idempotent power of s). Given a semigroup S, we define $S^1 = S \cup \{1\}$ as the monoid formed by the semigroup to which an identity is added if necessary. For any subsets X and Y of S, we denote by $X \cdot_S Y$ the set $\{x \cdot_S y \mid x \in X, y \in Y\}$. Similarly, for any $x \in S$ and $Y \subseteq S$, we write $x \cdot_S Y$ and $Y \cdot_S x$ respectively for $\{x\} \cdot_S Y$ and $Y \cdot_S \{x\}$. Given a finite set Σ, we call Σ^+ the *free semigroup over* Σ with the concatenation as the associative binary operation. This is the only infinite semigroup that we consider. Given a semigroup S, we will denote by S^+ the free semigroup with the underlying set of S as alphabet.

Canonical morphism. We denote concatenation implicitly: given two words u, v, their concatenation is written uv. For instance, taking two elements x, y of S, xy denotes a word of S^+ of length 2. This notation **must not** be confused with $x \cdot_S y$ that denotes the element of S obtained by multiplying x and y with the inner operation of S. We **never use** concatenation

to mark the product within S. However, we relate words of the free semigroup S^+ to their value in S by means of *the canonical morphism*: $\pi_S \colon S^+ \to S$. It is the unique associative morphism verifying both the following properties: for every $x \in S$, $\pi_S(x) = x$ and, for every $u, v \in S^+$, $\pi_S(uv) = \pi_S(u) \cdot_S \pi_S(v)$.

Languages and semigroups. A link can be established between logics and semigroups by taking the *syntactic semigroup* of a language. This semigroup is defined as follows: given a language L on an alphabet Σ, the *syntactic congruence* of L in Σ^* is the relation $\sim_L$ defined on Σ^* such that, for any words $u, v \in \Sigma^*$, $u \sim_L v$ if and only if, for all words $x, y \in \Sigma^*$, $xuy \in L \Leftrightarrow xvy \in L$. The syntactic semigroup of L is the quotient $\Sigma^+/\sim_L$, and the syntactic monoid of L is the quotient $\Sigma^*/\sim_L$.

The link between logic and semigroups has already been well studied and gave birth to very nice algebraic characterizations of some well-known classes of languages. For more information on this topic, see the survey [30] along with the books [28] and [23]. Notably, the class of starfree languages is equivalent to the variety of aperiodic semigroups, the semigroups satisfying an equation of the form $\pi_S(x^{\omega+1}) = \pi_S(x^\omega)$, for any $x \in S$. We can also mention the class $\mathrm{FO}_2[<]$, which is equivalent to the variety **DA** of semigroups, the semigroups satisfying an equation of the form $\pi_S((xy)^\omega x (xy)^\omega) = \pi_S((xy)^\omega)$, for any $x, y \in S$. For an explanation of the name **DA**, see Section 3.1. We refer to [29] for a complete exposition of this class and its relationship to various logics.

2.2 Vectorial circuits

We call the words on the alphabet $\{0, 1\}$ *vectors*. For a vector $\mathbf{x}$, we may use the term *dimension* to refer to its length $|\mathbf{x}|$. We refer to vectors of dimension n as n-vectors. We let $\mathbf{1}_n$ and $\mathbf{0}_n$ respectively denote the sequence of n 1's and the sequence of n 0's. When n is irrelevant or obvious for the context, we may write $\mathbf{1}$ and $\mathbf{0}$. Two vectors $\mathbf{x} = x_0 \dots x_{n-1}$ and $\mathbf{y} = y_0 \dots y_{n-1}$, of dimension $n \in \mathbb{N}$, are said to be *disjoint* if, for any index $i \leq n-1$, $x_i \wedge y_i = 0$.

Vectorial circuits and their semantics. *Vectorial circuits* are labeled directed acyclic graphs. The nodes that have no input edge are called *input nodes*. The nodes with incoming edges are called *gates* and are labeled with *commutative operations*. The in-degree of a gate should be equal to the arity of its labeling operation while the out-degree can be arbitrary. We usually write input nodes and terms in bold-face fonts: $\mathbf{v}$, $\mathbf{v_1}, \dots$ *Output nodes* are distinguished nodes of the circuit. Generally they include the nodes that have no output edge. The *size of a circuit* is the number of its nodes.

Vectorial circuits can be seen as circuit templates that, for each n, instantiate a *concrete circuit* working on vectors of dimension n. Once the dimension is fixed to n, associating n-vectors to the input nodes and flowing the values through the gates (where the right function operating on n-vectors is used) yields output values in the output nodes. Take a circuit C with input nodes $\mathbf{i_1}, \dots, \mathbf{i_p}$ and output nodes $\mathbf{o_1}, \dots, \mathbf{o_r}$, given p n-vectors $\mathbf{x_1}$, $\dots$, $\mathbf{x_p}$, we write $C(\mathbf{x_1}, \dots, \mathbf{x_p})$ for the tuple of n-vectors $\mathbf{y_1}$, $\dots$, $\mathbf{y_r}$ that are respectively yielded in the output nodes $\mathbf{o_1}$, $\dots$, $\mathbf{o_r}$ when evaluating the C with the vector $\mathbf{x_1}$, $\dots$, $\mathbf{x_p}$ respectively associated to the input nodes $\mathbf{i_1}$, $\dots$, $\mathbf{i_p}$.

Operations for labeling gates. We use bitwise Boolean operations: the unary negation $\neg$ and the binary operations $\wedge$ and $\vee$, respectively the bitwise conjunction and disjunction.

Given a function $f : \{0, 1\}^+ \to \{0, 1\}$, we define the unary operation pref-f (resp. suf-f): given a n-vector $\mathbf{x} = b_0 \dots b_{n-1}$, with $b_0, \dots, b_{n-1}$ in $\{0, 1\}$, pref-$f(\mathbf{x})$ (resp. suf-$f(\mathbf{x})$) is the n-vector $\mathbf{z} = c_0 \dots c_{n-1}$ where for each $i \in [n]$, $c_i = f(b_0 \cdots b_i)$ (resp. $c_i = f(b_i \cdots b_{n-1})$). In this paper, we use the unary operations pref-$\vee$, suf-$\vee$, pref-$\wedge$ and suf-$\wedge$.

Binary vectors $\mathbf{x}$ of dimension n naturally represent numbers in $[2^n]$. We write $nb(\mathbf{x})$ for the number represented by $\mathbf{x}$ with the convention that its least significant bit is the left-most one. For a natural number k, we write $bin_n(k)$ to denote the vector of dimension n that represents k modulo 2^n.

The unary operations LSB (Least Significant Bit) and MSB (Most Significant Bit) replace by 0 respectively the left-most 1 and right-most 1 of their argument vector. For these two operations, when the argument vector is $\mathbf{0}_n$, the resulting vector is also $\mathbf{0}_n$. The binary operation + is defined by the family $(\text{plus}_n)_{n\in\mathbb{N}}$ so that for any two vectors of dimension n, denoted by $\mathbf{x}$ and $\mathbf{y}$, $\text{plus}_n(\mathbf{x}, \mathbf{y}) = bin_n(nb(\mathbf{x}) + nb(\mathbf{y}))$.

We study two families of vectorial circuits:

- Sweeping-*vectorial circuits*, circuits built only with the operations, $\wedge$, $\vee$, $\neg$, pref-$\vee$, pref-$\wedge$, suf-$\vee$, suf-$\wedge$, LSB, MSB,
- and ADD-*vectorial circuits*, circuits built only with the operations of Sweeping-vectorial circuits and +.

Term notation for vectorial circuits. Trees are a particular kind of directed acyclic graphs. Circuits that are tree shaped are those where nodes have at most one out-going edge and exactly one output node. These particular circuits can be advantageously denoted by terms built with operations (respecting their arity) and input nodes. The term $(\mathbf{v_1} \wedge \neg\mathbf{v_2}) \vee (\neg\mathbf{v_3} \wedge \mathbf{v_4})$ represents the left-most circuit of Figure 1. Allowing input nodes to have several occurrences in terms gives access to some limited kind of sharing. This is exemplified with the central and right-most circuits of Figure 1. So as to fully capture such sharing capabilities with the term notation, we use equations: a term t that is to be shared is associated to a node $\mathbf{v}$ with the equation $\mathbf{v} = t$ and, when $\mathbf{v}$ is used in another term, this refers to the *shared* circuit t. For example, we write $\mathbf{v} = \text{pref-}\vee(\text{suf-}\vee(\mathbf{v_1}))$, $(\neg\mathbf{v} \wedge \mathbf{v_2}) \vee (\mathbf{v} \wedge \mathbf{v_3})$ to denote the third circuit in Figure 1.

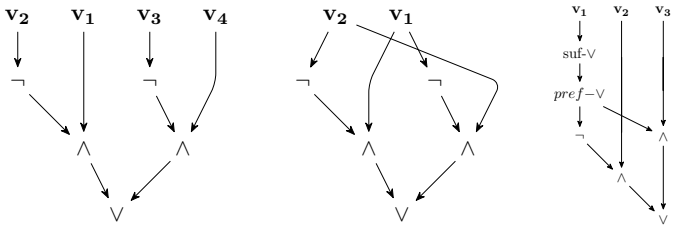


Figure 1 Graph representation of terms: $(\mathbf{v_1} \wedge \neg\mathbf{v_2}) \vee (\neg\mathbf{v_3} \wedge \mathbf{v_4})$; $(\mathbf{v_1} \wedge \neg\mathbf{v_2}) \vee (\neg\mathbf{v_1} \wedge \mathbf{v_2})$; and $\mathbf{v} = \text{pref-}\vee(\text{suf-}\vee(\mathbf{v_1}))$, $(\neg\mathbf{v} \wedge \mathbf{v_2}) \vee (\mathbf{v} \wedge \mathbf{v_3})$.

Terms also offer a convenient way for reusing certain circuits in several places: it suffices to change the input nodes at their leaves. We adopt a notation that denotes *parametrized circuits*: $\mathrm{c}(\mathbf{v_1}, \dots, \mathbf{v_n}) := t$ where t is a term built with the nodes $\mathbf{v_1}, \dots, \mathbf{v_n}$. For circuits $t_1, \dots, t_n$, we write $\mathrm{c}(t_1, \dots, t_n)$ the circuit described by the term obtained by replacing $\mathbf{v_1}, \dots, \mathbf{v_n}$ respectively with $t_1, \dots, t_n$ in t. For example, we define the bitwise exclusive-or as $\mathbf{v_1} \oplus \mathbf{v_2} := (\mathbf{v_1} \wedge \neg\mathbf{v_2}) \vee (\neg\mathbf{v_1} \wedge \mathbf{v_2})$. We can compose parametrized circuits to define others et construct complex circuits.

Parameterized circuits. We can compose parametrized circuits to define others:

- NotZero($\mathbf{v}$) := pref-$\vee$(suf-$\vee$($\mathbf{v}$))
- IsZero($\mathbf{v}$) := $\neg$NotZero($\mathbf{v}$)
- IfThenElse($\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$) := $\mathbf{v}$ = NotZero($\mathbf{v_1}$), ($\mathbf{v} \wedge \mathbf{v_2}$) $\vee$ ($\neg\mathbf{v} \wedge \mathbf{v_3}$)
- Thr2($\mathbf{v}$) := NotZero(LSB($\mathbf{v}$))

The parametrized circuits NotZero($\mathbf{v}$), IsZero($\mathbf{v}$), Thr2($\mathbf{v}$) perform tests on their input: on an n-vector, they return $\mathbf{1}_n$ when the test succeeds and $\mathbf{0}_n$ when it fails. NotZero($\mathbf{v}$) tests whether its input vector contains an occurrence of 1, IsZero($\mathbf{v}$) tests whether its input vector is $\mathbf{0}$ and Thr2($\mathbf{v}$) tests whether its input vector contains at least two occurrences of 1. The parametrized circuit IfThenElse($\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$), tests whether $\mathbf{v_1}$ is $\mathbf{0}$ and in this case outputs $\mathbf{v_2}$. Otherwise, it outputs $\mathbf{v_3}$. With parameterized circuits, we can construct complex circuits. An example is the circuit IfThenElse($\mathbf{v_1} \oplus \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_2} \oplus \mathbf{v_3}$) which is depicted in Figure 2.

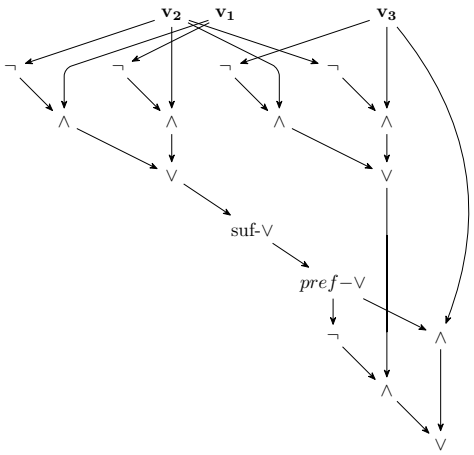


Figure 2 Circuit represented by IfThenElse($\mathbf{v_1} \oplus \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_2} \oplus \mathbf{v_3}$) .

The addition Lemma. We say that a vector $\mathbf{x}$ is contained in a vector $\mathbf{y}$ if for any position i, $\mathbf{x}_i = 1$ implies $\mathbf{y}_i = 1$. Let $\mathbf{x}, \mathbf{y}$ be two disjoint vectors of dimension n and $\mathbf{z}$ a vector of dimension n that contains both $\mathbf{x}$ and $\mathbf{y}$. We denote by $\mathbf{v}$ = Successor $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ the vector such that for all $i < n$, $\mathbf{v}_i = 1$ if and only if $\mathbf{x}_i = 1$, there exists $j < i$ such that $\mathbf{y}_j = 1$ and, for all $k \in \mathbb{N}$ such that $j < k < i$, $\mathbf{z}_k = 0$. In other words, Successor $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ indicates the positions marked in $\mathbf{x}$ that follow a marked position of $\mathbf{y}$, with no other position marked by $\mathbf{z}$ in-between.

▶ **Lemma 1** (Addition lemma). *Let* $\mathbf{x}, \mathbf{y}$ *be two disjoint vectors of dimension* n *and* $\mathbf{z}$ *a vector of dimension* n *that contains both* $\mathbf{x}$ *and* $\mathbf{y}$*. Then, we have* Successor $(\mathbf{x}, \mathbf{y}, \mathbf{z}) := (\mathbf{y}+(\mathbf{y} \vee \neg\, \mathbf{z})) \wedge \mathbf{x}$

This lemma has a rather tedious proof. So as to relieve the reader from checking its details, we provide a formalization and a proof this Addition Lemma in Coq [21]. The proof consists of two steps. First, we show that vectorial circuits built with the Next-Until modality of LTL, aka XU, and logical gates, are equivalent to circuits built with addition and logical gates. This equivalence is proved by constructing circuits based on XU and logical gates to encode addition and circuits based on addition and logical gates to encode XU. The intuition behind this equivalence is that carry propagation and XU both propagate information sideways. Second, encoding XU in first order-logic and using the previous equivalence allows us to relate the computation of the circuit of Lemma 1 to the specification of Successor $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and prove the relation correct.

Vectorial circuits as language recognizers and functions from words to finite sets. Given a fixed alphabet Σ, we say that a vectorial circuit C *recognizes* a set of words in Σ^+ when it has a unique output node and there is a bijection between the letters $a_1, \ldots, a_p$ of Σ and its input nodes $\mathbf{a_1}, \ldots, \mathbf{a_p}$. We consider a particular bijection: given a word u of length n, we write $\mathbb{1}_a(u)$ for the n-vector $\mathbf{x}$ so that $\mathbf{x}_i = 1$ if and only if $u_i = a$ for every i in $[n]$. We say that u is *accepted* or *recognized* by the circuit when $C(\mathbb{1}_{a_1}(u), \ldots, \mathbb{1}_{a_p}(u)) \neq \mathbf{0}$. As a shorthand, we write $\mathrm{enc}(u)$ for the tuple $(\mathbb{1}_{a_1}(u), \ldots, \mathbb{1}_{a_p}(u))$ and thus $C(\mathrm{enc}(u))$ for $C(\mathbb{1}_{a_1}(u), \ldots, \mathbb{1}_{a_p}(u))$.

Vectorial circuits can also represent functions f from Σ^+ to a finite domain E. It suffices to consider circuits C which have a bijection between their input nodes and the letters of Σ, but also a bijection between the elements $e_1, \ldots, e_r$ of E and their output nodes $\mathbf{e_1}, \ldots, \mathbf{e_r}$. We say that C *represents* f when for every u, the output $(\mathbf{z_1}, \ldots, \mathbf{z_r}) = C(\mathrm{enc}(u))$ is such that, for every i in $\{1, \ldots, r\}$, $\mathbf{z_i} \neq \mathbf{0}$ if and only if $f(u) = e_i$.

Example. Consider the alphabet $\Sigma = \{a, b, c\}$. The language $c^*a\Sigma^*$ is recognized by the Sweeping-vectorial circuit $C = \neg(\text{suf-}\vee(\text{pref-}\vee(\text{suf-}\vee(F_a) \wedge \mathbf{b})))$, where the circuit $F_a = \mathrm{LSB}(\mathbf{a}) \oplus \mathbf{a}$ gives the position of the first occurrence of a. The language $\Sigma^*ac^*a\Sigma^*$ is recognized by the ADD-vectorial circuit $C = (\mathbf{a} + \neg\mathbf{b}) \wedge \mathbf{a}$, which is equal to $(\mathbf{a} + (\mathbf{a} \vee \neg(\mathbf{a} \vee \mathbf{b}))) \wedge \mathbf{a} = \mathrm{Successor}(\mathbf{a}, \mathbf{a}, \mathbf{a} \vee \mathbf{b})$. An example is showed in Table 1; note that every 1 in the last vector indicates, as we want, each letter a such that there is no letter b between this position and the previous letter a, and no other letter is indicated.

Table 1 Example of application of the operation $(\mathbb{1}_a + \neg\mathbb{1}_b) \wedge \mathbb{1}_a$.

$\mathbb{1}_b$	0	0	1	0	0	0	0
$\mathbb{1}_a$	0	1	0	1	0	1	1
$\mathbb{1}_a + \neg\mathbb{1}_b$	1	0	1	0	0	1	1
$(\mathbb{1}_a + \neg\mathbb{1}_b) \wedge \mathbb{1}_a$	0	0	0	0	0	1	1

2.3 Going through first-order logic

The computational model we propose is actually very close to some classical fragments of logic over words. For a full exposition of this subject, we refer to [5]. Informally, we can describe some regular languages with the help of first-order formulas. In those formulas, quantifications range over positions of the word and atomic predicates can check either numeric constraints between positions (typically their order) or the label of the position. For instance, the formula $\exists x, y,\ a(x) \wedge a(y) \wedge \forall z,\ ((x < z \wedge z < y) \Rightarrow c(z))$ describes the language over the alphabet $\Sigma = \{a, b, c\}$ which words belong to $D = \Sigma^*ac^*a\Sigma^*$.

In this context, linear temporal logic (LTL for short) has an interesting role that we will rely on. LTL, even restricted to future or past operators, allows to define the same languages as full first-order logic [9]. For instance, the language D can be described with the following LTL formula: $\mathrm{F}(a \wedge \mathrm{X}(c\mathrm{U}a))$.

This class of languages has many equivalent characterizations. In [25], Schützenberger proved for regular languages the equivalence between aperiodic languages and *star-free* languages. Later, McNaughton and Papert proved that star-free languages are equivalent with LTL and first-order logic [15, 5]. In a similar fashion to McNaughton and Papert, we can prove that languages computed by ADD-vectorial circuits are exactly the starfree languages. Both directions of the equivalence have been proved by Serre [26, Theorem 1.] by relying on a syntactic logical rewriting of LTL formulas.

▶ **Proposition 2.** *Let L be a regular language. The following propositions are equivalent.*
1. *L is computed by an* ADD*-vectorial circuit.*
2. *L is starfree.*
3. *L is definable in Forward-LTL.*

Proof. The equivalence between 1 and 2 is from [26], Corollary 1. In this paper, the vectorial computational model consists of straight-line expressions over a basis which includes addition, Boolean operations and right shift. This basis is equivalent to ADD-vectorial circuits. The equivalence between 2 and 3 is standard. It can be proved by combining theorems proved by Kamp [9] and McNaughton and Papert [15]. The former proves the equivalence between being definable in Forward-LTL and First-Order logic for regular languages and the later the equivalence between First-Order logic and star-free languages. ◀

Another fragment of interest is the class of languages definable by two-variable first-order logic, or equivalently by LTL without the Until operation but with neXt and Yesterday operations. This fragment has been studied a lot and is well understood [9, 5, 34, 36].

In the proof of the next result, we rely on the known equivalence between $\mathrm{FO}_2[<]$ and a logic called $\mathrm{TL}[X_a, Y_a]$. More formally, the languages describable in $\mathrm{FO}_2[<]$ are exactly the languages describable by the logic $\mathrm{TL}[X_a, Y_a]$ presented in [5]. This logic uses only two kinds of operators: X_a, for a any letter, verifies that there exists a position greater than the current one which holds an a, and if so, it sets the new current position at the position of the closest a. If no occurrence of a is found, the word is not recognized. The operator Y_a, for any letter a, is the symmetrical operator: it operates the same way except for the fact that it searches for the closest occurrence of a before the current position.

▶ **Proposition 3.** *A language is recognized by a* Sweeping*-vectorial circuit if and only if it is definable in $TL[X_a, Y_a]$.*

Proof. We rely on the equivalence between $\mathrm{FO}_2[<]$ and $\mathrm{TL}[X_a, Y_a]$, to prove the result. First, we show that $\mathrm{TL}[X_a, Y_a]$ is captured by Sweeping-vectorial circuits, i.e. that any formula of $\mathrm{TL}[X_a, Y_a]$ defines a function from words to Boolean vectors that can be modeled by these circuits; here we interpret a formula of $\mathrm{TL}[X_a, Y_a]$ as a function from words to Boolean vectors, by considering the truth vector of a formula on every position of the input word. Clearly, Boolean operations are equivalent in both models. Now, suppose that we have a Sweeping-vectorial circuit C_α that is equivalent to a $\mathrm{TL}[X_a, Y_a]$ formula α, i.e. given a word $u \in \Sigma^+$ and a vector $\mathbf{pos}$ (of the same dimension as u) that has a unique 1 in some position x, we have $C_\alpha(\mathbf{pos}, \mathrm{enc}(u)) \neq \mathbf{0}$ if and only if $u, x \models \alpha$. Then, for any letter $a \in \Sigma$, the formula $\beta = X_a\alpha$ can be modeled by the circuit C_β defined as follows: for any word $u \in \Sigma^*$ and any vector $\mathbf{pos}$ having a unique 1, we begin by finding the first letter a strictly after the position marked by $\mathbf{pos}$, by sequentially computing the following vectors: $\mathbf{authorizedPos} = \mathrm{LSB}(\text{pref-}\vee(\mathbf{pos}))$, $\mathbf{next} = \mathbf{authorizedPos} \wedge \mathbb{1}_a(u)$ and $\mathbf{nextPos} = \mathbf{next} \oplus \mathrm{LSB}(\mathbf{next})$. This gives us the new position vector to give as parameter to the new circuit. Then we can define $C_\beta(\mathbf{pos}, \mathrm{enc}(u)) := C_\alpha(\mathbf{nextPos}, \mathrm{enc}(u))$, which is equivalent to β. Finally, we can model the formula $\gamma = Y_a\alpha$ using the same formulas, except for the vector $\mathbf{authorizedPos}$, which is computed as $\mathbf{authorizedPos} = \mathrm{MSB}(\text{suf-}\vee(\mathbf{pos}))$, in order to get the positions situated strictly before the position x marqued by $\mathbf{pos}$.

Now, we prove that Sweeping-vectorial circuits are captured by formulas in $\mathrm{FO}_2[<]$. Since Sweeping-vectorial circuits output a truth vector indicating if some property is true for each position in the input word, we must consider $\mathrm{FO}_2[<]$ formulas which can take into account a starting position. Thus, we consider only $\mathrm{FO}_2[<]$ formulas with one free variable, which represents the position in which we begin to evaluate the formula in the input word.

Clearly, Boolean operations are equivalent in both models. We can also interpret pref-$\vee$ as an $\mathrm{FO}_2[<]$ formula: consider an $\mathrm{FO}_2[<]$ formula φ with one free variable and suppose that it is computed by a circuit C_φ. Then, the formula $\varphi_{\text{pref-}\vee}$ with one free variable defined by $\varphi_{\text{pref-}\vee}(y) := \exists x \leq y,\ \varphi(x)$ has its truth vector equal to the output of pref-$\vee(C_\varphi)$. The other prefix and suffix operations can be dealt with in a similar way. Finally, we can interpret LSB and MSB as $\mathrm{FO}_2[<]$ formulas. Indeed, consider an $\mathrm{FO}_2[<]$ formula φ with one free variable and suppose that it is computed by a circuit C_φ. Then, the formula φ_{LSB} with one free variable defined by $\varphi_{\mathrm{LSB}}(y) := \varphi(y) \wedge \exists x < y,\ \varphi(x)$ has its truth vector equal to the output of $\mathrm{LSB}(C_\varphi)$. MSB is equivalent to the same formula where $x < y$ is replaced by $x > y$. ◀

▶ Remark 4. We can always build a vectorial circuit of size linear in the size of the equivalent formula. Conversely, if we consider formulas with sharing of sub-formulas, we can also build an equivalent circuit of size linear in the size of the formula.

2.4 Direct compilation scheme

In the previous section, we have seen the relationship between vectorial circuit classes and LTL logic formulas. However, the proofs of those results do not give satifying algorithms for building a vectorial circuit recognizing a given formula. Indeed, the proofs going from the automata to the formulas are either indirect and not constructive, and they do not give any upperbound on the sizes of the formulas, or they are constructive but give formulas (and thus circuits) of too large sizes. Our goal is to reduce the size of the formulas we obtain through semigroups, so that we can obtain tractable algorithms for the languages that have a small syntactic monoid. Because of this point of view, all complexity measures of our circuits are provided in terms of the semigroup size.

▶ **Theorem 5.** *Let S be a semigroup in* **DA** *of $\mathcal{J}$-depth d (see Section 3.1 for the definition of $\mathcal{J}$-depth). We can construct a* Sweeping*-vectorial circuit of size $O(2^d|S|^2)$ that computes the operation π_S.*

Moving from Sweeping-vectorial circuits to Unary-LTL could cost an exponential blowup, which in total gives a doubly-exponential blowup for constructing a Unary-LTL formula from a semigroup in **DA**. Note that the classical constructions do not provide upperbounds on the minimum size of a Unary-LTL formula equivalent to a given semigroup in **DA**. However, these constructions already have considerable sizes of formulas.

We conjecture that no Sweeping-vectorial circuit of polynomial size exists and believe that it is an interesting open question to provide lowerbounds for those circuit models.

▶ **Theorem 6.** *Let S be an aperiodic semigroup of $\mathcal{J}$-depth d. We can construct an* ADD*-vectorial circuit of size $O(d|S|^3)$ that computes the operation π_S.*

Proofs' organisation. We chose to separate the proofs into two main parts. In the first one, we introduce the notion of evaluation program, which is a very generic tool allowing to replace a sub-word by a single letter equal to its product, and we define two generic evaluation strategies which are evaluation programs. These strategies were chosen according to the properties of the classes of semigroup that will be considered later but, supposing that we can provide an efficient encoding of those strategies, they are valid on any semigroup. Then, in the second part, we provide vectorial circuits encoding the two evaluation strategies on the classes of semigroups that we are interested in.

3 Semigroups evaluation strategies

3.1 Green's relations

We refer the reader to [24, 23] for a complete exposition of algebraic automata theory. We remind here some more basic notations and definitions about semigroups. Consider a function $F\colon S \to \mathcal{P}(S)$, where $\mathcal{P}(S)$ denotes the power set of S. We write $x \mathcal{F} y$ when $F(x) = F(y)$; $x \leq_{\mathcal{F}} y$ when $F(x) \subseteq F(y)$; and $x <_{\mathcal{F}} y$ when $x \leq_{\mathcal{F}} y$ and $F(x) \neq F(y)$. The relation $\mathcal{F}$ is an equivalence relation and $\leq_{\mathcal{F}}$ is a partial pre-order. We also write $\mathcal{F}(x) = \{y \mid y \mathcal{F} x\}$, the *$\mathcal{F}$-class of x*. We say that a semigroup is *$\mathcal{F}$-trivial* when $\mathcal{F}(x)$ is a singleton for any element $x \in S$. Green's relations are defined with the following functions: $R\colon x \mapsto x \cdot_S S^1$, $L\colon x \mapsto S^1 \cdot_S x$, $J\colon x \mapsto S^1 \cdot_S x \cdot_S S^1$, $H\colon x \mapsto R(x) \cap L(x)$. Note that the respective relations obtained from R, L, J and H are denoted by $\mathcal{R}$, $\leq_{\mathcal{R}}$, $<_{\mathcal{R}}$, $\mathcal{L}$, $\leq_{\mathcal{L}}$, $<_{\mathcal{L}}$, $\mathcal{J}$, $\leq_{\mathcal{J}}$, $<_{\mathcal{J}}$,$\mathcal{H}$, $\leq_{\mathcal{H}}$ and $<_{\mathcal{H}}$. In finite semigroups, the relation $\mathcal{J}$ is equal to the relation $\mathcal{D}$, which is the union of $\mathcal{L}$ and $\mathcal{R}$. From this relation $\mathcal{D}$ comes the name of the class **DA**, which indicates the class of semigroups which regular $\mathcal{D}$-classes are aperiodic semigroups.

The $\mathcal{J}$-depth of a semigroup. Let S be a semigroup. The *$\mathcal{J}$-depth* of a $\mathcal{J}$-class is the length of a maximal strictly decreasing sequence of $\mathcal{J}$-classes to it. Formally, given a semigroup S and a $\mathcal{J}$-class J, we say that J is of $\mathcal{J}$-depth i if there exist i $\mathcal{J}$-classes $J_1 >_{\mathcal{J}} J_2 \ldots >_{\mathcal{J}} J_i$ such that $J_i = J$, but there exists no decreasing sequence $J'_1 >_{\mathcal{J}} J'_2 \ldots >_{\mathcal{J}} J'_{i+1}$ such that $J'_{i+1} = J$. The $\mathcal{J}$-depth of a semigroup is the maximum $\mathcal{J}$-depth of its $\mathcal{J}$-classes. For any semigroup, there exists a unique $\mathcal{J}$-class of maximum $\mathcal{J}$-depth. Given d the $\mathcal{J}$-depth of S, for each integer i such that $1 \leq i \leq d$, we denote by $D_i(S)$ the union of all the $\mathcal{J}$-classes of depth i and we denote by S_i the sub-semigroup composed exactly of all the elements of S of $\mathcal{J}$-depth at least i.

$\mathcal{J}$-constant words. Let S be a semigroup. A word $s_0 \cdots s_k$ in S^+ is *left $\mathcal{J}$-constant* if, for any index i such that $0 \leq i \leq k$, we have $\pi_S(s_0 \cdots s_i) \mathcal{J} s_0$. Symetrically, $s_0 \cdots s_k$ is *right $\mathcal{J}$-constant* if, for any index i such that $0 \leq i \leq k$, $\pi_S(s_i \cdots s_k) \mathcal{J} s_k$. Finally, a word in S^+ is $\mathcal{J}$-constant if it is both left and right $\mathcal{J}$-constant. The latter property is equivalent to having a left $\mathcal{J}$-constant word such that $s_0 \mathcal{J} s_k$.

3.2 Evaluation programs

In this paper, we consider words over some semigroup S. Our goal is to compute the product in S of the letters composing these words. For any word $u \in S^+$, this amounts to computing $\pi_S(u)$. This computation can be performed by vectorial circuits. Instead of directly building these circuits, we first pay attention to evaluation strategies that we call *evaluation programs*. These strategies form an overlay of abstraction over the intricacy of circuits. They are meant to modularize the construction of vectorial circuits.

Given a semigroup S, an evaluation program over S transforms words in S^+ by replacing some of the factors by their values (through the canonical morphism) in S. In this section, we consider a fixed semigroup S.

▶ **Definition 7** (Partial evaluation). *A* partial evaluation step *over S is a relation over S^+ denoted by $\to_S$ and defined as $uvw \to_S u\pi_S(v)w$ for any v in S^+ and $u, w \in S^*$. We denote by $\to_S^+$ the transitive closure of $\to_S$. We say that v* is a partial evaluation of *u when $u \to_S^+ v$.*

Note that if v is a partial evaluation of u over S, then $\pi_S(u) = \pi_S(v)$. Usually, the context makes it clear which semigroup is considered. Thus, we generally leave the semigroup implicit and only say that v is a partial evaluation of u. Note that each word u is a partial evaluation of itself.

▶ **Definition 8** (Partial evaluation programs). *A* partial evaluation program *over S is a partial function $P : S^+ \to S^+$ such that, for any word $u \in S^+$ that is in the domain of P, $P(u)$* is a partial evaluation of *u. If the domain of P is S^+, then we say that P is* total.

The function π_S is an example of a partial evaluation (which is, in this case, total). Another example is the function $\mathrm{LProd}_S \colon S^+ \to S^+$ that performs the product of the first two elements of the input, if there are at least two elements, and otherwise returns the input word. In symbols, it is defined by $\mathrm{LProd}_S(s_0 \cdots s_n) = \pi_S(s_0 s_1) s_2 \cdots s_n$ for any word $u = s_0 \cdots s_n$ of length at least two, and $\mathrm{LProd}_S(s_0) = s_0$ for any element $s_0 \in S$. Similarly, we define RProd_S as the partial evaluation program which performs the product of the two last elements, if there are at least two elements, and otherwise returns the input word. Evaluation programs are closed under composition.

3.3 Waterfall evaluation programs

In this section, we are designing a first set of specific evaluation programs. These programs work by evaluating the semigroup in a top-down fashion (with respect to the $\mathcal{J}$-order). We first detect each maximal subword whose product is maximal for the $\mathcal{J}$-order, evaluate those subwords and multiply the results with the letters that immediately follow.

▶ **Definition 9** ($\mathcal{J}$-maximal falling words). *A word u in S^+ is* $\mathcal{J}$-maximal falling *whenever for every $p, s \in S^*$ and $v, w \in S^+$ so that $u = pvws$, we have $\pi_S(vw) \in S_2$.*

Note that when u is $\mathcal{J}$-maximal falling and $|u| > 1$, $\pi_S(u) \in S_2$.

▶ **Definition 10** ($\mathcal{J}$-maximal decomposition). *Consider a word $u \in S^+$. If $u \notin S_2^+$, let $t \in \mathbb{N}$ and some words $w_0, \ldots, w_{t+1} \in S^*$, $v_0, \ldots, v_t \in S^+$, that define a decomposition $u = w_0 v_0 w_1 \cdots v_t w_{t+1}$. This decomposition is called* $\mathcal{J}$-maximal *if we have*

- *for any integer i such that $0 \leq i \leq t+1$, w_i is a word in S_2^**
- *for any integer i such that $1 \leq i \leq t$, v_i is a maximal subword of u verifying $\pi_S(v_i) \in D_1(S)$. More formally, if we consider the decomposition $u = pav_ibs$, with $p, s \in S^*$, and $a, b \in S \cup \epsilon$ (a is the letter preceding v_i, if it exists, and b is the letter following v_i, if it exists) then, if $a \neq \epsilon$, $\pi_S(av_i) <_{\mathcal{J}} \pi_S(v_i)$ and, if $b \neq \epsilon$, $\pi_S(v_ib) <_{\mathcal{J}} \pi_S(v_i)$*

We call $t+1$ the size of the decomposition.

If $u \in S_2^+$, the $\mathcal{J}$-maximal decomposition of u is composed of only one element equal to u itself. In this case, the decomposition is unique and by convention of size 0.

▶ Remark 11. The property of the (v_i)'s implies that each v_i is a word of $D_1(S)^+$. Indeed, the product of a word is at most $\mathcal{J}$-equivalent to the letter of greatest $\mathcal{J}$-depth, so all the letters must be of $\mathcal{J}$-depth 1 for the product to be in $D_1(S)$.

We will use the following useful technical lemma.

▶ **Lemma 12.** *Let S be a semigroup. Let x, y be $\mathcal{J}$-equivalent elements of S and z another element of S.*

- *If $\pi_S(xy) \mathcal{J} x$ and $\pi_S(zxy) <_{\mathcal{J}} x$, then $\pi_S(zx) <_{\mathcal{J}} x$.*
- *If $\pi_S(xy) \mathcal{J} x$ and $\pi_S(xyz) <_{\mathcal{J}} x$, then $\pi_S(yz) <_{\mathcal{J}} x$.*

We are now proving the existence and uniqueness of $\mathcal{J}$-maximal decomposition. The proof works by considering a variant of $\mathcal{J}$-maximal decomposition by enforcing the maximality constraint only at the right and showing that the two variants are actually equivalent.

▶ **Lemma 13.** *Let S be a semigroup. For any finite word $u \in S^+$, there exists a unique $\mathcal{J}$-maximal decomposition of u.*

Proof. We focus on proving the existence of such a decomposition. Unicity follows from the proof of existence. First, note that $S = D_1(S) \uplus S_2$. Thus, for any word $u \in S^+$, there necessarily exists a unique decomposition of u of the form $u = w_0v_0w_1 \cdots w_tv_tw_{t+1}$ such that $w_0, w_{t+1} \in S_2^*$, for each integer i such that $1 \leq i \leq t$, $w_i \in S_2^+$, and, for each $i \in [t+1]$, $v_i \in D_1(S)^+$. Informally, this is a decomposition of u based only on the $\mathcal{J}$-depth of each letter. Thanks to Remark 11, we know that the words w_i that are not empty in any $\mathcal{J}$-maximal decomposition of u correspond exactly to the words w_i of this decomposition. Thus, we only need to prove that any word over $D_1(S)$ can be decomposed into a unique sequence of maximal subwords whose product is in $D_1(S)$.

Consider a fixed word $u \in D_1(S)^+$. We prove that there exists a decomposition of u of the form $u = v_0 \cdots v_s$, where each word v_i is maximal in u such that $\pi_S(v_i) \in D_1(S)$.

If $\pi_S(u) \in D_1(S)$, then both existence and unicity follow from the definition.

Suppose now that $\pi_S(u) \in S_2$. Then we define a decomposition of u with weaker properties than the $\mathcal{J}$-maximal decompositions, and we prove that it is a $\mathcal{J}$-maximal decomposition of u (with empty words as the (w_i)'s). This decomposition is of the form $u = v_0v_1 \cdots v_t$ for some integer $t \in \mathbb{N}$ and is defined from left to right such that, for each integer $i \in [t+1]$, $\pi_S(v_i) \in D_1(S)$ and, for each integer $i \in [t]$, $\pi_S(v_ix_{i+1}) \in S_2$, where x_{i+1} is the first letter of v_{i+1}. By construction, this decomposition exists and is unique. We prove that this decomposition satisfies the properties of the (v_i)'s given in Definition 10, that is, we prove that each word v_i is maximal such that $\pi_S(v_i) \in D_1(S)$. To do that, we only need to prove that, for each integer i such that $1 \leq i \leq t$, $\pi_S(y_{i-1}v_i) \in S_2$, where y_{i-1} is the last letter of the word v_{i-1}.

Thus, consider some integer $i \in [t]$. We focus on the subword v_iv_{i+1} in order to prove that $\pi_S(y_iv_{i+1}) \in S_2$, where y_i is the last letter of v_i. This fact is a consequence of Lemma 12. Since we know that $\pi_S(v_ix_{i+1}) \in S_2$, where x_{i+1} is the first letter of v_{i+1}, and that $\pi_S(v_i) \in D_1(S)$, Lemma 12 implies that $\pi_S(y_ix_{i+1}) \in S_2$. Thus, $\pi_S(y_iv_{i+1}) \in S_2$. Thus, this decomposition is the unique $\mathcal{J}$-maximal decomposition of u. ◀

Now, we need to refine the $\mathcal{J}$-maximal decomposition. Keeping the same notation for u, and its $\mathcal{J}$-maximal decomposition $u = w_0v_0 \cdots v_tw_{t+1}$, we set, for any $0 \leq i \leq t$, $v_i = x_iv_i'$ ($x_i \in S$ is the first letter of v_i), $w_i' = w_ix_i$ and $w_i' = y_iw_i''$ ($y_i \in S$ is the first letter of w_i'). The evaluation program Collapse_S takes a word $u \in S^+$ and performs at once all the products $\pi_S(v_i'y_{i+1})$, the operation $\pi_S(v_t')$ if $w_{t+1} = \epsilon$, or $\pi_S(v_t'y_{t+1})$ otherwise, with $w_{t+1}' = y_{t+1}w_{t+1}$ ($y_{t+1} \in S$ is the first letter of w_{t+1}). In symbols:

$\text{Collapse}_S(u) = w_0'\pi_S(v_0'y_1)w_1'' \cdots \pi_S(v_i'y_{i+1})w_{i+1}'' \cdots w_t''z$, where z denotes either $\pi_S(v_t')$ if w_{t+1} is the empty word, or $\pi_S(v_t'y_{t+1})w_{t+1}'$. The image of any word $u \in S^+$ by Collapse_S is a $\mathcal{J}$-maximal falling word.

Given an element $s \in D_1(S)$, we also define the partial evaluation program $\text{Falling}_S(s)$ which takes a word $u \in S^+$ which is a $\mathcal{J}$-maximal falling word such that the last letter of u is in S_2, and returns a partial evaluation of u in which there is no occurrence of s. Given a word u, we call s-decomposition of u the decomposition of the form $u = w_0s^{k_0}x_0w_1 \cdots s^{k_t}x_tw_{t+1}$ where the k_i's are positive integers, the x_i's are non-s elements of S, except for x_t which

can also be equal to ϵ if $w_{t+1} = \epsilon$, and the w_i's are words without any occurrence of s. The s-decomposition of a word always exists and is unique. Given the s-decomposition of some word $u \in S^+$, keeping the same notation, we define

$$\mathrm{Falling}_S(s)(u) = w_0\pi_S(s^{k_1}x_1)\cdots\pi_S(s^{k_t}x_t)w_t$$

▶ **Lemma 14.** *Let S be a semigroup of $\mathcal{J}$-depth d. The evaluation program π_S is equal to the composition of $O(|S|)$ evaluation programs among* RProd_S, $\mathrm{Collapse}_{S_i}$ *and* $\mathrm{Falling}_{S_i}(s)$ *for all $1 \leq i \leq d$ and $s \in D_1(S_i)$.*

▶ Remark 15. Waterfall evaluation programs have some ressemblance with the *factorizations forest* of Simon [3]. Indeed, our programs create a factorization forest for each word they are applied to. Moreover, the proof of the factorization forests theorem uses an induction on $\mathcal{J}$-classes, as we do for our programs. However, they are not quite the same. A waterfall evaluation program can be applied to any word on the right alphabet, whereas the factorization forests theorem proves the existence of a forest of bounded depth for a fixed word. This theorem is used to prove the existence of a formula corresponding to a given semigroup in two cases: the classes $\mathcal{B}\Sigma_1$ and Σ_1. To our knowledge, there are no proofs for the class **DA** or the class of aperiodic semigroups. Lastly, such proofs amount to consider all the formulas of some quantifier depth that depends on the forest depth, a technique that resembles Wilke's proof for **DA** [35, Corollary 1] and also gives awful upper bounds.

3.4 Sweeping evaluation programs

We introduce an evaluation program which processes words in a more lateral fashion – from left to right or right to left. In this subsection, S is a semigroup of $\mathcal{J}$-depth d.

In the proofs of Section 4, we will perform evaluations that produce left (resp. right) $\mathcal{J}$-constant prefixes (resp. suffixes). We define those programs depending on the $\mathcal{J}$-depth of the semigroup that is considered. First, we introduce a *left splitting* higher order operation that applies an evaluation program over a prefix of the input word defined by some constraints over Green's relations. Formally, for any integer $i \leq d$, we define the operation $\mathrm{LSplit}_{S,i}$ as follows. Consider a word $u = s_0 \cdots s_k \in S^+$ and an evaluation program P defined at least on all left $\mathcal{J}$-constant words of depth i. If s_0 is not of $\mathcal{J}$-depth i, we set $\mathrm{LSplit}_{S,i}\langle P\rangle(u) = u$. Otherwise, there exist two uniquely defined words $p \in S^+$, $s \in S^*$ such that $u = ps$, where p is left $\mathcal{J}$-constant and either s is empty or, if we denote by $x \in S$ its first letter, $\pi_S(px) <_{\mathcal{J}} \pi_S(p)$. In this case, $\mathrm{LSplit}_{S,i}\langle P\rangle(u) = P(p)s$. We define similarly the symmetric operation $\mathrm{RSplit}_{S,i}$. Finally, we define the partial function JProd_S, that is the restriction of π_S over words that are $\mathcal{J}$-constant. Formally, JProd_S is defined only on $\mathcal{J}$-constant words and, given $u \in S^+$ such a word, $\mathrm{JProd}_S(u) = \pi_S(u)$.

A *sweeping evaluation program* is an evaluation program built with the following operations: LProd_S, RProd_S, the partial function JProd_S, and the higher order operations $\mathrm{LSplit}_{S,i}$ and $\mathrm{RSplit}_{S,i}$ for any integer i such that $1 \leq i \leq d$.

Example. Consider a semigroup S and the sweeping evaluation program $P = \mathrm{LProd}_S \circ \mathrm{LSplit}_{S,1}\langle \mathrm{JProd}_S\rangle$. The program P first executes the operation $\mathrm{LSplit}_{S,1}\langle \mathrm{JProd}_S\rangle$. To do so, it begins by looking at the beginning of the input word w. If the first letter of w is not in $D_1(S)$, then P computes only $\mathrm{LProd}(w)$, which is the word obtained by multiplying the two first letters of w. Otherwise, P decomposes w into two words $p \in D_1(S)^+$, $s \in S^*$ such that $u = ps$ and, if s is not the empty word, $\pi_S(px) \in S_2$, where x is the first letter of s. Then, P applies the operation JProd_S to the prefix p. Then, the result of $\mathrm{LSplit}_{S,1}\langle \mathrm{JProd}_S\rangle(w)$ is the word $\pi_S(p)s$. Finally, P applies the operation LProd_S to the previous result, multiplying the first two letters of $\pi_S(p)s$, if s is not empty.

▶ **Lemma 16** (Sweeping evaluation programs). *Let S be a semigroup. There exists a sweeping evaluation program computing π_S. Moreover, there exists such a program that is equal to the composition of $O(2^d)$ operations.*

4 Proof of the main results

We recall the proof strategy. Given a regular language which is aperiodic (resp. in **DA**), we prove that we can compute π_S with a ADD- (resp. Sweeping-) vectorial circuit. To build such circuits, we rely on respectively waterfall and sweeping evaluations programs. As the considered vectorial circuit classes are closed under functional composition, it is sufficient to prove that every basic operation in our evaluation programs is computable with the desired vectorial circuit class. We provide those arguments in this section and the corresponding section of the appendix.

Vectorial encoding of a partial evaluation of a word. Our sweeping and waterfall evaluation programs perform operations on partial evaluations of a word, so we need to provide an explanation on how we encode these partial evaluations. From now on, a partial evaluation will always designate a partial evaluation of a word, usually the initial word that needs to be processed with the evaluation program. Informally, a vectorial representation of a partial evaluation is a set of vectors, each vector corresponding to a semigroup element. The size of the vectors is equal to the size of the initial word on which we apply the evaluation program. We need the vectors we employ to always have the same size throughout the execution, so our definition needs to be more general than indicator vectors. Each bit set to one denotes the presence of the element in the partial evaluation, and the order of the bits set to one determines the order of the letters in the word. As in the case of characteristic functions of letters, in the vector encoding of a word, vectors in an encoding do not overlap, however their union may not cover all the positions. More formally, a *vectorial encoding of a partial evaluation* is a mapping $\mathbf{c}\colon S \to \{0,1\}^n$, for some integer $n \geq 1$, such that we have the following constraint: for any $s, t \in S$ such that $s \neq t$, we have $\mathbf{c}(s) \wedge \mathbf{c}(t) = \mathbf{0}$. Note that, by definition, a word is a partial evaluation of itself, so $\mathrm{enc}(u)$ is a vectorial encoding of a partial evaluation, with the additional property that $\bigvee_{a \in S} \mathbb{1}_a(u) = \mathbf{1}$.

Given such a function $\mathbf{c}$ outputting vectors of dimension n, we can interpret it as a word of length at most n by respecting the order of appearance of the bits. Formally, a word $s_0 \cdots s_k \in S^{\leq n}$ is represented by a vectorial encoding $\mathbf{c}$ of dimension n if, for every index $j \leq k$, there exists some integer i such that $0 \leq i < n$, $\mathbf{c}(s_j)_i = 1$ and $|\{t \in \mathbb{N} \mid t < i \text{ and } \bigvee_{s \in S} \mathbf{c}(s)_t = 1\}| = j - 1$.

In this section, we use only circuits of the following form: the input is a vectorial encoding of a partial evaluation $\mathbf{c}\colon S \to \{0,1\}^n$ (for some $n \in \mathbb{N}^+$) representing some word $u \in S^{\leq n}$, and the output is a vectorial encoding $\mathbf{out}\colon S \to \{0,1\}^n$ representing the partial evaluation of u obtained by applying the operation corresponding to the circuit. To prove our theorems, we construct vectorial circuits using this encoding that implement the partial evaluation functions we presented earlier.

Example. For example, consider the semigroup $S = \{a, b, c\}$ with the inner multiplication as follows: $\pi_S(ac) = \pi_S(ca) = c$, $\pi_S(bc) = \pi_S(cb) = c$, $\pi_S(aa) = b$. The word $w = aaabac$ has the vectorial encoding $\mathbf{a} = 111010$, $\mathbf{b} = 000100$, $\mathbf{c} = 000001$. With our circuits, if we multiply the last two letters of w using this encoding, we will obtain the encoding $\mathbf{a} = 111000$, $\mathbf{b} = 000100$, $\mathbf{c} = 000001$. As expected, it represents the word $w' = aaabc$: the first four

elements of the vectors are the same and still represent the word $aaab$, the fifth element is a 0 in all vectors so it represents no letter, and the sixth element is a 1 in $\mathbf{c}$, so it represents the letter c. Since we started from a word of length 6, the length of the vectors is still 6 but some of the indices do not represent a letter anymore.

Sketch of proof of Theorem 6. To prove Theorem 6, we consider a fixed aperiodic semigroup S, and we denote by d its $\mathcal{J}$-depth. Thanks to Lemma 14, it is sufficient to provide ADD-vectorial circuits computing the operations $\mathrm{Collapse}_S$, RProd_S and $\mathrm{Falling}_S(s)$ (for any aperiodic semigroup S and $s \in D_1(S)$) over some vectorial encoding of any partial evaluation of a word. Once we have those, we proceed as in Lemma 14, in which we prove that, for any integer i such that $1 \leq i \leq d$, π_{S_i} is equal to the composition of $\pi_{S_{i+1}}$ and $O(|S|)$ operations among $\mathrm{Collapse}_{S_i}$, RProd_S and $\mathrm{Falling}_S(s)$ (for any $s \in D_i(S)$). More precisely, the evaluation program for π_S uses $|S|$ operations of the form $\mathrm{Falling}_{S'}(s)$ (where $s \in S$), d operations of the form RProd_S and d operations of the form $\mathrm{Collapse}_{S_i}$ (recall that d is the $\mathcal{J}$-depth of S). Now, we can obtain the result by using the following lemmas:

▶ **Lemma 17.** *For any aperiodic semigroup S, we can compute* $\mathrm{Collapse}_S$ *over any vectorial encoding of a partial evaluation with an* ADD*-vectorial circuit of size $O(|S|^3)$.*

▶ **Lemma 18.** *For any aperiodic semigroup S, we can compute* RProd_S *on any vectorial encoding of a partial evaluation of a word over S with an* ADD*-vectorial circuit of size $O(|S|^2)$.*

▶ **Lemma 19.** *Let S be an aperiodic semigroup of $\mathcal{J}$-depth d. For any element $s \in D_1(S)$, we can compute* $\mathrm{Falling}_S(s)$ *over any vectorial encoding of a partial evaluation in its domain with an* ADD*-vectorial circuit of size $O(d|S|)$.*

With this, we can conclude that π_S can be computed with an ADD-vectorial circuit of size $O(d|S|^3)$.

Sketch of proof of Theorem 5. To prove Theorem 5, we consider a fixed semigroup $S \in \mathbf{DA}$, and we denote by d its $\mathcal{J}$-depth. Thanks to Lemma 16, it is sufficient to provide Sweeping-vectorial circuits computing the base operations over some vectorial encoding of any partial evaluation of a word. Those operations are LProd_S, RProd_S, JProd_S and, for each integer i such that $1 \leq i \leq d$, the operations $\mathrm{LSplit}_{S,i}$ and $\mathrm{RSplit}_{S,i}$. Once we have those, we proceed as in Lemma 16, in which we prove that π_S is equal to the composition of $O(2^d)$ operations among JProd_S, LProd_S, RProd_S and $\mathrm{LSplit}_{S,i}\langle E\rangle$ (for any integer i such that $1 < i \leq d$ and some sweeping program E composed of the same operations, that are taken into account in the $O(2^d)$). To prove this, we have $\pi_S = P_{d,l} = P_{d,r}$ where, for each integer i such that $1 \leq i \leq d$, the program $P_{i,l}$ computes π_S on the maximal prefix of $\mathcal{J}$-depth at most i (included), and the symmetric program $P_{i,r}$ which acts on suffixes instead of prefixes. Our proof constructs Sweeping-vectorial circuits for those programs, by induction on the depth i. Now, we can obtain the result by using the following lemmas:

▶ **Lemma 20.** *For any semigroup $S \in \mathbf{DA}$, we can compute* JProd_S *over any vectorial encoding of a partial evaluation in its domain with a* Sweeping*-vectorial circuit of size $O(|S|^2)$.*

▶ **Lemma 21.** *For any semigroup $S \in \mathbf{DA}$, we can compute* LProd_S *and* RProd_S *over any vectorial encoding of a partial evaluation in their domains with* Sweeping*-vectorial circuits of size $O(|S|^2)$.*

▶ **Lemma 22.** *Let S be a semigroup in* **DA** *of $\mathcal{J}$-depth d, i be an integer such that $1 \leq i \leq d$, let P be a sweeping evaluation program defined at least on all left $\mathcal{J}$-constant words of depth i, and suppose that we have a* Sweeping*-vectorial circuit of size s_P that computes P over any vectorial encoding of a partial evaluation. Then we can compute* $\mathrm{LSplit}_{S,i}\langle P\rangle$ *and* $\mathrm{RSplit}_{S,i}\langle P\rangle$ *over any vectorial encoding of a partial evaluation in their respective domains with* Sweeping*-vectorial circuits of size $O(|S|^2 + s_P)$.*

Thus, following the construction in the proof of Lemma 16, for each integer i such that $1 \leq i < d$, we have a Sweeping-vectorial circuit computing $P_{i+1,l}$ (or $P_{i+1,r}$, these are symmetrical operations) given some circuits computing $P_{i,l}$ and $P_{i,r}$, and its size is equal to $O(|S|^2 + 2c_i)$, where c_i is the size of the circuits computing $P_{i,l}$ and $P_{i,r}$. Since, by the construction of Lemma 16, we also have $c_1 = O(|S|^2)$, we then have $c_i = (2^i - 1)|S|^2$ for each integer i such that $1 < i \leq d$. Thus, the circuit computing $\pi_S = P_{d,l}$ is of size $O(2^d|S|^2)$.

5 Conclusion

We introduce vectorial circuits as abstractions of low level hardware architectures. As circuits, they put forward dependencies between steps of computation and thus opportunities of parallelism. Taking stock on previous work [16, Theorem 1], the next step is then to adapt these circuits to a streaming context. This streaming setting is closer to text processing problems as only small chunks of input data can hold at the level of CPUs. We shall then study how to take advantage of SIMD instructions to implement these machines. The vast number of SIMD instructions give many possibilities to compile vectorial circuits for streaming text. The challenge is then to find the right set of instructions, combine them sufficiently well together and obtain efficient programs. Here again, we hope that algebra can back our efforts up.

Concerning the circuits themselves, we can consider extensions with operations that have natural correspondence in CPU instructions such as *shifts*, *prefix-xor*, etc. The question is then to understand what classes of regular languages we can describe. Hopefully, the combinations of particular sets of operations could correspond to well-studied algebraic operations [27]. Instead of operations, we can also consider the use of arbitrary constants, (i.e. particular vectors that can be used as gates inputs). We think that using constants in circuits can be related to that of arbitrary monadic advices [7].

In the paper, we limit ourselves to consider vectorial circuits as recognizers. As these circuits produce outputs, they should be more adequately viewed as transducers. On the theoretical side, there is a need to explore their expressivity. This requires an adaptation of the algebraic tools we use, i.e. an understanding of classes of logical transducers. On the practical side, in the context of streaming, this point of view calls for composing stream processing programs. This probably requires to explore ideas from synchrone programming in combination with vectorial circuits.

This paper also calls for more foundational work concerning circuit complexity. In particular we conjecture that the bounds provided in Theorem 5 are tight. However, finding an adequate lower bound is a challenging open problem.

References

1 Anne Bergeron and Sylvie Hamel. Cascade decompositions are bit-vector algorithms. In *International Conference on Implementation and Application of Automata*, pages 13–26. Springer, 2001.

2 Anne Bergeron and Sylvie Hamel. Vector algorithms for approximate string matching. *International Journal of Foundations of Computer Science*, 13(01):53–65, 2002.

3 Mikolaj Bojańczyk. Factorization forests. In *International Conference on Developments in Language Theory*, pages 1–17. Springer, 2009.

4 Robert D Cameron, Thomas C Shermer, Arrvindh Shriraman, Kenneth S Herdy, Dan Lin, Benjamin R Hull, and Meng Lin. Bitwise data parallelism in regular expression matching. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 139–150. IEEE, 2014.

5 V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *Internat. J. Found. Comput. Sci.*, 19:513–548, 2008.

6 Michael Farrar. Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23(2):156–161, November 2006.

7 Nathanaël Fijalkow and Charles Paperman. Monadic second-order logic with arbitrary monadic predicates. *ACM Transactions on Computational Logic (TOCL)*, 18(3):1–17, 2017.

8 Tobias Grosser, Hongbin Zheng, Raghesh Aloor, Andreas Simbürger, Armin Größlinger, and Louis-Noël Pouchet. Polly-polyhedral optimization in LLVM. In *Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT)*, volume 2011, page 1, 2011.

9 Johan Anthony Wilem Kamp. *Tense logic and the theory of linear order*. University of California, Los Angeles, 1968.

10 John Keiser and Daniel Lemire. Validating UTF-8 in less than one instruction per byte. *Software: Practice and Experience*, 51(5):950–964, 2021.

11 Donald Ervin Knuth. The art of computer programming: Bitwise tricks & techniques. *Binary Decision Diagrams*, 4, 2009.

12 M Oguzhan Külekci. Filter based fast matching of long patterns by using SIMD instructions. In *Stringology*, pages 118–128, 2009.

13 Leslie Lamport. Multiple byte processing with full-word instructions. *Communications of the ACM*, 18(8):471–475, 1975.

14 Geoff Langdale and Daniel Lemire. Parsing gigabytes of JSON per second. *The VLDB Journal*, 28(6):941–960, 2019.

15 McNaughton, Robert, Papert, and Seymour A. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

16 Filip Murlak, Charles Paperman, and Michal Pilipczuk. Schema validation via streaming circuits. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 237–249, 2016.

17 Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)*, 46(3):395–415, 1999.

18 Dorit Nuzman, Sergei Dyshel, Erven Rohou, Ira Rosen, Kevin Williams, David Yuste, Albert Cohen, and Ayal Zaks. Vapor SIMD: Auto-vectorize once, run everywhere. In *International Symposium on Code Generation and Optimization (CGO 2011)*, pages 151–160. IEEE, 2011.

19 Dorit Nuzman and Ayal Zaks. Autovectorization in GCC–two years later. In *Proceedings of the 2006 GCC Developers Summit*, volume 6, 2006.

20 Dorit Nuzman and Ayal Zaks. Outer-loop vectorization: revisited for short SIMD architectures. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 2–11, 2008.

21 Charles Paperman, Sylvain Salvati, and Claire Soyez-Martin. Addition Lemma, September 2022. URL: `https://hal.archives-ouvertes.fr/hal-03787033`.

22 Charles Paperman, Sylvain Salvati, and Claire Soyez-Martin. An algebraic approach to vectorial programs. Complete version of the paper, January 2023. URL: `https://hal.archives-ouvertes.fr/hal-03831752v2`.

23 J.E. Pin and European Mathematical Society Publishing House ETH-Zentrum SEW A27. *Handbook of Automata Theory: Volume I: Theoretical Foundations; Volume II: Automata in Mathematics and Selected Applications*. EMS Press, 2021.

24 Jean Eric Pin. *Varieties of formal languages*, volume 184. Springer, 1986.

25 M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and control*, 8:190–194, 1965.

26 Olivier Serre. Vectorial languages and linear temporal logic. *Theoretical computer science*, 310(1-3):79–116, 2004.

27 Howard Straubing. Finite semigroup varieties of the form V*D. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

28 Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Springer Science & Business Media, 2012.

29 Pascal Tesson and Denis Thérien. Diamonds are forever: The variety DA. In *Semigroups, algorithms, automata and languages*, pages 475–499. World Scientific, 2002.

30 Denis Therien and Pascal Tesson. Logic meets algebra: the case of regular languages. *Logical Methods in Computer Science*, 3, 2007.

31 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 234–240, 1998.

32 Konrad Trifunovic, Dorit Nuzman, Albert Cohen, Ayal Zaks, and Ira Rosen. Polyhedral-model guided loop-nest auto-vectorization. In *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, pages 327–337. IEEE, 2009.

33 Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. Hyperscan: A fast multi-pattern regex matcher for modern CPUs. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 631–648, 2019.

34 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. In *International Workshop on Computer Science Logic*, pages 343–357. Springer, 2007.

35 Thomas Wilke. Classifying discrete temporal properties. In *Annual symposium on theoretical aspects of computer science*, pages 32–46. Springer, 1999.

36 James Worrell, Rastislav Lenhardt, and Michael Benedikt. Two variable vs. linear temporal logic in model checking and games. *Logical Methods in Computer Science*, 9, 2013.

A Proofs for Section 3 (Semigroups evaluation strategies)

Every word in S^+ admits a unique $\mathcal{J}$-maximal decomposition of a word. This property hinges on the Localisation Theorem of Clifford and Miller [24, Proposition 1.6, page 48].

▶ **Lemma 23** (Localisation Theorem). *Let S be a semigroup and x, y be in S. We have $xy\mathcal{J}x$ if and only if there exists an idempotent e in $R(y) \cap L(x)$.*

▶ **Lemma 12.** *Let S be a semigroup. Let x, y be $\mathcal{J}$-equivalent elements of S and z another element of S.*

- *If $\pi_S(xy)\mathcal{J}x$ and $\pi_S(zxy) <_{\mathcal{J}} x$, then $\pi_S(zx) <_{\mathcal{J}} x$.*
- *If $\pi_S(xy)\mathcal{J}x$ and $\pi_S(xyz) <_{\mathcal{J}} x$, then $\pi_S(yz) <_{\mathcal{J}} x$.*

Proof. Both cases are symmetric, so we only prove the first one. Suppose that $\pi_S(zx)\mathcal{J}x$, $\pi_S(xy)\mathcal{J}x$ and $\pi_S(zxy) <_{\mathcal{J}} x$. Lemma 23 implies that there is no idempotent in $R(y) \cap L(zx)$. But, as $\pi_S(zx)\mathcal{J}x$, by definition of $\mathcal{L}$, we have $\mathcal{L}(zx) = \mathcal{L}(x)$. Therefore $R(y) \cap L(x)$ does not contain an idempotent. Finally Lemma 23 entails $xy <_{\mathcal{J}} x$, a contradiction. ◀

▶ **Lemma 14.** *Let S be a semigroup of $\mathcal{J}$-depth d. The evaluation program π_S is equal to the composition of $O(|S|)$ evaluation programs among* RProd_S, $\mathrm{Collapse}_{S_i}$ *and* $\mathrm{Falling}_{S_i}(s)$ *for all $1 \leq i \leq d$ and $s \in D_1(S_i)$.*

Proof. For any integer i such that $1 \leq i \leq d$, we define $O_i = (s_1, \ldots, s_k)$ to be any enumeration of $D_1(S_i)$. Given such an enumeration, we define the following operation:

$$\mathrm{Falling}_{S_i}[O_i] = \mathrm{Falling}_{S_i}(s_k) \circ \cdots \circ \mathrm{Falling}_{S_i}(s_1)$$

We define an intermediate partial evaluation program f_i which is a restriction of π_S to the domain $S_i^+ \cup S$. In symbols, for any word $u \in S_i^+ \cup S$, $f_i(u) = \pi_S(u)$. Note that f_1 is equal to π_S on any word of S^+. We prove by induction over $j = d - i$, with $i \in [d]$, that

$$f_j = f_{j+1} \circ \mathrm{Falling}_{S_j}[O_j] \circ \mathrm{RProd}_S \circ \mathrm{Collapse}_{S_j}$$

with the convention that f_{d+1} is the identity. The base case (f_{d+1}) being fixed, we only need to prove the result by induction on i for $0 \leq i \leq d-1$. We remark that the image of $\mathrm{Collapse}_{S_j}$ produces a $\mathcal{J}$-maximal falling word, and then the application of RProd_S guarantees that the last element is not in $D_1(S)$. Finally, the application of $\mathrm{Falling}_{S_i}(s_i)$ on the resulting elements produces new elements that are only in S_{i+1} and removes all occurrences of the element s_i. Hence by applying $\mathrm{Falling}_{S_i}[O_i]$ we obtain a word in S_{i+1}^*, which concludes the proof. ◀

▶ **Lemma 16** (Sweeping evaluation programs)**.** *Let S be a semigroup. There exists a sweeping evaluation program computing π_S. Moreover, there exists such a program that is equal to the composition of $O(2^d)$ operations.*

To prove Lemma 16, we introduce an intermediate operation. Let i be an integer such that $1 \leq i \leq d$. We denote by $P_{i,l}$ the *left sweeping evaluation program* of $\mathcal{J}$-depth i, which computes π_S on the maximal prefix of $\mathcal{J}$-depth at most i (included). Formally, given a word $u = s_0 \cdots s_k \in S^+$, $P_{i,l}(u)$ is equal to u if s_0 is of $\mathcal{J}$-depth strictly greater than i. Otherwise, there exist $p \in S^+$, $q \in S^*$ such that $u = pq$, where either q is empty, or $x \in S$ is its first letter and then $\pi_S(p)$ is of $\mathcal{J}$-depth at most i and $\pi_S(px)$ is of $\mathcal{J}$-depth strictly greater than i. In this case, $P_{i,l}(u) = \pi_S(p)q$. We define symmetrically $P_{i,r}$, the *right sweeping evaluation program* of $\mathcal{J}$-depth i.

The next lemma allows to conclude the proof of Lemma 16 since $\pi_S = P_{d,l} = P_{d,r}$.

▶ **Lemma 24.** *For any integer i such that $1 \leq i \leq d$, there exist sweeping evaluation programs computing $P_{i,l}$ and $P_{i,r}$.*

Proof of Lemma 24. We will prove by induction on the $\mathcal{J}$-depth i that we can implement a left (resp. right) sweeping evaluation program $P_{i,l}$ (resp. $P_{i,r}$). In this proof, we consider a word $u = s_0 \cdots s_{k-1}$ over S. For the base case, we first suppose that $i = 1$, i.e. we consider maximal $\mathcal{J}$-classes. Thus, if s_0 is of $\mathcal{J}$-depth 1, we will compute the product of the unique prefix $s_0 \cdots s_p$ of u such that $\pi_S(s_0 \cdots s_p)\mathcal{J}s_0$ and $\pi_S(s_0 \cdots s_{p+1}) <_{\mathcal{J}} s_0$. If s_{p+1} does not exist, we want to compute $\pi_S(u)$. Note that $s_0 \cdots s_p$ is $\mathcal{J}$-constant, hence we can apply JProd_S to it. Thus, we can compute the base case $P_{1,l}$ using the program $\mathrm{LSplit}_{S,1}\langle \mathrm{JProd}_S \rangle$. Note that this program is well defined since JProd_S is in particular defined on all $\mathcal{J}$-constant words of depth 1, which are exactly the left $\mathcal{J}$-constant words of depth 1. Symmetrically, $P_{1,r} = \mathrm{RSplit}_{S,1}\langle \mathrm{JProd}_S \rangle$. To prove the induction case, we will rely on the following fact (see 3.1 for the definition of a left $\mathcal{J}$-constant word):

▶ **Fact.** *For any left $\mathcal{J}$-constant word $v \in S^+$ of $\mathcal{J}$-depth i, the word* $\mathrm{RProd}_S \circ P_{i-1,r}(v)$ *is $\mathcal{J}$-constant.*

Proof. The result is obtained from the fact that the last element of $w = \mathrm{RProd}_S \circ P_{i-1,r}(v)$ is necessarily of $\mathcal{J}$-depth i. Indeed, by definition, the word $x = P_{i-1,r}(v)$ is such that the product of its last two elements (if there are at least two elements) is at least of $\mathcal{J}$-depth i. Since we supposed that v is left $\mathcal{J}$-constant of $\mathcal{J}$-depth i, it is garanteed that this product is defined and is exactly of $\mathcal{J}$-depth i. Thus, both the first and last elements of w are of $\mathcal{J}$-depth i, as well as $\pi_S(w)$. Thus the product of any prefix or suffix of w will be of $\mathcal{J}$-depth i, and in the same $\mathcal{J}$-class as the first and last elements of w, which corresponds to the definition of $\mathcal{J}$-constant. ◀

For the induction step, we assume to have sweeping evaluation programs $P_{i,r}$ and $P_{i,l}$ for any integer $i < d$. We prove the result for $P_{i+1,r}$ and $P_{i+1,l}$. These two cases being symmetrical, we only show the result for $P_{i+1,l}$. Let $v = \mathrm{LProd}_S \circ P_{i,l}(u)$. If $|P_{i,l}(u)| \neq 1$, we have necessary that the first letter of v is of $\mathcal{J}$-depth strictly greater than i. Otherwise $v = P_{i,l}(u) = \pi_S(u)$. We are going to split v with respect to the $\mathcal{J}$-depth i and apply the program $E = \mathrm{JProd}_S \circ \mathrm{RProd}_S \circ P_{i,r}$ to the prefix. Indeed, after the split, and thanks to the previous Fact, we can apply JProd_S over the factor $\mathrm{RProd}_S \circ P_{i,r}(p)$, where p is the prefix obtained after the split. Indeed, this factor is $\mathcal{J}$-constant. To conclude, $P_{i+1,l} = \mathrm{LSplit}_{S,i+1}\langle E\rangle \circ \mathrm{LProd}_S \circ P_{i,l}$. Thus, each P_i is defined using $O(2^i)$ operations. ◀

B Proofs for Section 4 (Proof of the main results)

We introduce two vectors that we will use extensively in our circuits: given S a semigroup and $\mathbf{c}$ a vectorial encoding of a partial evaluation, we define the universe vector $\mathbf{U} = \bigvee_{s\in S} \mathbf{c}(s)$ We also define the vector marking the end of the vectors: $\mathbf{End} = \neg\mathrm{MSB}(\mathbf{1})$.

Proofs for Theorem 6. In our proofs involving aperiodic semigroups, we will rely on some classical equivalent characterizations of this variety of semigroups.

▶ **Proposition 25** ([25]). *Let S be a semigroup. The following conditions are equivalent:*

- *S is aperiodic*
- *there exists an integer ω such that for all $s \in S$, $\pi_S(s^\omega) = \pi_S(s^{\omega+1})$*
- *All $\mathcal{H}$-classes of S are trivial*

Here is a technical property of aperiodic semigroups that will be useful in the proofs.

▶ **Lemma 26.** *Let S be an aperiodic semigroup. Suppose that $u = s_0 \cdots s_k \in S^+$ is a $\mathcal{J}$-constant word. Then, $\pi_S(u)$ is the unique element of $\mathcal{R}(s_0) \cap \mathcal{L}(s_k)$. If $k > 0$, this also implies that $\pi_S(u) = \pi_S(s_0 s_k)$.*

▶ **Lemma 17.** *For any aperiodic semigroup S, we can compute* $\mathrm{Collapse}_S$ *over any vectorial encoding of a partial evaluation with an* ADD*-vectorial circuit of size $O(|S|^3)$.*

Proof. Consider a word $u \in S^+$ and its $\mathcal{J}$-maximal decomposition $u = w_0 v_1 x_1 \cdots v_t x_t w_t v_{t+1}$. Our goal is to compute a vectorial encoding of the partial evaluation $u' = w_0 \pi_S(v_1 x_1) w_1 \cdots w_{i-1} \pi_S(v_i x_i) w_i \cdots w_t \pi_S(v_{t+1})$. We proceed as follows. We start by computing, for each $s \in D_1(S)$, the vector $\mathbf{SecondEl}(s)$ marking the positions that indicate an s at the beginning of some sub-word v_i. This is done using the operation Successor and the fact that the first letter of each v_i either is the first letter of the word or is such that the product with

the previous letter stays in $D_1(S)$. Similarly, for each $s \in D_1(S)$, we compute the vector $\mathbf{BlockFall}(s)$ marking each letter x_i that follows an occurrence of s. This is done using the operation Successor and the fact that each x_i is after a block of letters which product stays in $D_1(S)$. We then compute the products in the vectors $\mathbf{Prod}(p)$, for any $p \in S_2$. To do so, we define the set $F_p = \{(s,t,x) \in (D_1(S))^2 \times S \mid \pi_S(\alpha \cdot x) = p$, where $\alpha = \mathcal{R}(s) \cap \mathcal{L}(t)\}$. This set is exactly what we need since, thanks to Lemma 26, we know that the product of each subword v_i is only determined by s and t. Thus, this set gives all the triplets (s,t,x) such that, if a subword v_i has s as its first letter, t as its last letter, and if $x_i = x$, then $\pi_S(v_i x_i) = p$. We compute $\mathbf{Prod}(p)$ as a union of calls to Successor over the set F_p. Note that each triplet $(s,t,x) \in (D_1(S))^2 \times S$ can only appear in at most one set F_p, so the size of the union of all these sets is only $O(|D_1(S)|^2 * |S|) \leq O(|S|^3)$. Finally, we compute the output vectors by adding the products computed earlier and removing all the letters used in these products. ◀

▶ **Lemma 18.** *For any aperiodic semigroup S, we can compute* RProd_S *on any vectorial encoding of a partial evaluation of a word over S with an* ADD*-vectorial circuit of size $O(|S|^2)$.*

Proof. Let $u \in S^+$ be a partial evaluation such that u is a $\mathcal{J}$-maximal falling word. To compute the vectorial encoding of $\mathrm{RProd}_S(u)$, we begin by labeling the last element by its value: the vector $\mathbf{LastEl}(t)$ is equal to $\mathbf{End}$ if and only the last letter of u is a t. Then, we use $O(|S|^2)$ calls to Successor to determine the value of the product of the last two elements. We replace the last two elements by that value if the size of u is at least 2, using the vector $\mathrm{Thr2}(\mathbf{U})$ and the IfThenElse circuit we presented earlier in the article. Since that circuit is of constant size, the circuit for RProd_S is of size $O(|S|^2)$. ◀

Before proving Lemma 19, we prove the following technical lemma.

▶ **Lemma 27.** *Let S be an aperiodic semigroup and $u \in S^+$ a $\mathcal{J}$-maximal falling word over S such that its last letter is not an element of $D_1(S)$. Consider a fixed element $s \in D_1(S)$ and write u as its s-decomposition $w_0 s^{k_1} x_1 w_1 \cdots w_{t-1} s^{k_t} x_t w_t$. Then, we can use an* ADD*-vectorial circuit of size $O(|S|)$ which takes the vectorial encoding as input and produces a vectorial encoding of the word $w_0 s^{k_1 - 1} \pi_S(s x_1) \cdots s^{k_t - 1} \pi_S(s x_t) w_t$.*

Proof. With a call to Successor and $O(|S|)$ conjunctions, we can obtain the vectors $\mathbf{Last}(t)$, for $t \in S \setminus \{s\}$, that mark the occurrences of t preceeded by an occurrence of s. Then, we can replace thos elements by the product $\pi_S(st)$. Instead of removing the corresponding occurrences of s, we remove the first s of each block, which is equivalent but far easier to do in our model. Those occurrences of s can be marked using a single call to Successor. ◀

▶ **Lemma 19.** *Let S be an aperiodic semigroup of $\mathcal{J}$-depth d. For any element $s \in D_1(S)$, we can compute* $\mathrm{Falling}_S(s)$ *over any vectorial encoding of a partial evaluation in its domain with an* ADD*-vectorial circuit of size $O(d|S|)$.*

Proof. Let u be a word of S^+ and the set $(\mathbf{c}(t))_{t \in S}$ be a vectorial encoding of u. Since we know that the last element of u is not in $D_1(S)$, we know that each block of occurrences of s is followed by at least one element. By applying Lemma 27, we can reduce by 1 the size of each of these blocks. Moreover, the semigroup S is aperiodic, so by Proposition 25 there necessarily exists an integer ω_s such that $\pi_S(s^{\omega_s+1}) = \pi_S(s^{\omega_s})$. Thus, if we apply Lemma 27 ω_s times, the only occurrences of s that will be left will be any letter s that was originally followed by at least ω_s other occurrences of s. Since $\pi_S(s^{\omega_s+1}) = \pi_S(s^{\omega_s})$, we can just forget those occurrences without changing anything else. Note that, for any $t \in S$, we have $\omega_t \leq d$, where d in the $\mathcal{J}$-depth of S, so we apply Lemma 27 at most d times. ◀

Proofs for Theorem 5 In our proofs involving semigroups in **DA**, we will rely on some classical equivalent characterizations of the variety **DA**.

▶ **Proposition 28** ([29, Theorem 2]). *Let M be a monoid. The following are equivalent:*
- *M is in the variety* **DA**
- *if J is a regular $\mathcal{J}$-class of M, then J is an aperiodic semigroup*
- *$\forall x, y, z \in M, (xyz)^\omega y (xyz)^\omega = (xyz)^\omega$ (this is the algebraic characterization of* **DA***)*

Any semigroup in **DA** is aperiodic, so we will also be able to use Proposition 25. Moreover, the class **DA** admits the following nice property that we will rely on in the proofs.

▶ **Lemma 29** (Algebra folklore). *Let S be a semigroup in* **DA** *and R an $\mathcal{R}$-class of S. Then there exist two sets $T, K \subseteq S$ such that $S = T \uplus K$ and, for all $x \in R$, we have $\forall s \in T, xs \mathcal{R} x$ and $\forall s \in K, xs <_{\mathcal{J}} x$. Moreover, both T and K are sub-semigroups of S such that, if we denote by J is the $\mathcal{J}$-class containing R, $J \subseteq T$ if R is regular and $J \subseteq K$ otherwise. It follows that if S is a monoid, then T is also a monoid.*

Before proving the lemmas necessary for Theorem 5, we define some intermediary operations. These operations will all be of the same form: for any $i \in \mathbb{N}$, we define the operation $\text{Value}_{i,S}$ that identifies the i^{th} semigroup element that occurs in the word represented by the input vectors. Formally, we define this operation as follows:

▶ **Definition 30.** *Let S be a semigroup in* **DA** *and let $(\mathbf{c}(s))_{s \in S}$ be a vectorial encoding of some word $u \in S^+$. For each $s \in S$ and $i \in \mathbb{N}$, we define the vector $\text{Value}_{i,S}(s)$ that is equal to $\mathbf{1}$ if and only if there exists an integer j such that the j^{th} element of $\mathbf{c}(s)$ is a 1 and the position j is the i^{th} position of the vector $\mathbf{U}$ to hold a 1. Otherwise, $\text{Value}_{i,S}(s) = \mathbf{0}$.*

▶ **Lemma 31.** *For any integer $i \geq 1$, we can compute the function $\text{Value}_{i,S}$ over any vectorial encoding of a partial evaluation with a* Sweeping*-vectorial circuit of size $O(i + |S|)$.*

Proof. Given a set of input vectors $I = (\mathbf{c}(s))_{s \in S}$, $\text{Value}_{i,S}(I)$ is a set of vectors $(\mathbf{out}(s))_{s \in S}$ such that, for each element $s \in S$, $\mathbf{out}(s)$ is computed as follows. We begin by removing the first $i-1$ bits set to 1 in the union of the inputs by defining the vectors $\mathbf{U}_0 = \mathbf{U}$ and, $\forall j < i-1$, $\mathbf{U}_{j+1} = \text{LSB}(\mathbf{U}_j)$. Then, for each $x \in S$, we set to 0, in $\mathbf{c}(x)$, the $i-1$ first bits set to 1 in $\mathbf{U}$ by computing the vector $\mathbf{rm}(x) = \mathbf{c}(x) \wedge \mathbf{U}_{i-1}$. Now, to detect the element associated to the i^{th} bit set to 1 in $\mathbf{U}$, we only need to detect the value associated to the first bit set to 1 in $\bigvee_{x \in S} \mathbf{rm}_x$, which is done as follows: for any $s \in S$, we compute the vector $\mathbf{out}(s)$ that is full of ones if and only if the position of the first bit set to 1 in $\mathbf{U}_{i-1}$ (that is the union of the vectors $\mathbf{rm}(x)$) is set to 1 in the vector $\mathbf{rm}(s)$. Thus, $\mathbf{out}(s) = \text{Eq}(\text{pref-}\vee(\mathbf{rm}(s)), \text{pref-}\vee(\mathbf{U}_{i-1}))$. ◀

▶ **Lemma 20.** *For any semigroup $S \in$ **DA**, we can compute JProd_S over any vectorial encoding of a partial evaluation in its domain with a* Sweeping*-vectorial circuit of size $O(|S|^2)$.*

Proof. Let $u = u_0 \cdots u_k$ be a word of S^+. To compute $\text{JProd}_S(u)$, we want to detect the first and last bits set to 1 in $\mathbf{U}$ in order to compute an encoding of the word composed only of the element $\pi_S(u_0 u_k)$. The first element is directly indicated by the vectors $\text{Value}_{1,S}(s)$ for each element $s \in S$. Now we detect the last element by computing the similar vectors $\mathbf{Last}(s)$ for each $s \in S$: $\mathbf{Last}(s) = \text{Eq}(\text{suf-}\vee(\mathbf{c}(s)), \text{suf-}\vee(\mathbf{U}))$. With these vectors, we know the value of the product: the product is $s \in S$ if and only if $\text{Value}_{1,S}(t) \wedge \mathbf{Last}(p)$ is equal to $\mathbf{1}$ for some $(t, p) \in S^2$ such that $\mathcal{R}(t) \cap \mathcal{L}(p) = \{s\}$. We set the last bit of the corresponding output vector to 1, and the rest to 0. ◀

▶ **Lemma 21.** *For any semigroup $S \in \mathbf{DA}$, we can compute* LProd_S *and* RProd_S *over any vectorial encoding of a partial evaluation in their domains with* Sweeping*-vectorial circuits of size $O(|S|^2)$.*

Proof. The two operations are symmetrical, so we present only the circuit for LProd_S. Let u be a word of S^+. We can use $\mathrm{Value}_{1,S}$ and $\mathrm{Value}_{2,S}$ to compute vectors that give the values of the first and second elements. If $\forall s \in S$, $\mathrm{Value}_{2,S}(s) = \mathbf{0}$, then $|u| = 1$ and there is nothing to do. Thus, we use a circuit IfThenElse. If $|u| > 1$, we perform the product by computing a vector **PosSec** with a unique 1 at the position of the second letter, which takes only a constant number of gates, then we remove the first two elements of the input vectors and add **PosSec** to the vector corresponding to the product. This last operation takes $O(|S|^2)$ gates since we need to check all the pairs of elements of S to compute the product. ◀

▶ **Lemma 22.** *Let S be a semigroup in* **DA** *of $\mathcal{J}$-depth d, i be an integer such that $1 \leq i \leq d$, let P be a sweeping evaluation program defined at least on all left $\mathcal{J}$-constant words of depth i, and suppose that we have a* Sweeping*-vectorial circuit of size s_P that computes P over any vectorial encoding of a partial evaluation. Then we can compute* $\mathrm{LSplit}_{S,i}\langle P\rangle$ *and* $\mathrm{RSplit}_{S,i}\langle P\rangle$ *over any vectorial encoding of a partial evaluation in their respective domains with* Sweeping*-vectorial circuits of size $O(|S|^2 + s_P)$.*

Proof. The two operations are symmetrical, so we only present the circuit for $\mathrm{LSplit}_{S,i}\langle P\rangle$. Let $u = u_0 \cdots u_k$ be a word of S^+. We want to detect the first element u_i such that $\pi_S(u_0 \cdots u_i)$ is of $\mathcal{J}$-depth at least $i+1$ in order to replace the prefix of $u_0 \cdots u_{i-1}$ by its image through P. To do that, we begin by checking if the first element of the word is of $\mathcal{J}$-depth i by computing the vectors $\mathrm{Value}_{1,S}(s)$ for all $s \in D_i(S)$. The union of those vectors is then used in a circuit IfThenElse: if the union is $\mathbf{0}$, nothing is done. Otherwise, we want to find the first position such that the product of the prefix is of $\mathcal{J}$-depth at least $i+1$. Thanks to Lemma 29, we know that the set of elements that make that product fall in a $\mathcal{J}$-class of greater $\mathcal{J}$-depth depends only on the $\mathcal{R}$-class of the prefix, which is uniquely determined by the first element, since that element is necessarily of $\mathcal{J}$-depth i. Using the vectors $\mathrm{Value}_{1,S}(s)$ we computed, we can determine the $\mathcal{R}$-class of the prefix. Depending on this $\mathcal{R}$-class, we search for the first letter of the word that belongs to the set K defined by Lemma 29, using calls to pref-$\vee$, and we mark all letters before its position: these letters are exactly the prefix we need to consider. Computing all these masks for each $\mathcal{R}$-class takes $O(|S|)$ gates. Then, we mask the input vectors and use the results as inputs for the circuit C_P. Finally, we reassemble the results with the suffixes that were not considered in C_P to get an encoding of $\mathrm{LSplit}_{S,i}\langle P\rangle(u)$. ◀

Reconstructing Words Using Queries on Subwords or Factors

Gwenaël Richomme
LIRMM, Université Paul-Valéry Montpellier 3, Université de Montpellier, CNRS, Montpellier, France

Matthieu Rosenfeld
LIRMM, Université de Montpellier, CNRS, Montpellier, France

Abstract

We study word reconstruction problems. Improving a previous result by P. Fleischmann, M. Lejeune, F. Manea, D. Nowotka and M. Rigo, we prove that, for any unknown word w of length n over an alphabet of cardinality k, w can be reconstructed from the number of occurrences as subwords (or scattered factors) of $O(k^2\sqrt{n\log_2(n)})$ words. Two previous upper bounds obtained by S. S. Skiena and G. Sundaram are also slightly improved: one when considering information on the existence of subwords instead of on the numbers of their occurrences, and, the other when considering information on the existence of factors.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Combinatorics on words

Keywords and phrases Word reconstruction, Subwords, Factors

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.52

Acknowledgements Authors thank Victor Poupet for useful discussions. Many thanks also for the referees for their accurate reading and their valuable suggestions.

1 Introduction

A natural combinatorial question is to ask how much partial information on an object is needed to reconstruct this object (see below and in our references for examples). For example, in [2, 3], P. Fleischmann, M. Lejeune, F. Manea, D. Nowotka and M. Rigo consider the problem of reconstructing a word w from information on the number of occurrences as subwords of w of some words. Let us recall that a word u is a *subword* of a word w (or a *scattered subword* of w) if u and w can be decomposed in the form $u = u_1 \cdots u_\ell$ and $w = v_0u_1v_1 \cdots u_\ell v_\ell$ for some words $u_1, \ldots, u_\ell, v_0, \ldots, v_\ell$. Such a double decomposition marks an occurrence of u as a subword of w. The number of occurrences of u as a subword of w is sometimes denoted as the binomial coefficient $\binom{w}{u}$ since this number coincides with the traditional coefficient $\binom{|w|}{|u|}$ when the words u and w are written on a single letter (here, as usual in combinatorics on words, $|w|$ denotes the length of w), see for instance [8, chap. 6]. The problem addressed by Fleischmann *et al.* is presented as a game in which the player has to guess an unknown word. In his task the player asks questions in a certain form until he has enough information to uniquely determine the word. More precisely, at each round, the player chooses a word u based on the previous answers that he obtained and asks for the value of $\binom{w}{u}$. The goal of the player is to minimize the number of questions. Fleischmann *et al.* proved that there is a strategy to ensure that at most $\min(|w|_a, |w|_b) + 1 \leq \lfloor \frac{|w|}{2} \rfloor + 1$ questions are needed when w is defined on the binary alphabet $\{a, b\}$ (for a letter α, $|w|_\alpha = \binom{w}{\alpha}$ denotes the number of occurrences of α in w). For any word w over the alphabet $\{1, \ldots, k\}$ they proved that the number of questions needed is bounded by $\sum_{i\in\{1,\ldots,k\}} |w|_i(k + 1 - i)$. Our main results (Theorem 1 and Corollary 6) prove that this number of questions is at

© Gwenaël Richomme and Matthieu Rosenfeld;
licensed under Creative Commons License CC-BY 4.0

most $\binom{k}{2}\left(7\left\lceil\sqrt{|w|\log_2(|w|)}\right\rceil+4\right)$. For any fixed k, our upper bound is asymptotically much stronger as the length of the word goes to infinity. For binary words in particular, their upper bound is $\frac{|w|}{2}+1$ and ours is $7\left\lceil\sqrt{|w|\log_2(|w|)}\right\rceil+4$. We also adapt this strategy (Theorem 2) to provide an algorithm whose expected running time over a uniform random binary word of length n is $\mathcal{O}(\log_2 n)$.

Let us recall that the previous game is related to another problem that seems to have been first introduced by L. O. Kalashnik [5]: What is the smallest ℓ such that we can reconstruct w from the values $\binom{w}{u}$ for all words u of length ℓ? As far as we know, the best upper bound, $\lfloor\frac{16}{7}\sqrt{|w|}\rfloor+5$, for this problem was obtained by I. Krasikov and Y. Roditty in 1997 [6] using a link with the Prouhet-Tarry-Escott problem about Diophantine analysis. Also the best known lower bound, $3^{(\sqrt{2/3}-o(1))\log_3^{1/2}(|w|))}$, is due to [1]. Our result does not improve this upper bound since, in the binary case, at least one query concerns a word u of length at least $\min(|w|_0,|w|_1)$ which is around $|w|/2$ for many words w.

In a variant of the previous problem queries in the form "what is the value of $\binom{w}{u}$?" is replaced with queries in the form "Is $\binom{w}{u}\geq 1$?" or equivalently "Is u a subword of w?". More precisely the problem is to determine the least value ℓ such that the set of subwords of length ℓ determines uniquely a word w. This problem arose in various areas. In [8, Chap 6], it is proved that any word w of length n over an alphabet $\mathcal{A}$ is uniquely determined by its set of subwords in the form a^*b^* of length at most $\lceil|w|_a+|w|_b+1/2\rceil$ with a and b distinct letters of $\mathcal{A}$. The problem is also studied in [7].

In [9, 10], in the context of DNA sequencing of hybridization, S. S. Skiena and G. Sundaram consider the problem of minimizing the number of queries in the form "Is u a subword of w?". They prove that a word w of length n over an alphabet $\mathcal{A}$ of cardinality k can be reconstructed using $O(n\log_2(k)+k\log_2(n))$ such queries. More precisely Theorem 15 in [10] states that $1.59n\log_2(k)+2k\log_2(n)+5k$ queries are sufficient to reconstruct w. Using a basic information theory approach S. S. Skiena and G. Sundaram also provide the lower bound $n\log_2 k$ for the number of queries. In Section 3, we slightly improve S. S. Skiena and G. Sundaram's strategy and we provide a new upper bound, reducing the gap with the lower bound. More precisely, we state that at most $n\log_2(k)+k(2+\lfloor\log_2(n+1)\rfloor)$ queries are sufficient to reconstruct w, reducing the gap between the bounds from $0.59n\log_2(k)+O(k\log_2(n))$ down to $O(k\log_2(n))$.

In Section 4, we consider factors instead of subwords (a word u is a *factor* of a word w if there exist words p and s such that $w=pus$) and the corresponding problem of minimizing the number of queries in the form "Is u a factor of w?" needed to reconstruct an unknown word w. In [9, 10], S. S. Skiena and G. Sundaram prove that, for an unknown word w over an alphabet $\mathcal{A}$ of cardinality k, if the length n of w is known then w can be reconstructed using a number of queries which is in $(k-1)n+2\log_2(n)+O(k)$. Actually their proof leads to the upper bound $(k-1)n+\log_2(n)+O(k)$, which is $n+\log_2(n)+O(1)$ in the binary case. This more accurate upper bound was already mentioned in the binary case in [10]. A simple double counting argument (there are k^n words of length n and each question has two possible outcomes) leads to the lower bound $n\log_2 k$. We improve their strategy and reduce the upper bound to $(k-1)(n+2)+\left\lceil\frac{\log_2(n)}{2}\right\rceil+3$. In the binary case, this reduces the gap between the lower and the upper bound from $\log_2(n)+O(1)$ down to $\left\lceil\frac{\log_2(n)}{2}\right\rceil+5$.

Queries in the form "What is the number of occurrences of a word u as a factor of w" have also been considered by S.S. Skiena et G. Subraman [10]. Their lower bound $nk/4-o(n)$ on the number of queries needed is, up to our knowledge, the best known. One can deduce whether a word u occurs as a factor in a word w from the number of occurrences of u in w.

This observation allows them to obtain the same upper bounds for this fourth problem than for the previous problem. Similarly, our bound applies. Hence, we also slightly improve the upper bound in this case, but this improvement is negligible compared to the size of the gap between the lower bound and the upper bound.

Basic definitions and notations have already been recalled (following [8]). Let us observe that $\#S$ denotes the cardinality of a set S. Moreover, given a word w over an alphabet $\mathcal{A}$, we will simply use n to denote the length $|w|$ of w and k to denote the cardinality $\#\mathcal{A}$ of $\mathcal{A}$.

2 How-many-subwords queries

In this section, we focus on queries in the form "How many occurrences of u as a subword does w contains?" or equivalently "What is the value of $\binom{w}{u}$?". We call such a query a #-subword query. Our main result regarding this kind of query is the following. Of course, as it will be the case for other queries in the next sections, we assume that such a query can be answered without knowing w.

▶ **Theorem 1.** *The number of #-subword queries needed to reconstruct a word of length n over $\{0,1\}$ is at most $7\left\lceil\sqrt{n\log n}\right\rceil + 4$ whether n is known or not.*

A word w that contains m occurrences of 1, can always be written as $w = 0^{s_0}10^{s_1}1\dots10^{s_m}$ where the s_i are nonnegative integers. Since $m = \binom{w}{1}$, it only requires one query to find m. Our goal is to find the values of all the s_i. Our strategy relies on the fact that if we know which of the s_i are "large" and if we know their values then we can determine multiple others s_i with a single query (this is shown in Lemma 4). On the other hand since we cannot have too many "large" s_i we have an efficient strategy to find all these s_i (see Lemma 5). Using these two facts together and optimizing the meaning of "large" we get the desired result.

Actually, in a uniform random word we do not expect to have any s_i larger than $\mathcal{O}(\log n)$ and this leads to a more efficient average case algorithm.

▶ **Theorem 2.** *There is a deterministic strategy that, given any integer n, reconstructs in average in $\mathcal{O}(\log_2(n))$ queries any word w taken uniformly at random among all binary words of length n.*

The next lemma allows to prove Lemma 4.

▶ **Lemma 3.** *Let r, ℓ, $s_1,\dots,s_r$ be non-negative integers such that $1 \le r \le \ell+1$ and for all $j \in \{1,\dots,r\}$, $s_j < \frac{\ell+1}{r}$. The values of $s_1,\dots,s_r$ are uniquely determined by the values of $\binom{0^{s_r}10^{s_{r-1}}1\cdots0^{s_2}10^{s_1}1^\ell}{01^\ell}$, r and ℓ.*

Proof. Let us first express the number of occurrences of 01^ℓ as subword in $0^{s_r}10^{s_{r-1}}1\cdots0^{s_1}1^\ell$. By considering separately the different possible positions of the 0 in the occurrence we obtain

$$\binom{0^{s_r}10^{s_{r-1}}1\cdots0^{s_2}10^{s_1}1^\ell}{01^\ell} = \sum_{j=1}^{r} s_j\binom{\ell+j-1}{\ell} = \sum_{j=1}^{r} s_j\binom{\ell+j-1}{j-1}. \tag{1}$$

Let $\beta = \max_j s_j$. We first show that for all $t \in \{1,\dots,r\}$,

$$\sum_{j=1}^{t} s_j\binom{\ell+j-1}{j-1} \le \beta\binom{\ell+t}{t-1}. \tag{2}$$

We proceed by induction on t. It is easily verified for $t = 1$. Now if (2) holds for t, then

$$\sum_{j=1}^{t+1} s_j \binom{\ell+j-1}{j-1} = \sum_{j=1}^{t} s_j \binom{\ell+j-1}{j-1} + s_{t+1}\binom{\ell+t}{t} \leq \beta\binom{\ell+t}{t-1} + s_{t+1}\binom{\ell+t}{t}$$
$$\leq \beta\left(\binom{\ell+t}{t-1} + \binom{\ell+t}{t}\right) = \beta\binom{\ell+t+1}{t}$$

which concludes the inductive proof of (2).

Moreover, for all $t \in \{1, \ldots, r\}$, $\beta\binom{\ell+t}{t-1} < \frac{\ell+1}{r}\binom{\ell+t}{t-1} \leq \frac{\ell+1}{t}\binom{\ell+t}{t-1} = \binom{\ell+t}{t}$. Together with (2), it implies that for all $t \in \{1, \ldots, r\}$,

$$0 \leq \sum_{j=1}^{t} s_j \binom{\ell+j-1}{j-1} < \binom{\ell+t}{t}. \tag{3}$$

Observe that, for all $t \in \{1, \ldots, r-1\}$,

$$s_{t+1} = \frac{\sum_{j=1}^{t+1} s_j \binom{\ell+j-1}{j-1} - \sum_{j=1}^{t} s_j \binom{\ell+j-1}{j-1}}{\binom{\ell+t}{t}}.$$

But s_{t+1} is an integer and by equation(3) the right part of the fraction in the left-hand-side is in $[0, 1[$ we deduce

$$s_{t+1} = \left\lfloor \frac{\sum_{j=1}^{t+1} s_j \binom{\ell+j-1}{j-1}}{\binom{\ell+t}{t}} \right\rfloor. \tag{4}$$

By Equations (1) and (4), we can deduce the value of s_r from r, l and $\sum_{j=1}^{r} s_j \binom{\ell+j-1}{j-1}$ which is itself deduced from $\binom{0^{s_r}10^{s_{r-1}}1\cdots 0^{s_2}10^{s_1}1^{\ell}}{01^{\ell}}$. From the value of s_r, we can now deduce $\sum_{j=1}^{r-1} s_j \binom{\ell+j-1}{j-1}$ and thus s_{r-1} by (4). Thus, by an "inverse induction" from $r-1$ to 1, we deduce the values of all the s_j. ◀

Lemma 3 allows us to determine the length of multiple consecutive 0-blocks with only one query under some strong hypothesis, but we can relax these hypotheses as follows. The idea is that if we have some large s_i and a prefix, it is enough to know the value of these s_i and of the prefix in order to remove their contribution before applying the previous lemma.

▶ **Lemma 4.** *Let p and v be words, r and $s_1, \ldots, s_r$ be nonnegative integers such that $1 \leq r \leq |v|_1 + 2$ and let $w = p0^{s_r}10^{s_{r-1}}\ldots 10^{s_1}1v$. Suppose that p, $|v|_1$ and r are known and that for all j, either s_j is known or $s_j < \frac{|v|_1+2}{r}$, then the value of $\binom{w}{01^{1+|v|_1}}$ uniquely determines the values of all the unknown s_j for $j \in \{1, \ldots, r\}$.*

Proof. For all $j \in \{1, \ldots, r\}$, let s'_j be such that if $s_j < \frac{|v|_1+2}{r}$, then $s'_j = s_j$ and $s'_j = 0$ otherwise. Then $s_j - s'_j$ is known for all j (it is s_j if s_j is known and 0 otherwise) and for all j, $s'_j < \frac{|v|_1+2}{r}$.

Now, by considering the possible positions of the 0 in the occurrences of $01^{1+|v|_1}$, we get

$$\begin{aligned}
\binom{w}{01^{1+|v|_1}} &= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \binom{0^{s_r}10^{s_{r-1}}\dots10^{s_1}1^{1+|v|_1}}{01^{1+|v|_1}} \\
&= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \sum_{j=1}^{r} s_j \binom{j+|v|_1}{1+|v|_1} \\
&= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \sum_{j=1}^{r} (s_j - s'_j)\binom{j+|v|_1}{1+|v|_1} + \sum_{j=1}^{r} s'_j \binom{j+|v|_1}{1+|v|_1} \\
&= \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} + \sum_{j=1}^{r} (s_j - s'_j)\binom{j+|v|_1}{1+|v|_1} + \binom{0^{s'_r}10^{s'_{r-1}}\dots10^{s'_1}1^{1+|v|_1}}{01^{1+|v|_1}}.
\end{aligned}$$

It implies that,

$$\binom{0^{s'_r}10^{s'_{r-1}}\dots10^{s'_1}1^{1+|v|_1}}{01^{1+|v|_1}} = \binom{w}{01^{1+|v|_1}} - \binom{p1^{r+|v|_1}}{01^{1+|v|_1}} - \sum_{j=1}^{r}(s_j - s'_j)\binom{j+|v|_1}{1+|v|_1}.$$

By assumption, $\binom{w}{01^{1+|v|_1}}$, p, r, $|v|_1$ and for all j, $(s_j - s'_j)$ are known. Hence, the quantity $\binom{0^{s'_r}10^{s'_{r-1}}\dots10^{s'_1}1^{1+|v|_1}}{01^{1+|v|_1}}$ is uniquely determined. For all j, $s'_j < \frac{|v|_1+2}{r}$ and we deduce from Lemma 3 that the values of all the s'_j are uniquely determined which concludes our proof. ◀

For any word w over $\{0,1\}$ decomposed as $w = 0^{s_0}10^{s_1}1\cdots0^{s_{t-1}}10^{s_t}$, we call i the *index* of the 0-block 0^{s_i}. If we want to use the previous lemma to reconstruct a word, we first need to determine the indices of all the 0-blocks that are longer than some predetermined length.

▶ **Lemma 5.** *Let $w \in \{0,1\}^*$ and m be an integer. Let I be the set of indices of 0-blocks of w of length at least m. Suppose that we know $|w|$ and $|w|_0$ (and so also $|w|_1 = |w| - |w|_0$), then the number of #-subword queries needed to determine I is at most*

$$\frac{2|w|_0\lceil\log_2(|w|_1+1)\rceil}{m}.$$

Proof. We use Algorithm 1 to determine I calling it with $\ell = 0$ and $u = |w|_1$. Note that $|w|_1 = |w| - |w|_0$ is known.

Algorithm 1 An algorithm that prints the indices $i \in \{\ell,\dots,u\}$ of the 0-blocks of length at least m that occur in w.

```
procedure RecBlocks(w, m, ℓ, u)
    if (w over 1^ℓ 0^m 1^{|w|_1 − u}) ≥ 1 then
        if u = ℓ then
            Print ℓ
        else
            RecBlocks(w, m, ℓ, ⌊(ℓ+u)/2⌋)
            RecBlocks(w, m, ⌊(ℓ+u)/2⌋ + 1, u)
```

The condition of the main "if" verifies that the lengths of the 0-blocks whose indices are in $\{\ell,\dots,u\}$ sum to at least m. If it doesn't then we know that none of these blocks can have length at least m so we do not need to call the function recursively on any of them. From this, verifying the correctness of the algorithm is rather straightforward.

Let us now bound the total number of queries. For this, we consider the *tree of recursive calls to Recblocks* defined as follows: the root of the tree is the initial call with $\ell = 0$ and $u = |w|_1$; a call a is the child of another call b if the call a was made in b. The *depth* of a call is its distance to the root. The *weight* of a call is the quantity $u + 1 - \ell$. For any call of weight x, the weights of its children are $\lceil x/2 \rceil$ or $\lfloor x/2 \rfloor$ (and the sum of the weights of the two children is x). Let f be the function such that $f : x \to \lceil \frac{x}{2} \rceil$. The root has weight $|w|_1 + 1$ and f is a non-decreasing function, so any call of depth d has weight at most $f^d(|w|_1 + 1)$. For any integer x, $f(x) \leq \frac{x+1}{2}$, and, in particular, for all $d \geq 1$, $f^d(|w|_1 + 1) \leq \frac{f^{d-1}(|w|_1+1)+1}{2}$. By induction on d, $f^d(|w|_1 + 1) < \frac{|w|_1+1}{2^d} + 1$. Any call of depth $\lceil \log_2(|w|_1 + 1) \rceil$ has weight at most 1 (the weight is an integer smaller than 2) and is a leaf of the tree. Hence, the depth of any call is at most $\lceil \log_2(|w|_1 + 1) \rceil$.

Moreover, one easily verifies by induction on the depth that for any two different calls c and c' at the same depth the corresponding intervals $[\ell, u]$ and $[\ell', u']$ are disjoint. We say that a call with the values ℓ and u *owns* the occurrences of 0 that belongs to all the blocks of indices between ℓ and u. Then by the previous remark, the set of occurrences of 0 owned by two calls at the same depth are disjoint. Since the condition of the first "if" is true if the call owns at least m occurrences of 0, we deduce that there are at most $\frac{|w|_0}{m}$ such calls on any given depth. Since each such call has two children, we deduce that the number of calls at any depth is at most $2\frac{|w|_0}{m}$. Hence the total number of calls, is at most $\frac{2|w|_0\lceil \log_2(|w|_1+1) \rceil}{m}$. Since we ask one query by call this concludes the proof. ◀

We are now ready to show our main result. We will first use the algorithm from Lemma 5 to find all the blocks that are of length $\lceil \sqrt{n \log n} \rceil$ and then we use Lemma 4 to determine all the other blocks.

Proof of Theorem 1.

Phase 1. Let w be the unknown word. It costs two queries to get $|w|_0 = \binom{w}{0}$ and $|w|_1 = \binom{w}{1}$. Then $n = |w| = |w|_0 + |w|_1$ is known. Suppose without loss of generality that $\binom{w}{0} \geq n/2 \geq \binom{w}{1}$ (otherwise simply exchange the role of 0 and 1 in the following).

Phase 2. Let $m = \lceil \sqrt{n \log n} \rceil$. We use the algorithm from Lemma 5 to locate all the 0-blocks of length at least m. There are at most $\frac{n}{m}$ such blocks and we can use one query for each of them to determine their respective length: Indeed if the block is at index i with $i \in \{0, \ldots, |w|_1\}$, its length is $\binom{w}{1^i 0 1^{|w|_1 - i}}$. Thus locating 0-blocks of length at least m together with their lengths require at most $\frac{2|w|_0\lceil \log(|w|_1+1) \rceil}{m} + \frac{n}{m}$ queries. This number of queries is less than $3\frac{n \log n}{m} \leq 3\sqrt{n \log n}$.

Phase 3. We now need to determine the lengths of 0-blocks of length at most m. We first determine the 0-blocks occurring before the $\left\lceil \frac{|w|_1}{2} \right\rceil$ last occurrences of 1. Secondly, we determine the 0-blocks occurring after the $\left\lceil \frac{|w|_1}{2} \right\rceil$ first occurrences of 1. After this, the lengths of all the 0-blocks are known and we know w. We describe only how to determine the first half of the blocks, since reconstructing the second half of the blocks can be done symmetrically.

There are $\left\lceil \frac{|w|_1}{2} \right\rceil + 1$ 0-blocks before the $\left\lceil \frac{|w|_1}{2} \right\rceil$ last occurrences of 1. We determine the unknown blocks among them in at most m steps from left to right considering, at each step, at most $r = \left\lfloor \frac{|w|_1}{2m} \right\rfloor$ blocks. Since $mr \geq \frac{|w|_1}{2} - m$, we might miss up to $m + 1$ blocks after this, that we can recover one by one for up to $m + 1$ extra queries. At one step $w = p0^{s_r}10^{s_{r-1}} \cdots 10^{s_1}1v$ with p an already known prefix of w (initially p is the empty word) and $|v|_1 \geq \left\lceil \frac{|w|_1}{2} \right\rceil$. For each $i \in \{1, \ldots, r\}$, if s_i is unknown then $s_i < m = \frac{|w|_1/2}{|w|_1/(2m)} < \frac{|v|_1+2}{r}$.

By Lemma 4, only one query is needed to know the r blocks. Hence, we determine the 0-blocks occurring before the $\left\lceil \frac{|w|_1}{2} \right\rceil$ last occurrences of 1 in at most $2m+1 = 1+2\left\lceil \sqrt{n \log n} \right\rceil$ queries (and similarly to know the 0-blocks occurring after the $\left\lceil \frac{|w|_1}{2} \right\rceil$ last occurrences of 1).

In total, our strategy uses $2 + 3\left\lceil \sqrt{n \log n} \right\rceil + 2(1 + 2\left\lceil \sqrt{n \log n} \right\rceil) = 7\left\lceil \sqrt{n \log n} \right\rceil + 4$. ◀

For any alphabets $\mathcal{A}$ and $\mathcal{B} \subseteq \mathcal{A}$ and any word u over $\mathcal{A}$, the *projection* of u onto $\mathcal{B}$ is the word obtained by removing from u any letter that does not belong to $\mathcal{B}$. We denote it $\pi_{\mathcal{B}}(u)$. For instance, $\pi_{\{0,1\}}(0120201) = 01001$. Over an alphabet of cardinality k if we know the projections over all the binary sub-alphabets, we can uniquely determine the whole word [8, Lemma 6.2.19]. So Theorem 1 has the following corollary.

▶ **Corollary 6.** *The number of #-subword queries needed to reconstruct a word of length n over an alphabet of cardinality k is at most $\binom{k}{2}(7\left\lceil \sqrt{n \log n} \right\rceil + 4)$.*

In Theorem 1 and Corollary 6, we did not try to optimize the multiplicative constant, because we believe that the $\sqrt{n \log n}$ bound is not "sharp up to a multiplicative constant". As suggested by Theorem 2, the number of required queries in Theorem 1 and Corollary 6 might be in $O(\log n)$.

As we will see in Lemma 7, the probability that there is a 0-block of length more than $\lceil 2\log_2(n) \rceil$ is small.

▶ **Lemma 7.** *Let w be a word taken uniformly at random among all binary words of length n. The probability that w contains the factor $0^{\lceil 2\log_2(n) \rceil}$ is at most $1/n$.*

Proof. Let $m = \lceil 2\log_2(n) \rceil$. Let $w_1, \ldots, w_n \in \{0,1\}$ be such that $w = w_1 \cdots w_n$. For all $i \in \{1, \ldots, n-m+1\}$, let E_i be the event that $w_i w_{i+1} \ldots w_{i+m-1} = 0^m$. Then for all i, $\mathbb{P}(E_i) = 2^{-m} \leq 1/n^2$. By union bound,

$$\mathbb{P}(0^m \text{ is a factor of } w) = \mathbb{P}(\cup_{i=1}^{n-m+1} E_i) \leq \sum_{i=1}^{n-m+1} \mathbb{P}(E_i) \leq \frac{1}{n}$$

as desired. ◀

Proof of Theorem 2. First, we determine the number of 0 and 1 in w in 2 queries. Let $m = \lceil 2\log_2(n) \rceil$. We first assume that there is no factor 0^m in w. We can now apply Lemma 4 as in Phase 3 of the proof of Theorem 1, but with $m = \lceil 2\log_2(n) \rceil$. We now have a candidate word w' and we can ask one more question, $\binom{w}{w'}$, to verify if $w = w'$ (this might not be the case, if our starting assumption was false). All of this take $\mathcal{O}(\log_2(n))$ queries.

If we did not obtain the correct word, we know that our assumption was false and we use Theorem 1 to find w in $\mathcal{O}(\sqrt{n \log_2(n)})$ extra queries. By Lemma 7, this happens with probability at most $1/n$, so the expected number of queries of this procedure is at most $\mathcal{O}(\log_2(n)) + \mathcal{O}(\sqrt{n \log_2(n)}/n) = \mathcal{O}(\log_2(n))$. ◀

3 Exists-subword queries

In this section, we focus on queries in the form "Is u a subword of w?" or equivalently "Is $\binom{w}{u} \geq 1$?". We call such a query an $\exists$-subword query. The reconstruction problem using $\exists$-subword queries of a word w of unknown length n over an alphabet $\mathcal{A}$ of cardinality k was solved by S. S. Skiena and G. Sundaram [9, 10] using $1.59n\log_2(k) + 2k\log_2(n) + 5k$ queries. We improve the main coefficient of the bound, replacing 1.59 by 1 which is optimal (any such algorithm requires at least $n\log_2(k)$ queries in the worst case [9, 10]).

▶ **Theorem 8.** *The number of $\exists$-subword queries needed to reconstruct an unknown word w of unknown length n over an alphabet $\mathcal{A}$ of cardinality k is at most*

$$n\lceil \log_2(k)\rceil + k\left(2 + \lfloor \log_2(n+1)\rfloor\right) .$$

Actually, our approach is similar to the method used in [9, 10]. We act essentially by dichotomy on the alphabet but when reconstructing words from their projections on a smaller alphabet we improve the bound on the number of queries. Also on small alphabets we use a linear decomposition instead of a binary decomposition in order to reduce the number of queries needed to deduce the number of occurrences of some letters.

To prove Theorem 8 we use the next two lemmas. The first one considers the reconstruction problem in the one letter alphabet case. The second one describes upper bounds on the number of queries needed to reconstruct a word from projections on disjoint alphabets.

▶ **Lemma 9.** *Given an unknown nonempty word w of length n over an alphabet $\mathcal{A}$ and a letter $\alpha \in \mathcal{A}$, the value $|w|_\alpha$ can be determined using*

- *at most $2\lfloor 1 + \log_2(|w|_\alpha + 1)\rfloor$ $\exists$-subword queries if n is unknown and*
- *at most $\lceil \log_2(n+1)\rceil$ $\exists$-subword queries if n is known.*

The proof of this Lemma is a simple binary search. The details can be found in Appendix A. In the next Lemma we explain how to reconstruct a word w from its projections on two disjoint complementary alphabets. Note that [10, Lemma 14], is almost the same result with a number of queries $2.18|\pi_{\mathcal{B}}(w)| + |\pi_{\mathcal{C}}(w)| + 5$ instead of $|\pi_{\mathcal{B}}(w)| + |\pi_{\mathcal{C}}(w)| + 1$. The main difference is that instead of using a binary search we simply go greedily from left to right when combining the two words. This lemma almost exclusively explains the improvement we obtain over [10, Theorem 2].

▶ **Lemma 10.** *Let w be an unknown word of length n over an alphabet $\mathcal{A}$. Let $\mathcal{B}$ and $\mathcal{C}$ be two disjoint alphabets such that $\mathcal{A} = \mathcal{B} \cup \mathcal{C}$, then*

1. *if we know both projections $\pi_{\mathcal{B}}(w)$ and $\pi_{\mathcal{C}}(w)$, then the word w can be reconstructed using at most $n-1$ $\exists$-subword queries,*
2. *if we know the word $\pi_{\mathcal{B}}(w)$ and $\#\mathcal{C} = 1$, then the word w can be reconstructed using at most $n+1$ $\exists$-subword queries.*

It may be observed that in item 1 of Lemma 10, the length of w can be determined without asking any query since it is equal to $|\pi_{\mathcal{B}}(w)| + |\pi_{\mathcal{C}}(w)|$. This is not the case in item 2. In both cases, the length is not directly used in the proof.

For any word $x = x_1 \cdots x_\ell \in \{0,1\}^\ell$ and integers $i, j \in \{1, \ldots, \ell\}$, let $x[i \ldots j] = x_i x_{i+1} \cdots x_j$ when $i \leq j$. By extension, if $i > j$ (and possibly $i = |x| + 1$ or $j = 0$), then $x[i \ldots j]$ is the empty word.

Proof of Lemma 10. Assume first that $u = \pi_{\mathcal{B}}(w)$ and $v = \pi_{\mathcal{C}}(w)$ are known. The first letter of w is either u_1 or v_1. More precisely, $u_1 v$ is a subword of w if and only if u_1 is the first letter of w, otherwise v_1 is the first letter of w. Thus in one question we can determine the first letter of w, and the projections $\pi_{\mathcal{B}}(w[2 \ldots n])$ and $\pi_{\mathcal{C}}(w[2 \ldots n])$. We can repeat this process and after each new query we obtain the next letter of w and the two projections of the rest of w over $\mathcal{B}$ and $\mathcal{C}$.

Hence Algorithm 2 allows to reconstruct w from u and v. In this algorithm i and j store respectively the successive length of $\pi_{\mathcal{B}}(w[1 \ldots i+j])$ and $\pi_{\mathcal{C}}(w[1 \ldots i+j])$: at the beginning of each while loop, we know $p = w[1 \ldots i+j]$.

From the preliminary comments, it is straightforward that at the end of the algorithm $p = w$ and that the number of $\exists$-subword queries asked is at most $n-1$.

Algorithm 2 An algorithm that returns an unknown word w over $\mathcal{B} \cup \mathcal{C}$ with $\mathcal{B} \cap \mathcal{C} = \emptyset$ from $u = \pi_{\mathcal{B}}(w)$ and $v = \pi_{\mathcal{C}}(w)$.

```
p ← ε ; i ← 0 ; j ← 0
while i < |u| and j < |v| do
    if pu_{i+1}v[j+1..|v|] is a subword of w then
        p ← pu_{i+1} ; i ← i+1
    else
        p ← pv_{j+1} ; j ← j+1
p ← pu[i+1..|u|]v[j+1..|v|]
return p
```

From now on assume that we only know the word $\pi_{\mathcal{B}}(w)$ and the fact that $\mathcal{C} = \{a\}$ for some letter a. We use a strategy similar to the previous case, that is, we try to insert occurrences of the letter a between the letters of $\pi_{\mathcal{B}}(w)$ in a greedy way. Once the places of all letters of $\pi_{\mathcal{B}}(w)$ are known, one has to determine the remaining occurrences of a at the end of w. This leads to the variant Algorithm 3 for which the number of $\exists$-subword queries asked is exactly $n+1$: there is one query by letter of $\pi_{\mathcal{B}}(w)$ and $\pi_{\mathcal{C}}(w)$ and one additional query needed to determine when there is no more letter in $\pi_{\mathcal{C}}(w)$. ◀

Algorithm 3 An algorithm that returns an unknown word w over $\mathcal{B} \cup \{a\}$ with $a \notin \mathcal{B}$ from $u = \pi_{\mathcal{B}}(w)$.

```
p ← ε ; i ← 0
while i < |u| do
    if pau[i+1..|u|] is a subword of w then
        p ← pa
    else
        p ← pu_{i+1} ; i ← i+1
while pa is a subword of w do
    p ← pa
return p
```

The proof of the next result explains the strategy to solve the reconstruction problem using $\exists$-subword queries. The length of w may be unknown.

▶ **Proposition 11.** *Let w be an unknown word over an alphabet of cardinality k. For any $\mathcal{B} \subseteq \mathcal{A}$ with $\#\mathcal{B} \geq 2$, the number of $\exists$-subword queries needed to reconstruct the word $\pi_{\mathcal{B}}(w)$ is at most*

$$\lceil \log_2(\#\mathcal{B}) \rceil |\pi_{\mathcal{B}}(w)| + \#\mathcal{B} \left(2 + \max_{\alpha \in \mathcal{B}} \lfloor \log_2(|w|_\alpha + 1) \rfloor \right) .$$

Proof. We proceed by induction on the cardinality of $\mathcal{B}$ with the two base cases being $\#\mathcal{B} = 2$ and $\#\mathcal{B} = 3$.

If $\mathcal{B} = \{x, y\} \subseteq \mathcal{A}$ with $x \neq y$, we can apply Lemma 9 to determine $\pi_{\{x\}}(w) = x^{|w|_x}$ in at most $2\lfloor 1 + \log_2(|w|_x + 1) \rfloor$ queries. Case 2 of Lemma 10 implies that we can then determine $\pi_{\{x,y\}}(w)$ in at most $|\pi_{\{x,y\}}(w)| + 1$ extra queries. The total number of queries is at most

$$|\pi_{\{x,y\}}(w)| + 1 + 2\lfloor 1 + \log_2(|w|_x + 1) \rfloor \leq \lceil \log_2(\#\mathcal{B}) \rceil |\pi_{\mathcal{B}}(w)| + \#\mathcal{B} \left(2 + \max_{\alpha \in \mathcal{B}} \lfloor \log_2(|w|_\alpha + 1) \rfloor \right)$$

as desired.

If $\mathcal{B} = \{x, y, z\}$ for some distinct letters $x, y, z \in \mathcal{A}$, we use the strategy of the previous paragraph to determine $\pi_{\{x,y\}}(w)$ and we use case 2 of Lemma 10 once again to obtain $\pi_{\{x,y,z\}}(w)$ in at most $|\pi_{\{x,y,z\}}(w)| + 1$ extra queries. The total number of queries is then at most

$$|\pi_{\{x,y\}}(w)| + |\pi_{\mathcal{B}}(w)| + 2 + 2\lfloor 1 + \log_2(|w|_x + 1)\rfloor \leq \lceil \log_2(\#\mathcal{B})\rceil |\pi_{\mathcal{B}}(w)| + \#\mathcal{B}\left(2 + \max_{\alpha\in\mathcal{B}} \lfloor \log_2(|w|_\alpha + 1)\rfloor\right)$$

as desired.

We now have to deal with the induction. Assume $\#\mathcal{B} \geq 4$. Let $\mathcal{C}, \mathcal{C}' \subseteq \mathcal{B}$ be two disjoint alphabets such that $\mathcal{B} = \mathcal{C} \cup \mathcal{C}'$, $\#\mathcal{C} = \lfloor \frac{\#\mathcal{B}}{2} \rfloor$ and $\#\mathcal{C}' = \lceil \frac{\#\mathcal{B}}{2} \rceil$. The two last conditions imply

$$\lceil \log_2 \#\mathcal{C} \rceil \leq \lceil \log_2 \#\mathcal{C}' \rceil = \lceil \log_2 \#\mathcal{B} \rceil - 1\,.$$

By induction hypothesis, the number of queries to determine $\pi_{\mathcal{C}}(w)$ and $\pi_{\mathcal{C}'}(w)$ is at most

$$\begin{aligned}
&\lceil \log_2(\#\mathcal{C})\rceil |\pi_{\mathcal{C}}(w)| + \#\mathcal{C}\left(2 + \max_{\alpha\in\mathcal{C}} \lfloor \log_2(|w|_\alpha + 1)\rfloor\right) \\
+&\lceil \log_2(\#\mathcal{C}')\rceil |\pi_{\mathcal{C}'}(w)| + \#\mathcal{C}'\left(2 + \max_{\alpha\in\mathcal{C}'} \lfloor \log_2(|w|_\alpha + 1)\rfloor\right) \\
&\leq (\lceil \log_2(\#\mathcal{B})\rceil - 1)(|\pi_{\mathcal{C}}(w)| + |\pi_{\mathcal{C}'}(w)|) + (\#\mathcal{C}' + \#\mathcal{C})\left(2 + \max_{\alpha\in\mathcal{C}'\cup\mathcal{C}} \lfloor \log_2(|w|_\alpha + 1)\rfloor\right) \\
&\leq (\lceil \log_2(\#\mathcal{B})\rceil - 1)(|\pi_{\mathcal{B}}(w)|) + \#\mathcal{B}\left(2 + \max_{\alpha\in\mathcal{B}} \lfloor \log_2(|w|_\alpha + 1)\rfloor\right)\,.
\end{aligned}$$

By case 1 of Lemma 10, we only need $|\pi_{\mathcal{B}}(w)|$ extra queries to determine $\pi_{\mathcal{B}}(w)$. In total, we used at most $\lceil \log_2(\#\mathcal{B})\rceil(|\pi_{\mathcal{B}}(w)|) + \#\mathcal{B}\left(2 + \max_{\alpha\in\mathcal{B}} \lfloor \log_2(|w|_\alpha + 1)\rfloor\right)$ queries as required. ◀

Proof of Theorem 8. Theorem 8 is an immediate consequence of Proposition 11 taking $\mathcal{B} = \mathcal{A}$ and using $\max_{\alpha\in\mathcal{B}} \lfloor \log_2(|w|_\alpha + 1)\rfloor \leq \lfloor \log_2(|w| + 1)\rfloor$ ◀

4 Exists-factor queries

In this section, we focus on queries in the form "Is u a factor of w?". Our aim is to prove Theorem 16. As for the result from [10] that we improve here, we assume in this section that the length of the word to determine is known.

A factor u is said *right-extendable* in a word w if there exists a letter a such that ua is also a factor of w. The word ua is a *right extension* of u. A non-right-extendable factor u of w is a suffix of w but the converse does not hold. For instance the word $u = a$ is a suffix of the word $w = aa$ but it is right-extendable. Actually it can be straightforwardly checked that a factor u is not right-extendable in w if and only if u is a suffix of w which has only one occurrence as a factor of w. The notions of left-extendability and left extensions are defined similarly.

The global strategy to reconstruct an unknown word w using queries on factors is to apply the following three steps. First we find a long block of a fixed letter α (proof of Lemma 15). Second we determine a non-right-extendable factor of w having this long block of α as a prefix. Two different approaches are developed in the proof of Lemmas 13 and 14. Depending on the length of the previously found long block of α, one or the other of the two approaches reveals to be more efficient. Finally we determine w from the previous non-right-extendable factor (Lemma 12). Let us first explain this last step.

▶ **Lemma 12.** *Let w be an unknown word of known length n over an alphabet of cardinality k. If we know a non-right-extendable factor s of w then we can reconstruct w with at most $(k-1)(n-|s|)$ $\exists$-factor queries.*

Proof. Assume that $|s| < n$. Then s is a proper suffix of w. Fix a letter α. We can ask "is βs a factor of w?" for each letter β different from α. If the answer is positive for some β then we know that βs is a non-right-extendable factor of w and if the answer is negative for all β then we know that αs is a non-right-extendable factor of w. We then repeat the same process until we reach a word of length n (this word necessarily is w). It costs us at most $k-1$ queries by letter that we have to determine, that is, $(k-1)(n-|s|)$ queries. ◀

We now explain how to efficiently find a non-right-extendable factor of w. For this a letter α is fixed and we assume that we know the greatest t such that α^t occurs as a factor in w. And we will present two different strategies that we will use for different values of t in the proof of Theorem 16. The first strategy will be used when t is not too large. It is described in the proof of the following result.

▶ **Lemma 13.** *Let w be an unknown word of known length n over an alphabet $\mathcal{A}$ of cardinality k. Let $\alpha \in \mathcal{A}$. If we know the largest integer t such that α^t is a factor of w, then a non-right-extendable factor s of w can be determined with at most $(k-1)(|s|+2)$ $\exists$-factor queries.*

Proof. Let σ be a variable that aims to contain the searched non-right-extendable factor of w. We initialize σ with the word α^t. We search for successive right extensions of σ asking the query "is $\sigma\beta$ a factor of w?" for each letter $\beta \neq \alpha$. If the answer is "yes" for some $\beta \neq \alpha$ then we know that $\sigma\beta$ is a factor of w and we set $\sigma\beta$ to be the new value of σ.

If the answer is "no" for all $\beta \neq \alpha$, then either $\sigma\alpha$ is a factor of w or σ is non-right-extendable. If σ does not end with the suffix α^t, we set $\sigma\alpha$ to be the new value of σ. It is possible that σ is no longer a factor of w (and so σ is not a non-right-extendable factor of w), in particular, when the previous value of σ already was the searched non-right-extendable factor of w. But if later, while trying to add a letter $\beta \neq \alpha$, we get "yes" as an answer we deduce that we were right for every previous assumption. If we obtain the answer "no" $t+1$ consecutive times then we have added $t+1$ occurrences of α at the end of σ. This implies that we were wrong since by definition of t, α^{t+1} is not a factor of w. At this point $\sigma = v\alpha^{t+1}$ for some word v that ends with a letter different from α and there exists $r \leq t$ such that $v\alpha^r$ is a suffix of w and both $v\alpha^{r+1}$ and all words $v\alpha^r\beta$ with $\beta \neq \alpha$ are not factors of w: $v\alpha^r$ is the searched non-right-extendable factor of w. We can determine r by asking "is $v\alpha^{r+1}$ a factor of w?" from $r = 0$ and until a negative answer.

Let us now provide an upper-bound for the number of queries. Let $v\alpha^{t+1}$ be the value of σ obtained after $t+1$ consecutive negative queries and let $r+1$ be the number of additional queries asked to determine the final value s of σ. Observe that v was determined using $(k-1)(|v|-t)$ queries. Then we use $(k-1)(t+1)$ queries to get $v\alpha^{t+1}$ and finally we use $r+1$ queries to determine the final value. The total amount of queries is thus bounded by $(k-1)((|v|-t)+(t+1)+(r+1))$. Since $|s| = |v|+r$, this number of queries is bounded by $(k-1)(|s|+2)$. ◀

Let us illustrate in an example the strategy used in the proof of Lemma 13. Assume that the word to reconstruct is $w = 00011100111011$ and that we use $\alpha = 1$. We have $t = 3$ and initially $\sigma = 111$. The answer to the two first queries are positive and we get $\sigma = 11100$. Then the answers to the next three queries are negative and we assume that $\sigma = 11100111$ is a prefix of the expected result. This is confirmed by the next query that sets

$v = 111001110$. The next four negative queries on $v0$, $v10$, $v110$ and $v1110$ imply that the non-right-extendable factor is v, $v1$, $v11$, or $v111$. After three additional queries, we know that 11100111011 is a non-right-extendable factor (hence a suffix) of w.

If t is large (essentially if $t \geq \lceil 4\sqrt{n} \rceil$; see the proof of Theorem 16), then a better strategy is to verify slightly more often that our assumptions are correct when building the non-right-extendable factor. Doing so leads to the alternative strategy provided in the proof of the next result.

▶ **Lemma 14.** *Let w be an unknown word of known length n over an alphabet $\mathcal{A}$ of cardinality k. Let $\alpha \in \mathcal{A}$ be a letter with at least one occurrence in w. Assume that we know n and the largest positive integer t such that α^t is a factor of w. A non-right-extendable factor s of w can be determined using at most $(k-1)(|s|-t) + k\lceil\sqrt{n}\rceil + 1$ $\exists$-factor queries.*

Proof. The strategy is almost identical to the previous one. We initialize σ with the word α^t and we try to extend it by asking whether $\sigma\beta$ for some $\beta \neq \alpha$ is a factor of w and we proceed as previously.

If we obtain the answer "no" r consecutive times then we added r occurrences of α at the end of s. In this case, every $\lceil\sqrt{n}\rceil$ new consecutive occurrences of α, we verify if our current value of σ is a factor of w. If this holds we keep going. Otherwise letting v be the word such that $\sigma = v\alpha^{\lceil\sqrt{n}\rceil}$, $v\alpha^{\lceil\sqrt{n}\rceil}$ is not a factor of w. We need to find the largest r such that $v\alpha^r$ is a factor of w. This can be done by setting $\sigma = v$ and asking the query "is $\sigma\alpha^i$ a factor of w ?", where i starts at 1 and increases until we receive the answer "no".

Let us now count the number of queries. In the first phase, until reaching $v\alpha^{\lceil\sqrt{n}\rceil}$, the length of σ increases from t to $|v\alpha^{\lceil\sqrt{n}\rceil}|$. Each new letter requires at most $k-1$ queries, but each $\lceil\sqrt{n}\rceil$ query a verification query is done. So the number of queries in this first phase is at most (remember $t \geq 1$)

$$(k-1)(|v\alpha^{\lceil\sqrt{n}\rceil}| - t) + \left\lfloor \frac{|v\alpha^{\lceil\sqrt{n}\rceil}| - t}{\lceil\sqrt{n}\rceil} \right\rfloor \leq (k-1)(|v\alpha^{\lceil\sqrt{n}\rceil}| - t) + 1 + \left\lfloor \frac{|w|-1}{\lceil\sqrt{n}\rceil} \right\rfloor$$

which is upper-bounded by $(k-1)(|v|-t) + k\lceil\sqrt{n}\rceil$.

In the second phase there is one verification query and every other query increases the value of i from 1 to $r+1$. So there are at most $1 + r = 1 + |s| - |v| \leq 1 + (k-1)(|s|-|v|)$ other queries in this second phase. Summing the queries of the first and second phase, we deduce that at most $(k-1)(|s|-t) + k\lceil\sqrt{n}\rceil + 1$ queries are used. ◀

Before using Lemma 13 or Lemma 14 we need to determine the greatest power of a letter in a word w. This can be done using a binary search with queries in the form "Is a^t a factor of w?" for $1 \leq t \leq n$. A negative answer to the query "Is a^1 a factor of w?" shows that the letter a does not occur in w. The next result holds for arbitrary alphabets. Its proof specifies how the binary search is done.

▶ **Lemma 15.** *Let w be an unknown word. Let a be a letter, x, y be two known integers and t be the largest integer such that a^t is a factor of w. If we know that $x \leq t \leq y$ then at most $\lceil\log_2(y+1-x)\rceil$ $\exists$-factor queries are needed to determine the value of t.*

Once again the idea of this Lemma is to use a binary search and the details of the proof can be found in Appendix B.

Applying successively Lemma 15, then Lemma 13 or Lemma 14 and finally Lemma 12, we get the next result.

▶ **Theorem 16.** *An unknown nonempty word w of known length n over an alphabet of cardinality $k \geq 2$ can be reconstructed in at most $(k-1)(n+2) + \lceil \frac{\log_2 n}{2} \rceil + 3$ $\exists$-factor queries.*

Proof. We start with the query "is $\alpha^{\lceil 4\sqrt{n} \rceil}$ a factor of w?".

If we obtain a positive answer, we use Lemma 15 (with $x = \lceil 4\sqrt{n} \rceil$ and $y = n$ ($n \geq 1$)) to compute the largest t such that α^t is a factor of w in at most $\lceil \log_2 n \rceil$ queries. Then we apply Lemma 14 to find a non-right-extendable factor s in at most $(k-1)(|s|-t) + k\lceil \sqrt{n} \rceil + 1$ queries. Since $t \geq \lceil 4\sqrt{n} \rceil \geq 4\lceil \sqrt{n} \rceil - 3$,

$$(k-1)(|s|-t) + k\lceil \sqrt{n} \rceil + 1 \leq (k-1)(|s|+3) - (3k-4)\lceil \sqrt{n} \rceil + 1\,.$$

We finally apply Lemma 12 to find w in $(k-1)(n-|s|)$ queries. In this case, including the initial query, we need a total of at most $(k-1)(n+3) + \lceil \log_2 n \rceil - (3k-4)\lceil \sqrt{n} \rceil + 2 \leq (k-1)(n+2)$ queries (we use $k \geq 2$ and $n \geq 1$ for this inequality).

If we obtain a negative answer, we use Lemma 15 (with $x = 0$ and $y = \lceil 4\sqrt{n} \rceil - 1$) to compute the largest t such that α^t is a factor of w in at most $\lceil \log_2(4\sqrt{n}) \rceil = \lceil \frac{\log_2 n}{2} \rceil + 2$ queries. Then we apply Lemma 13 to find a non-right-extendable factor s in $(k-1)(|s|+2)$ queries and we finally apply Lemma 12 to find w in $(k-1)(n-|s|)$ queries. In this case we need a total of $(k-1)(n+2) + \lceil \frac{\log_2 n}{2} \rceil + 3$ queries including the initial query. ◀

5 Conclusion

We have studied three reconstruction problems and, for each of them, we have improved upper bounds on the number of necessary queries. For reconstruction of a word w of length n over an alphabet of cardinality k using $\exists$-subword queries, we have a lower bound $n \log_2(k)$ and in Section 3, we reduce the gap between the lower and the upper bound to an $O(k \log_2(n))$. An open question is whether this gap can be further reduced to an $O(k)$ number of queries or even lower.

For the reconstruction using #-subword queries as considered in Section 2, up to our knowledge, no lower bound is known. Our upper bound is much lower than the previous one, but it could still be far from the truth. In particular, we showed that there exists a deterministic algorithm that requires in average $O(\log n)$ queries to reconstruct a uniform random binary word of length n, but this algorithm requires $\Theta(\sqrt{n \log n})$ queries in the worst case. This might be possible to find a deterministic algorithm that requires $O(\log n)$ queries in the worst case. We were not able to find a simple proof that this cannot be done in constant time only depending on the size of the alphabet.

For the reconstruction using $\exists$-factor queries as considered in Section 4, a simple counting argument yields the lower bound $n \log_2(k)$ on the number of queries. S. S. Skiena and G. Sundaram provide in [10] a lower bound in $kn/4 - o(n)$ queries which is better for large alphabets. In the binary case, we were able to improve the gap between the lower and the upper bound, reducing it to $\left\lceil \frac{\log_2(n)}{2} \right\rceil + 5$. In the general case, even if our result improves the gap between the lower and upper bounds, this gap is still important. As already mentioned in the introduction, the lower bound $kn/4 - o(n)$ given by S. S. Skiena and G. Sundaram is also valid if one considers queries in the form "What is the number of occurrences of u as a factor of w?". In some sense, considering the numbers of occurrences of factors does not bring a significant amount of extra-information for reconstruction comparatively to information on the existence of factors. This contrasts with the subword case where the number of occurrences gives much more information than the existence of occurrences.

To end, let us mention the existence, in the binary case, of a deterministic algorithm that requires, in average, $n + \mathcal{O}(1)$ $\exists$-factor queries over a uniform random word [4] which is optimal up to an additive constant. The main idea of this algorithm is similar to the approach

used in Section 4, but the length t of the longest block of 0 is determined faster. Indeed, for a binary word of length n taken uniformly at random, the average value of $|t - \log_2(n)|$ is in $\mathcal{O}(1)$. The existence of a deterministic algorithm using an $n + \mathcal{O}(1)$ number of $\exists$-factor queries in the worst case is open.

References

1 M. Dudik and L.J. Schulman. Reconstruction from subsequences. *J. Combin. Theory Ser. A*, 103:337–348, 2003.

2 P. Fleischmann, M. Lejeune, F. Manea, D. Nowotka, and M. Rigo. Reconstructing words from right-bounded-block words. In N. Jonoska and D. Savchuk, editors, *Developments in Language Theory - 24th International Conference, DLT 2020, Tampa, FL, USA, May 11-15, 2020, Proceedings*, volume 12086 of *Lecture Notes in Computer Science*, pages 96–109. Springer, 2020. doi:10.1007/978-3-030-48516-0_8.

3 P. Fleischmann, M. Lejeune, F. Manea, D. Nowotka, and M. Rigo. Reconstructing words from right-bounded-block words. *Internat. J. Found. Comput. Sci.*, 32(6):619–640, 2021. doi:10.1142/S0129054121420016.

4 Kazuo Iwama, Junichi Teruyama, and Shuntaro Tsuyama. Reconstructing Strings from Substrings: Optimal Randomized and Average-Case Algorithms. *arXiv e-prints*, 2018. arXiv:1808.00674.

5 L.O. Kalashnik. The reconstruction of a word from fragments. In *Numerical Mathematics and Computer Technology, Preprint IV*, pages 56–57. Akad. Nauk. Ukrain. SSR Inst. Mat., 1973.

6 I. Krasikov and Y. Roditty. On a reconstruction problem for sequences. *J. Combin. Theory Ser. A*, 77:344–348, 1997.

7 V. I. Levenshtein. Efficient reconstruction of sequences from their subsequences or supersequences. In *J. Combin. Theory Ser. A*, volume 93, pages 310–332, 2001.

8 M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, 1983. Reprinted in the *Cambridge Mathematical Library*, Cambridge University Press, UK, 1997.

9 S. Skiena and G. Sundaram. Reconstructing strings from substrings (extended abstract). In F. Dehne, J.-R. Sack, N. Santoro, and S. Whitesides, editors, *Proceedings of the third workshop an Algorithms and Data Structures (WADS '93), Montréal, Canada, August 11-13*, number 709 in Lecture Notes in Comput. Sci., pages 565–576. Springer-Verlag, Berlin, 1993.

10 S.S. Skiena and G. Sundaram. Reconstructing strings from substrings. *J. Comput. Bio.*, 2(2):333–353, 1995.

A Proof of Lemma 9

Proof of Lemma 9. Assume first that n is unknown. We start by finding M the smallest power of 2 larger than $|w|_\alpha$. This can be done asking whether α^i is a subword of w starting from $i = 1$ and doubling i while the answer is positive. The upper bound is reached by $M = i$ when the answer is negative.

If $M = 1$, then $|w|_\alpha = 0$ and exactly one query was asked (and $1 \leq 2\lfloor 1 + \log_2(|w|_\alpha + 1)\rfloor$ as desired). Otherwise, $M = 2^{\lfloor \log_2 |w|_\alpha \rfloor + 1}$ is found in $\lfloor \log_2 |w|_\alpha \rfloor + 2$ queries. In this case we know, $M/2 \leq |w|_\alpha < M$, and we can find the value of $|w|_\alpha$ by binary search. The interval $\{M/2, \ldots, M-1\}$ contains $2^{\lfloor \log_2 |w|_\alpha \rfloor}$ values, hence the binary search requires $\lfloor \log_2 |w|_\alpha \rfloor$ $\exists$-subword queries. In the whole process $|w|_\alpha$ can be found using $2\lfloor 1 + \log_2 |w|_\alpha \rfloor \leq 2\lfloor 1 + \log_2(|w|_\alpha + 1)\rfloor$ $\exists$-subword queries as desired.

When n is known, n is an upper bound on $|w|_\alpha$ and the binary search can be done in the interval $[0, n]$. Hence $|w|_\alpha$ can be determined using at most $\lceil \log_2(n+1)\rceil$ $\exists$-subword queries. ◀

B Proof of Lemma 15

Proof of Lemma 15. We proceed by induction on the value $y+1-x$. If $x=y$ then we know the value of t and no more queries are needed as expected. If $y>x$, then we ask the query "is $a^{\lceil (x+y)/2 \rceil}$ a factor of w?".

We deduce $x' \leq t \leq y'$ where, if the answer is "yes", $x' = \lceil (x+y)/2 \rceil$ and $y'=y$ and, if the answer is "no", $x'=x$ and $y' = \lceil (x+y)/2 \rceil - 1$. In the two cases,

$$y'-x'+1 \leq 1 + \lfloor (y-x)/2 \rfloor \,. \tag{5}$$

The map $f : z \mapsto \lfloor \frac{z-1}{2} \rfloor + 1$ is non-decreasing over the non-negative reals and for all integers n, $f(2^n) = 2^{n-1}$, thus for all $z \leq 2^n$, we have $f(z) \leq 2^{n-1}$. Since (5) can be rewritten, $y'-x'+1 \leq f(y+1-x)$, we deduce that for all integers n, if $y+1-x \leq 2^n$ then $y'+1-x' \leq 2^{n-1}$. In particular, choosing $n = \lceil \log_2(y+1-x) \rceil$ yields, $y'+1-x' \leq 2^{\lceil \log_2(y+1-x) \rceil - 1}$, hence

$$\lceil \log_2(y'+1-x') \rceil \leq \lceil \log_2(y+1-x) \rceil - 1 \,.$$

By induction hypothesis, it implies that we need at most $\lceil \log_2(y+1-x) \rceil - 1$ other queries to determine the value of t. With the initial query, this is a total of at most $\lceil \log_2(y+1-x) \rceil$ queries as desired. ◀

Dynamic Binary Search Trees: Improved Lower Bounds for the Greedy-Future Algorithm

Yaniv Sadeh
Tel Aviv University, Israel

Haim Kaplan
Tel Aviv University, Israel

Abstract

Binary search trees (BSTs) are one of the most basic and widely used data structures. The best static tree for serving a sequence of queries (searches) can be computed by dynamic programming. In contrast, when the BSTs are allowed to be dynamic (i.e. change by rotations between searches), we still do not know how to compute the optimal algorithm (OPT) for a given sequence. One of the candidate algorithms whose serving cost is suspected to be optimal up-to a (multiplicative) constant factor is known by the name Greedy Future (GF). In an equivalent geometric way of representing queries on BSTs, GF is in fact equivalent to another algorithm called Geometric Greedy (GG). Most of the results on GF are obtained using the geometric model and the study of GG. Despite this intensive recent fruitful research, the best lower bound we have on the competitive ratio of GF is $\frac{4}{3}$. Furthermore, it has been conjectured that the additive gap between the cost of GF and OPT is only linear in the number of queries. In this paper we prove a lower bound of 2 on the competitive ratio of GF, and we prove that the additive gap between the cost of GF and OPT can be $\Omega(m \cdot \log \log n)$ where n is the number of items in the tree and m is the number of queries.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Sorting and searching

Keywords and phrases Binary Search Trees, Greedy Future, Geometric Greedy, Lower Bounds, Dynamic Optimality Conjecture

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.53

Related Version *Full Version*: `https://arxiv.org/abs/2301.03084`

Funding The work of the authors is partially supported by Israel Science Foundation (ISF) grant number 1595-19, German Science Foundation (GIF) grant number 1367 and the Blavatnik research fund at Tel Aviv University.

1 Introduction

Binary search trees (BSTs) are one of the most basic and widely used data-structures. They are used to store a sorted set of keys from a totally ordered universe. Traversing BSTs is usually done by using a single pointer, initially pointing to the root, and moving to the left or right child according to the order of the searched key and the key of the item at the current node. Therefore, we typically define the cost[1] of a search to be the length of the search path. The data structure itself may be static, or change dynamically throughout time, in response to insertions and deletions of items, and possibly even restructured during queries.

Static BSTs are well understood. One can guarantee that the longest path from the root to a leaf is of length $O(\log n)$ if the number of keys is n, by using a balanced tree. If the access sequence is known in advance (in fact only the frequency of accesses of each key matters) then an $O(n^2)$ time algorithm computing the optimal static tree for the particular set of

[1] Our cost model is formally defined in Definition 1, in Section 2.

© Yaniv Sadeh and Haim Kaplan;
licensed under Creative Commons License CC-BY 4.0

frequencies was given by Knuth [13]. It is also notable that the lower bound on the cost when the known frequencies are $\vec{f} = [f_1, f_2, \ldots, f_n]$ and the number of queries is m, is $\Omega(m \cdot H(\vec{f}))$ where $H(\vec{f}) = \sum_{i=1}^{n} f_i \log \frac{1}{f_i}$ is the entropy function. A simple way with $O(n \log n)$ running time to construct a near-optimal static (centroid) tree whose cost is $O(m \cdot H(\vec{f}))$, has been described by Mehlhorn [17]. The running time has been improved to $O(n)$ by Fredman [10].

In contrast to the static case, the dynamic case is less understood. One can, of course, serve the sequence with a static tree. But, for many sequences we must change the structure of the tree as we make the searches in order to be efficient. For example, the requested items may be different in different parts of the sequence so a different set of items has to be placed near the root during different parts of the sequence. Restructuring is done by rotations that maintain the symmetric order. When rotations are allowed, the cost is defined to be the size of the subtree that contains the search path and all edges which we rotate.

Here, we assume that the set of values stored in the tree does not change (no insertions or deletions), yet restructuring the tree is allowed to speed up future searches. One famous dynamic algorithm for doing this is the *Splay* algorithm of Sleator and Tarjan [20]. After each query, the splay algorithm moves the queried item to the root of the tree, according to three simple rules called *zig-zag*, *zig-zig* and *zig*. The splay algorithm is efficient in the sense that it is able to exploit the structure of many families of sequences. In particular splay is proven to be as good as the static optimum (up to a constant factor), which also implies that the cost of splay on any given sequence is at most $O(\log n)$ times the (dynamic) optimum cost. Sleator and Tarjan conjectured that splay is in fact dynamically-optimal, meaning that its cost is like the cost of an optimal algorithm that knows the whole sequence of queries in advance, up to some constant factor. However, this dynamic-optimality conjecture of splay is still open. In fact, it is open whether there is any dynamically-optimal online binary search tree algorithm. The best competitive ratio achievable to date is $O(\log \log n)$, and it is obtained by Tango [8], Multi-splay [21] and Chain-splay [11] trees, and a geometric divide-and-conquer approach of [1].

While seeking for (better) guaranteed competitiveness, other dynamic algorithms were considered. A promising candidate was independently proposed by Lucas [16] and Munro [18], which is now commonly referred to as *Greedy Future*, henceforth: GF in short. As its name suggests, GF is a greedy algorithm that rearranges the nodes on the path from the root to the current queried item as a treap whose priorities are according to the future accesses[2] (as this paper deals with analyzing GF, we detail it formally in Algorithm 1). Note that unlike splay, GF, by definition, is required to know the future in order to restructure the tree. Surprisingly however, Demaine et al. [7] showed that one can simulate GF without knowing the future by a hierarchy of split-trees while losing only a constant factor in performance.

Additionally, [7] presented a geometric view of an algorithm serving queries by a dynamic binary search tree using a two dimensional grid on which we mark the sequence as well as the items accessed by the algorithm. In this presentation there is yet another natural promising candidate for dynamic optimality, which is commonly known as *Geometric Greedy* and sometimes simply *Greedy*, which we shall refer to as GG. [7] showed that GG is in fact the same algorithm as GF.

[2] Each item in a treap has two keys: *value* and *priority*. The treap is a binary search tree with respect to the values of the items and a heap with respect to their priorities. That is, the priority of an item is no larger than the priorities of its children. In our case, the priorities are deterministically defined by future requests in a way that we define precisely in Algorithm 1.

The geometric view proved useful to obtain new results regarding GG and hence GF. Fox [9] proved that an *access-lemma* that is analogues to the so called *access-lemma* of splay trees holds for GG. From this follows that most of the nice properties that hold for splay also hold for GF. In particular, it follows that GF is $O(\log n)$ competitive. Chalermsook et al. [3] analyzed upper bounds on the cost of GG for access patterns which are permutations, and in particular found that for highly structured permutations, which they called *k-decomposable*, the cost is $n \cdot 2^{\alpha(n)^{O(k)}}$ where $\alpha(n)$ is the inverse-Ackermann function. Chalermsook et al. [5] study special access patterns that belong to a broader family of *pattern-avoiding* permutations. See [4] for a survey of currently known properties of greedy and splay.

Our Contributions.

1. It is known that GF is not exactly optimal, but it is conjectured, like splay, to be optimal up to a constant factor. In fact, it has been even more strongly conjectured by Demaine et al. [7] to be optimal up to an additive $O(m)$ term, and possibly even exactly m. Kozma [14] refuted the second part and gave a specific sequence for which this additive gap is $m + 1$. In this paper we refute the linear gap conjecture and show a family of sequences for which the additive gap is at least $\Omega(m \log \log n)$.
2. The largest lower bound on the competitive ratio of GF is $\frac{4}{3}$ by Demaine et al. [7]. They show a family of sequences on which after an initial query, the optimum pays 1.5 on average per query while GF pays 2.[3] We describe a technique that allows us to improve this lower bound to 2. We note that the best known lower bound on the competitive ratio of splay is 2 (see [15, Section 2.5]). In both cases, the construction requires a rather large number of items (large n).
3. Based on the multiplicative lower bound described above we show the following two interesting properties of GF: (1) There are sequences X such that the cost of GF on the reverse sequence is twice larger than the cost of GF on X. (2) There are sequences X such that we can remove some queries from them and get a subsequence X', such that the cost of GF on X' is twice larger than the cost of GF on X.

We study subsequences and reversal (contribution 3) since any dynamically-optimal algorithm A must have a "nice" behavior in these cases. Concretely, A must satisfy the approximately-monotone property (Definition 8) which states that there is a fixed constant c such that the cost of A on any subsequence of any sequence is never more than c times the cost on the whole sequence. As for reversal, the optimum can process a sequence and its reversal with similar costs up to a difference of n, thus any dynamically-optimal algorithm must be able to do so with costs that differ by at most a constant factor. We discuss this motivation in more detail in Section 3 (right after stating Theorem 7).

Our contributions are all based on the same technique, which is quite simple. We enforce GF to maintain a static tree and only query the leaves of this tree. Although being dynamic in general, there are some access-patterns that cause GF not to change the tree. By studying these patterns, we can study GF on a static tree, and the analysis of its cost simplifies to the weighted-average of the depth of the queries (weighted by frequency). To lower-bound the gap between GF and OPT, we analyze the average cost that can be saved by promoting the items in the leaves to locations closer to the root. Note that any other item can be placed further away from the root since it is never queried by the sequence.

[3] Reddmann [19] found an example in which the cost ratio between GF and the optimum is $\frac{26}{17} \approx 1.53$. But this is for one particular sequence of a fixed length so it does not rule out any competitive ratio if we allow an additive constant.

2 Model

In this section we describe the model which we use, and define our notations. First, we note that throughout the paper $\lg x$ is used to denote the base two logarithm of x.

We consider a totally ordered universe of (fixed size) n items. For simplicity, one may think of the values $\{1, \ldots, n\}$. The items are organized in some initial BST which we denote by T_0. Then, a sequence of queries, denoted by $X = [x_1, x_2, \ldots, x_m]$, is given, one query at a time. We reserve m to denote the length of the sequence. The tree before serving x_t is denoted by T_{t-1}. An algorithm has to find the queried value x_t, by traversing T_{t-1} from its root. After finding x_t, the algorithm is allowed to re-structure T_{t-1} to get T_t. We define the cost of the algorithm at time t to be the total number of nodes that were touched at time t, both on the path to x_t and for restructuring. The cost of an algorithm for the whole sequence is simply the sum of its costs over all times. We define it formally below.

▶ **Definition 1** (Cost). *Let X be a sequence of queries, and let T_0 be an initial tree. Let A be an algorithm that serves X and let T_t be the tree that A has after serving x_t. Let P_t be the set of nodes on the path from the root to x_t in T_{t-1} and let U_t be the set of nodes of the minimal subtree that contains all the edges that were rotated by A to transform T_{t-1} to T_t. Then the cost of A for serving X at time t is $|P_t \cup U_t|$, and the cost of A for serving X is the sum of costs over $t = 1, \ldots, m$. We denote the cost of A to serve X starting with T_0 by $cost(A, X, T_0)$. We denote the average cost per query by $\hat{c}(A, X, T_0) = \frac{cost(A,X,T_0)}{m}$. When T_0 is clear from the context, or immaterial, we write $cost(A, X)$ and $\hat{c}(A, X)$.*

▶ **Definition 2** (Depth). *Let T be a tree. The depth of a node $v \in T$, denoted by $d(v)$, is the number of edges in the path from the root to v (in particular $d(root) = 0$). Note that the cost of querying v (without restructuring) is $d(v) + 1$. We also define the depth of the tree, denoted by $d(T)$, as the maximum depth of a node in T, that is $d(T) = \max_{v \in T} d(v)$.*

▶ **Definition 3** (Competitiveness). *We say that an algorithm A is (α, β)-competitive for initial tree T_0 if for any sequence of queries X, it holds that $cost(A, X, T_0) \leq \alpha \cdot cost(OPT, X, T_0) + \beta$ where OPT is a best algorithm to serve X given T_0 (with full knowledge of X). When we do not specify T_0 we mean that the relation holds for all initial trees. We refer to α as the* multiplicative term *and to β as the* additive term. *For ease of language, we regard the multiplicative term as the* competitive ratio, *and also write "the competitive ratio of" instead of "the multiplicative term of the competitiveness of". In such cases, we assume that the additive term is $o(m)$. It is easiest to think of $\beta = O(n)$ while assuming that $m = \omega(n)$.*

To conclude this section, we give a precise description of the GF algorithm, in Algorithm 1. We emphasize that its implementation is complex and probably would not be good in practice. However, its main benefit is its theoretical value, as a candidate for dynamic optimality. Should it be proven to be dynamically-optimal, then we would get a better understanding of the problem and also a stepping-stone to analyze simpler algorithms, such as splay, in comparison to GF rather than against some "vague" optimum that depends on the sequence.

3 Stable Sequences and Lower Bounds

In this section we properly define the family of *stable sequences* (Definition 11) for which the tree maintained by GF is never changed (i.e. the access path of the current query is a treap with respect to the suffix of the sequence). To prove our lower bounds we use such sequences in which only the items at the leaves of GF are requested, and the internal nodes

Algorithm 1 GreedyFuture (GF) Algorithm.

Input: A sequence of queries $X \in [n]^m$ and an initial BST T_0. We restructure T_{t-1} to T_t after serving the request x_t with T_{t-1} for $t = 1, \ldots, m$.

Function `Restructure`(*query value* v, *current tree* T_{t-1}, *future accesses* X')**:**

Let $v_1 < v_2 < \ldots < v_k$ be the nodes on the path from the root of T_{t-1} to the queried value v (including v and the root). We also define $v_0 = -\infty$ and $v_{k+1} = +\infty$. Denote the subtrees hanging off this path by $R_0, \ldots, R_k$.

For each $i = 1, \ldots, k$, let $\tau(v_i)$ be the index of the first appearance of a query of a value $x \in (v_{i-1}, v_{i+1})$ in X'. Restructure the nodes $v_1, \ldots, v_k$ as a treap: maintain a BST ordering, while the heap's priorities are set to be the τ values, where the root's τ is smallest. Tie-break arbitrarily, e.g. in favor of smaller values, or smaller depth prior to restructuring. Then, hang the subtrees $R_0, \ldots, R_k$ unchanged at their appropriate locations. The resulting tree is T_t.

cause some extra cost that OPT avoids. We use a natural way to represent such sequences as trees, and use this representation to prove the following lower bounds, which are the main results of this section.

▶ **Theorem 4.** *If GF is (c, d)-competitive where the additive term d is sublinear in the length of the sequence, i.e. $d = o(m)$, then $c \geq 2$.*

▶ **Theorem 5.** *For every $n \geq 2$ there exist sequences $X \in [n]^m$ such that $cost(GF, X) = cost(OPT, X) + \Omega(m \cdot \lg \lg n)$. Among these sequences, there exists a sequence whose length is $m = n^{\Theta(\frac{\lg \lg n}{\lg \lg \lg n})}$. (There exist other longer sequences too.)*

Theorem 4 enables us to prove the following two theorems, proven in Appendix A.2.

▶ **Theorem 6.** *For any $\epsilon > 0$ there exists a sequence X with a subsequence (not necessarily consecutive) $X' \subseteq X$ such that $cost(GF, X') \geq (2 - \epsilon) \cdot cost(GF, X)$.*

▶ **Theorem 7.** *Let S be a sequence, we define $rev(S)$ to be the sequence S in reverse. For any $\epsilon > 0$ there exists a sequence X such that $cost(GF, rev(X)) \geq (2 - \epsilon) \cdot cost(GF, X)$.*

The motivation for studying subsequences (Theorem 6) is the fact that OPT always saves costs when queries are removed from its sequence. Formally, if $X' \subseteq X$, then $cost(OPT, X') \leq cost(OPT, X)$. Indeed, OPT can serve X' by simulating a run on X. More generally, this relation of costs when comparing a sequence to a subsequence of it, is an important property which even has a name:

▶ **Definition 8** (Approximate-monotonicity [12, 15])**.** *An algorithm A is approximately-monotone with a constant c if for any sequence X, subsequence $X' \subseteq X$, and initial tree T, it holds that $cost(A, X', T) \leq c \cdot cost(A, X, T)$.*

▶ **Corollary 9.** *If GF is approximately-monotone with a constant c, then $c \geq 2$.*

As noted, OPT is approximately-monotone with $c = 1$ (strictly monotone). The reason that approximate-monotonicity is of interest, in particular for GF, is because it is one of two properties that together are necessary and sufficient for any dynamically-optimal algorithm. The complementing property, which GF is known to satisfy, is simulation-embedding:

▶ **Definition 10** (Simulation-Embedding [15]). *An algorithm A has the simulation-embedding property with a constant c if for any algorithm B and any sequence X, there exists a supersequence $Y \supseteq X$ such that $cost(A, Y) \leq c \cdot cost(B, X)$. ($X$ is a subsequence of Y, not necessarily of consecutive queries.)*

An algorithm A which is approximately-monotone with a constant c_1 and has the simulation-embedding property with a constant c_2 is dynamically-optimal with a constant $c_1 \cdot c_2$. Indeed, for any sequence X, there is some supersequence $Y(X) \supseteq X$ such that $cost(A, X) \leq c_1 \cdot cost(A, Y(X)) \leq c_1 \cdot c_2 \cdot cost(OPT, X)$. Harmon [12] proved that GG, and hence GF, has the simulation-embedding property, hence GF is dynamically-optimal if and only if it is approximately-monotone. An alternative indirect proof was given by [6], proving that GG is $O(1)$-competitive versus the move-to-root algorithm, therefore inheriting the property from move-to-root.

The motivation for studying reversal (Theorem 7) is that OPT is oblivious to reversing the sequence of queries, up to an additive difference of n. Indeed, to serve a sequence X in reverse, we can pay n to restructure the initial tree T_0 to the final tree T_m, and then "reverse the arrow of time": when serving query x_t, also modify the tree from T_t to T_{t-1} where T_i is the tree that OPT would get by the end of processing the i-th query of X, in order. This means that any dynamically-optimal algorithm must be able to serve a sequence of requests and its reverse with the same cost up to a constant factor. Theorem 7 does not disprove dynamic-optimality for GF, but gives some insight of how reversal affects GF.

3.1 Maintaining a Static Tree for *GF*

In this section we describe the basic "tool" which we use to fix a tree structure for GF despite its dynamic nature. That is, we describe a class of sequences which we call *mixed-stable sequences* such that GF never restructures its tree when serving a sequence in this class. For the sake of simplicity, we assume that the initial tree is structured as we need it to be. Appendix A.1 explains how to enforce a specific "initial" tree given an arbitrary initial tree, and also argues why this minor issue does not affect the competitive ratio of GF.

As noted, our objective is to produce a sequence that "tricks" GF into having unnecessary nodes in the core of the tree, such that the requested values are only at the leaves. As an example, consider the classic sequence of queries $X = [1, 3, 1, 3, \ldots]$ with an initial tree containing 2 at the root, 1 as a left child of the root and 3 as a right child of the root. Because of the alternating pattern, GF never re-structures the tree, and the cost per query is 2 rather than 1.5 on average (e.g. when 1 is in the root, and 3 is its right child).

▶ **Definition 11** (Stable Nodes and Sequences). *Let T be a full binary search tree, and let X be a sequence of queries over the items in the leaves of T. We define the stability of nodes as follows, see also Figure 1.*

We say that an inner node v in T is strongly-stable *if it has two children, and the subsequence of X consisting only of the items in the subtree of v, alternates between accesses to the left and right subtrees of v.*

We say that an inner node v with a left child u in T is weakly-stable with a left-bias *if both v and u have two children, and the subsequence of X consisting only of the items in the subtree of v, repeats the following 3-cycle. First it accesses the left-subtree of u, then the right subtree of u, and finally right subtree of v. (It is left-biased because $\frac{2}{3}$ of the accesses are to the left of v). Symmetrically, we say that v is* weakly-stable with a right-bias *if v has two children, its right child u has two children, and the restriction of X to accesses in the*

subtree of v repeats a 3-cycle consisting of an access to the right subtree of u, the left subtree of u, and the left subtree of v. Notice that u is a strongly-stable node by definition, and we refer to it as the favored-child *of v.*

We regard the sequence X as being induced by the tree T with stability "attached" to its inner nodes. We assume that every node is stable, and refer to X as a mixed-stable *sequence and to T as a* mixed-stable *tree. We distinguish two special cases: If all inner nodes are strongly-stable then we refer to X and T as* strongly-stable*, and if exactly half of the inner nodes of T are weakly-stable then we refer to X and T as* weakly-stable *(recall that each weakly-stable node has a strongly-stable favored-child).*

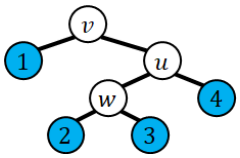


Figure 1 Node and sequence stability (Definition 11). First, consider the repeated sequence 421, i.e. $X = 421421421\ldots$. Then v is a weakly-stable right-biased node because its visits pattern is a repetition of $right(u), left(u), left(v)$. u is a strongly-stable node because its visits pattern is $right(u), left(u)$. w is not stable at all, because its visits pattern is always $left(w)$. Second, consider the repetition of the access pattern 12141314. One can verify that all three inner nodes are strongly-stable. Hence, this is a strongly-stable sequence. Third, note that no weakly-stable sequence corresponds to the figure, because it requires an even number of inner nodes, but if we make w a leaf (removing $2, 3$), then the repeated access pattern of $4w1$ is a weakly-stable sequence.

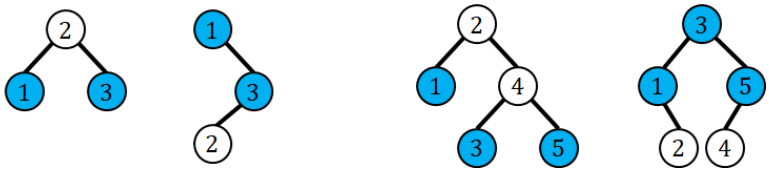
To motivate Definition 11 a little, note that the sequence $X = [1, 3, 1, 3, \ldots]$ is a strongly-stable sequence that corresponds to a tree over the items $\{1, 2, 3\}$ where 2 is in the root. X yields a lower-bound of $\frac{4}{3}$ on the competitive ratio of GF. Similarly, the sequence $X' = [5, 3, 1, 5, 3, 1, \ldots]$ is a weakly-stable sequence that corresponds to the tree over $\{1, 2, 3, 4, 5\}$ with 2 at the root and 4 its right-child. X' yields a lower-bound of $\frac{8}{5}$ on the competitive ratio of GF, which is already an improvement over the best known lower bound, see also Figure 2. The distinction between strongly-stable and weakly-stable nodes is that GF may modify the structure of the tree when a weakly-stable node is considered, but only temporarily and without affecting the cost. In our example with X', after querying 5, GF may put 4 in the root instead of 2, but following the query of 3 this change will be reverted.

Motivated by the power of stable sequences over small trees, we proceed to a more general analysis of stable sequences.

▶ **Definition 12** (Atomic Sequence). *A tree T, along with stability type (weak/strong) for each node, and a subtree of each node to be accessed initially, induce a stable sequence. This sequence is unique up to its length, which can be extended indefinitely. We define the "atomic unit" of this sequence as the shortest sequence X such that any repetition of X is also a stable sequence that corresponds to T.*

Throughout the paper we work with whole multiples of the atomic sequence. Moreover, unless stated otherwise, we work with the atomic sequence itself (a single repetition).

▶ **Lemma 13.** *Let X be a mixed-stable sequence with respect to a tree T. Then every leaf u is visited once every $2^{a(u)} \cdot 3^{b(u)}$ queries where $a(u)$ and $b(u)$ are non-negative integers. In particular, the atomic length of X is $2^{\max_{leaf\ u} a(u)} \cdot 3^{\max_{leaf\ u} b(u)}$ (the lcm). Moreover, if X is strongly-stable then $\forall u : b(u) = 0$, and if X is weakly-stable then $\forall u : a(u) = 0$.*



(a) $X = [1, 3, 1, 3, \ldots]$. **(b)** $X' = [5, 3, 1, 5, 3, 1, \ldots]$.

Figure 2 Examples of the simplest strongly-stable (a) and weakly-stable (b) sequences. Their corresponding trees are the left tree in each pair while the right tree in each pair is an optimized static tree to serve the same sequence. Queried nodes are colored in blue. One can verify that $\hat{c}(X, GF) = 2$ and $\hat{c}(X', GF) = \frac{8}{3}$ while based on the optimized tree, $\hat{c}(X, OPT) \leq \frac{3}{2}$ and $\hat{c}(X', OPT) \leq \frac{5}{3}$.

Proof. Consider a leaf u. Define the frequency of visiting an ancestor w of u to be the frequency of accessing a leaf in the subtree of w. If w is a strongly-stable ancestor then the frequency of visiting a child of w is $\frac{1}{2}$ of the frequency of visiting w. If w is weakly-stable, v is its favored-child, and x is a child of v then the frequency of visiting x is $\frac{1}{3}$ of the frequency of visiting w. Similarly if w is weakly-stable, v is its non-favored-child then the frequency of visiting v is $\frac{1}{3}$ of the frequency of visiting w. It follows that u is visited exactly once every $2^{a(u)} \cdot 3^{b(u)}$ queries where $a(u)$ is the number of strongly-stable nodes that are not favored-children (there are no such nodes if X is weakly-stable), and $b(u)$ is the number of weakly-stable nodes (no such nodes if X is strongly-stable), on the path to u. Finally, since every leaf u is visited with a specific period, the whole sequence has a period which is the lcm of all periods. ◀

▶ **Lemma 14.** *Let X be a mixed-stable sequence with respect to a tree T. If GF serves X with T as initial tree, and breaks ties in favor of nodes of smaller-depth, then it never restructures T.*

Proof. The proof is by induction on the size of the tree. If T has a single node, then it is trivial. Otherwise, the root r is an inner-node, and we prove that it always remains the root. It then follows, by restricting the access sequence to values within each subtree, that the rest of the tree remains fixed as well. We use the notations of $\tau(v)$ and v_i as in Algorithm 1.

First, consider the case that r is a strongly-stable node (Definition 11). Given an access to some value x in the left subtree of r, by definition, the next access would be to a value in the right subtree of r, hence $\tau(r) < \tau(v_i)$ for any $v_i \neq r$ on the path from r to x, and therefore GF will keep r in the root. The same argument holds if x is in the right subtree of r, and the next access is in the left subtree.

Next, consider the case that r is a weakly-stable node. Without loss of generality, assume that it is left-biased, and denote its favored-child (left child) by u. Denote the left and right subtrees of u by A and B respectively, and the right subtree of r by C. The access pattern of subtrees is $ABC(ABC\ldots)$.

- If the current access was to some $x \in A$, both r and u have been touched. The next access queries in B, so $\tau(u) = \tau(r) < \tau(v_i)$ for any $v_i \neq u, r$ on the access path to x. Since GF tie-breaks in favor of smaller-depth, it will keep r in the root.[4]

[4] This is the reason we defined this kind of access pattern as weakly-stable, because the stability can be chosen, but is not forced. We emphasize that putting u as a parent of r will not make the next access cheaper as both u and r will be touched anyway, and then r will be reinstated as the root.

- If the current access was to some $x \in B$, then both r and u have been touched. The next access touches C, so $\tau(r) < \tau(v_i)$ for any $v_i \neq r$ on the access path to x, including u, thus r must remain the root.
- If the current access was to some $x \in C$, since the next access touches A, $\tau(r) < \tau(v_i)$ for any $v_i \neq r$ on the access path to x, thus r must remain the root. In this case u was not touched, but nonetheless it remains the left child of r. ◀

▶ **Lemma 15.** *If X is a mixed-stable sequence, the frequency of accessing $x \in X$ is in the range of $[\frac{1}{3^{d(x)}}, \frac{1}{3^{d(x)/2}}]$. In particular, if X is strongly-stable then the frequency equals $\frac{1}{2^{d(x)}}$.*

Proof. The frequency of visiting a node depends on the path to it. The frequency is multiplied by $\frac{1}{2}$ when passing through a strongly-stable node, and multiplied by either $\frac{1}{3}$ or $\frac{2}{3}$ when passing through a weakly-stable node. Every factor of $\frac{2}{3}$ is followed by $\frac{1}{2}$, due to the strongly-stable favored-child of the weakly-stable node. Thus the frequency is bounded between $\frac{1}{3^{d(x)}}$ and $\frac{1}{2^{d(x)/2}} \cdot \left(\frac{2}{3}\right)^{d(x)/2} = \frac{1}{3^{d(x)/2}}$. ◀

▶ **Corollary 16.** *Let X be a strongly-stable sequence, then: $\hat{c}(GF, X) = \sum_{x \in X} \frac{d(x)+1}{2^{d(x)}}$.*

3.2 Promotions and Recursive Trees

The way in which we show our lower bounds relies on the fact that serving the leaves of a static tree is sub-optimal, since a trivial static optimization is to move the leaves closer to the root. We refer to this operation as a *promotion* of the leaf that we move. We emphasize that for the purpose of our result, we analyze the improvement one gets from promotions, but the actual OPT, which is dynamic, may be able to reduce the cost further.

▶ **Definition 17** (Promotion). *Consider trees T and T'. We say that a node x was promoted in T' by h (with respect to T), if $d_T(x) - d_{T'}(x) = h$. Given a mixed-stable sequence X, the average promotion of T to T' is the weighted average promotion in T' of the nodes of T, weighted by the query frequencies of the nodes.*

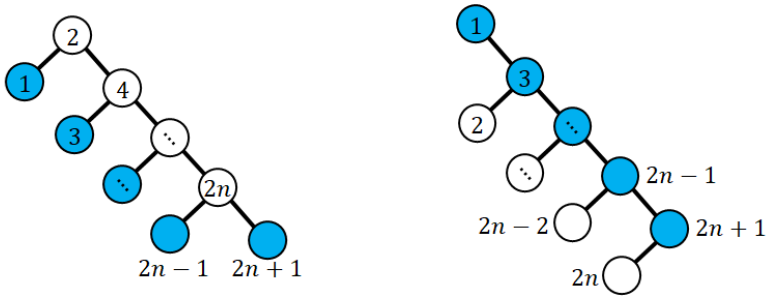
By definition, static optimization of a tree T to T' for a mixed-stable sequence X, implies a cost improvement for OPT which is at least the average promotion of T to T', per query. Intuitively, promoting leaves that are closer to the root contributes more to the average promotion than promoting deeper leaves since the access frequencies decrease exponentially with depth. That being said, our promotion scheme will be relatively uniform, promoting most leaves by roughly the same amount, as in the following example.

▶ **Example 18.** To clarify promotions, consider Figure 3. There, we can safely promote every node by one, except for one of the deepest nodes. Therefore, we immediately conclude that for the corresponding strongly-stable sequence X, we have: $\hat{c}(GF, X) \geq \hat{c}(OPT, X) + (1 - \frac{1}{2^n})$.

We define our trees using recursive structures.

▶ **Definition 19.** *A recursive tree, T_r, of depth r is defined by a specific full binary tree T (independent of r) such that at least one of its leaves is an actual leaf, and some of its leaves are roots of recursive trees, T_{r-1}, of depth $r-1$. We refer to the inner nodes of T as the* trunk *of T_r, and define T_0 to be a single node. See Figure 4 for two examples.*[5]

[5] The name of the pattern F in Figure 4, stands for Fibonacci: One can verify that for $r \geq 2$, the number of leaves at depth $1 \leq d \leq r-1$ is the $(d-1)$th Fibonnaci number F_{d-1} (we define $F_0 = 0$). Moreover, this can be used to prove the nice equation: $\sum_{d=0}^{\infty} \frac{F_d}{2^d} = 2$.



(a) Before promotions. **(b)** After promotions.

Figure 3 (a) A tree which induces a strongly-stable sequence X, only blue nodes are queried. The frequency of querying an odd number $v = 2i - 1$ in this tree is $\frac{1}{2^i}$ except for $v = 2n + 1$ which has the same frequency as $v = 2n - 1$. (b) An improved static tree, in which each node except for one has been promoted one step closer to the root. The cost of serving X over this tree is cheaper by almost 1 per query.

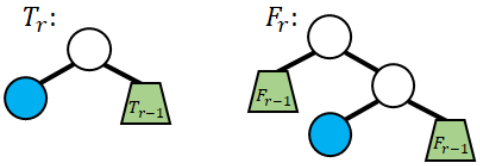


Figure 4 Two recursive trees of depth r. Each of the trees T and F is a full binary tree with at least one actual leaf (in blue), and some hanging subtrees. At the bottom of the recursion (for $r = 0$), the subtrees are nodes. Note that: (a) Expanding T for $r = n$ results in the tree in Figure 3; (b) The pattern F is important for Theorem 4.

3.3 Multiplicative Lower Bound for *GF*

In this section we prove Theorem 4. We do it by describing a concrete weakly-stable sequence, whose average cost per query is 6 while an average promotion of 3 is possible, resulting in an optimal cost of at most 3. We start by stating a purely mathematical lemma that will be used in the analysis.

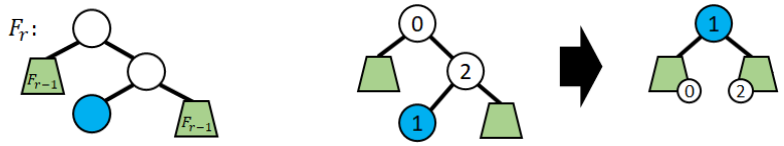
▶ **Lemma 20.** *Let b_r be a sequence defined by an initial value b_0 and the relation $b_r = \alpha \cdot b_{r-1} + \beta + \gamma \cdot \frac{r}{2^r}$ for some constants α, β, γ where $\alpha \neq \frac{1}{2}, 1$. Then $b_r = \frac{\beta}{1-\alpha}(1-\alpha^r) + \alpha^r \cdot b_0 + \frac{2\alpha\gamma}{(2\alpha-1)^2} \cdot (\alpha^r - \frac{1}{2^r}) - \frac{\gamma}{(2\alpha-1)} \cdot \frac{r}{2^r}$. In particular, when $\gamma = 0$ then $b_r = \frac{\beta}{1-\alpha}(1-\alpha^r) + \alpha^r \cdot b_0$.*

Proof Sketch. Either use induction, or "guess" that a geometric sequence y_r with a multiplier of α satisfies $y_r = p \cdot \frac{r}{2^r} + q \cdot \frac{1}{2^r} + s + b_r$, and determine the fixed coefficients p, q, s. ◀

▶ **Lemma 21.** *Let X be a weakly-stable sequence implied by the recursive tree F_r in Figure 4, where the root is a weakly-stable node with a right-bias. Then for any $\epsilon > 0$, there is a sufficiently large recursive depth r such that (1) $\hat{c}(GF, X) > 6 - \epsilon$, (2) a static optimization of the tree saves an average cost of at least $3 - \epsilon$, and (3) regardless of r, $\hat{c}(OPT, X) < 3$.*

Proof. Let c_r denote the average cost of serving X with F_r. Then $c_0 = 1$ and $c_r = \frac{1}{3}(c_{r-1} + 1) + \frac{1}{3} \cdot 3 + \frac{1}{3}(c_{r-1} + 2) = \frac{2}{3}c_{r-1} + 2$, which yields by Lemma 20 that $c_r = \frac{2}{1-2/3}(1 - (2/3)^r) + (2/3)^r \cdot 1 = 6 \cdot (1 - (2/3)^r) + (2/3)^r$. To analyze the average promotion, we re-structure F_r to a new static structure F'_r as follows, see Figure 5. The leaf is moved to the root, whose children are the recursive subtrees, optimized themselves by the same

logic. The old root is moved to be a right child of the maximal value in the new left subtree, and the old right-child (of the old-root) is moved to be a left child of the minimal value in the new right subtree. F'_r maintains the order of values as was in F_r. The demotions of the old root and its right child do not affect the cost, because X does not query these values. Denote by p_r the average promotion of F_r to F'_r. Then $p_0 = 0$ since nothing is promoted for a singleton, and $p_r = \frac{1}{3}p_{r-1} + \frac{1}{3}\cdot 2 + \frac{1}{3}(p_{r-1}+1) = \frac{2}{3}p_{r-1} + 1$. Again by Lemma 20 we get that $p_r = \frac{1}{1-2/3}(1-(2/3)^r) + (2/3)^r \cdot 0 = 3\cdot(1-(2/3)^r)$. Observe that for $r \to \infty$ we get that $c_r \to 6$ and $p_r \to 3$, thus parts (1) and (2) of the claim follow. For part (3), observe that $c_r - p_r = 6\cdot(1-(2/3)^r) + (2/3)^r - 3\cdot(1-(2/3)^r) = 3 - 2\cdot(2/3)^r < 3$. ◀



(a) F-tree pattern. **(b)** Promotion scheme.

Figure 5 The F-tree pattern and its promotion scheme in Lemma 21. Only the top-level promotions are presented in (b), but more promotions are done recursively within each subtree.

▶ **Theorem 4.** *If GF is (c,d)-competitive where the additive term d is sublinear in the length of the sequence, i.e. $d = o(m)$, then $c \geq 2$.*

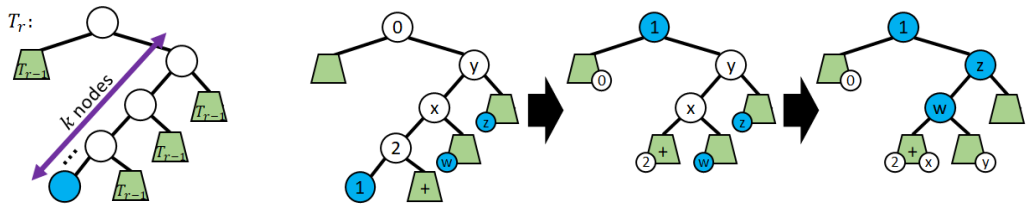
Proof. Assume by contradiction that GF is $(2-\delta, f(m))$-competitive for some $\delta > 0$ and a function $f(m) = o(m)$. Let X' be a sequence that consists of s repetitions of the atomic weakly-stable sequence that corresponds to the recursive tree F_r. It follows that $\hat{c}(GF, X') \leq (2-\delta)\cdot\hat{c}(OPT, X') + \frac{f(|X'|)}{|X'|}$. By Lemma 21, we can choose r large enough such that $\hat{c}(GF, X') > 6-\delta$, and regardless of r, $\hat{c}(OPT, X') < 3$. Then, since f is sub-linear, we can choose the number of repetitions s to be large enough such that $\frac{f(|X'|)}{|X'|} < 2\delta$. But then we also get that $\hat{c}(GF, X') < (2-\delta)\cdot 3 + 2\delta = 6 - \delta$, which is a contradiction. ◀

By Analyzing mixed-stable sequences we proved a lower bound of 2 on the competitve ratio of GF. Theorem 22 gives an upper bound.

▶ **Theorem 22.** *Let X be a mixed-stable sequence and let T be the tree that corresponds to it. Then $cost(GF, X, T) < c \cdot cost(OPT, X, T)$ for $c = \frac{5}{2}$. If X is strongly-stable, then $c = 2$.*

We defer the proof of Theorem 22 to Appendix A.2. The upper-bound in Theorem 22 is clearly not tight, since in the proof of Theorem 22 we neglected a term using the inequality $\hat{c}(GF, X) \leq \frac{2}{\alpha}\cdot\hat{c}(OPT, X) - \frac{1}{\alpha}\left(1 - \frac{n-1}{2m}\right) < \frac{2}{\alpha}\cdot\hat{c}(OPT, X)$, for a constant α. The lack of tightness is more prominent when $\hat{c}(OPT, X)$ is small, like in the sequence studied in Lemma 21 (for Theorem 4). We suspect that the lower bound in Theorem 4 is tight, and more strongly, that the F-tree pattern is the best pattern to use. This is based on studying several other recursive patterns, including those in Figure 4 and Figure 6: None was stronger, and it also seems that patterns with large costs do not "compensate" with large enough promotions.

As a closing remark to the multiplicative results, we note that by the static optimality theorem for GG [9], competitive analysis against a *static* algorithm (i.e. an algorithm that does not change its initial tree) cannot show a super-constant lower bound. Concretely, the theorem states that $cost(GF, X) \equiv cost(GG, X) = O(m + \sum_{i=1}^{n} n_i \lg \frac{m}{n_i})$ and one can



(a) (k, r)-tree pattern. **(b)** Promotion scheme. The main gain is due to the first step.

Figure 6 (a) The recursive pattern of a (k, r)-tree, T_r. The *trunk* of the tree has k nodes: the root, and a chain of $k-1$ nodes leading to an actual leaf. The rest of the leaves are $(k, r-1)$-trees. (b) The promotion scheme used later in Lemma 26, exemplified for $k = 4$ (see also Figure 5 for the degenerate case of $k = 2$). The main gain is from the first step of promoting the actual leaf to the root, and its sibling subtree (marked with $+$) one step upwards. Additional gain is achieved by promoting the left-most node of each hanging right subtree to the trunk at the expense of demoting trunk nodes. More promotions are done recursively within each subtree. The nodes marked $0, 1, 2$ are indeed consecutive, and also: $2 < x$ and $x + 1 = w < y = z - 1$.

verify that the actual constants are $5m + 6\sum_{i=1}^{n} n_i \lg \frac{m}{n_i}$. This bound can be re-written as $5m + 6m \cdot H_2(X)$ where $H_2(X) = \sum_{i=1}^{n} \frac{n_i}{m} \lg \frac{m}{n_i}$ is the base-2 entropy of the frequencies of the values in X. By [17], $cost(OPT^s, X) \geq m \cdot \frac{H_2(X)}{\lg 3}$ where OPT^s is the static optimum, and therefore $cost(GF, X) \leq (5 + 6 \lg 3) \cdot cost(OPT^s, X)$. Thus, no static argument can show a lower bound larger than ≈ 11.59.

3.4 Additive Lower Bounds for *GF*

In this section we move on to analyze the additive gap between GF and OPT. For this, we construct and analyze more elaborate patterns of recursively-defined trees, in order to get a large average promotion when optimizing the structure of the trees. The analysis is more involved since we cannot simply assume that the depth of the recurrence, r, approaches infinity. Here n is a function of r and the difference of cost can be meaningful in terms of n only if n is finite.

▶ **Definition 23.** *For $k \geq 2$, and $r \geq 0$ we define a (k, r)-tree T_r as follows. The tree is recursive of depth r (as in Definition 19), such that its trunk is composed of a root and a left-chain of length $k-1$ that starts in the right-child of the root. The left child of the deepest node of the trunk is an actual leaf, and the rest of the leaves are T_{r-1} subtrees. T_0 is a single node. See Figure 6. When k is clear from the context, we also refer to the tree as T_r.*

Observe that the tree F_r that was used to prove Theorem 4 is in fact a (k, r)-tree with $k = 2$. When we conclude the analysis, we will get the two ends of a "tradeoff" such that on the one end we have a relatively high cost ratio, and on the other a relatively high cost difference. Moreover, we will show that the higher the difference of costs on a sequence induced by (k, r)-tree, the closer the cost ratio is to 1 (comparing GF to OPT).

▶ **Lemma 24.** *The depth of a (k, r)-tree is $k \cdot r$, and its left-most node is at depth r.*

Proof. Trivial by induction: For $r = 0$, the deepest node is the root, at depth 0. For $r \geq 1$, observe that the deepest node belongs to the deepest subtree T_{r-1}, which is rooted at depth k since the path to it includes k trunk nodes. Similarly, the depth of the left-most node is increased by 1 per recursive level of the tree. ◀

▶ **Lemma 25.** *Let T_r be a (k,r)-tree. Then $|T_r| = (2+\frac{2}{k-1})k^r - (1+\frac{2}{k-1})$ where $|T_r|$ is the number of nodes in T_r. In rougher terms, $|T_r| = \Theta(k^r)$.*

Proof. Denote $n_r = |T_r|$. By definition, $n_0 = 1$ and $n_r = (k+1) + k \cdot n_{r-1}$. Hence, by Lemma 20 (with $\gamma = 0$): $n_r = \frac{k+1}{1-k}(1-k^r) + k^r = (2+\frac{2}{k-1})k^r - (1+\frac{2}{k-1})$. ◀

▶ **Lemma 26.** *Let X be any mixed-stable sequence corresponding to a (k,r)-tree T_r. Denote the average weighted promotion possible in T_r by p_r, where weighting is according to the frequency of querying each leaf. Then $p_r > k \cdot (1-\alpha^r)$ for $\alpha = 1 - \frac{1}{3^k}$. In particular, if X is a strongly-stable sequence, then $p_r = (k+1)\cdot(1-\alpha^r) + \delta$ for $\alpha = 1 - \frac{1}{2^k}$ and $0 \le \delta < \alpha^r$.*

Proof. We can promote by k every explicit leaf in every $T_{r'}$ for all recursive levels $1 \le r' \le r$, from its location to the root of $T_{r'}$. Only nodes that are T_0 leaves do not contribute an explicit promotion of at least k, therefore $p_r > k \cdot (1-f)$ where f is the sum of query-frequencies of all T_0 leaves (the inequality is strict due to unaccounted subtree promotions). To conclude, we argue that $f \le (1-\frac{1}{3^k})^r$. The frequency of accessing the explicit leaf of T_r is at least $\frac{1}{3^k}$ by Lemma 15, hence with frequency of at most $1-\frac{1}{3^k}$ we query a value in some T_{r-1} subtree. Similarly, within the chosen subtree there is again a relative frequency of at most $1-\frac{1}{3^k}$ to query within some T_{r-2} subtree. Overall, since there are r levels of recursion, we conclude that $f \le (1-\frac{1}{3^k})^r$.

Proving the second part of the claim required a more careful analysis. We define the following method of promotion, depicted in Figure 6. In the (k,r)-tree we promote the (only) explicit leaf to the root, and promote its sibling subtree by 1. Then we apply similar promotions recursively within every $(k,r-1)$-subtree. Finally, we promote the left-most node within each $(k,r-1)$-subtree that hangs as a right-subtree from the trunk to the parent of this subtree. Denote the total average (weighted) promotion by p_r. Note that it does not matter if we promote the left-most nodes of the right subtrees before or after the recursive promotions, because the total order on the items guarantees that there is only one value that can be put instead of every demoted trunk node, and the recursive promotions within a specific subtree do not change the depth of its leftmost leaf.

The promotion of the explicit leaf of T_r saves a cost of k weighted by a factor (query frequency) of $\frac{1}{2^k}$. The promotion of the sibling subtree saves 1 weighted by a factor of $\frac{1}{2^k}$. The recursive promotions are p_{r-1} weighted by $\sum_{i=1}^{k}\frac{1}{2^i}$ (for all the k subtrees), and finally the last promotions are technically negligible (as seen in the analysis below), but for the sake of completeness we consider them in the analysis as well: promoting the left-most node from each subtree saves $(r-1)+1 = r$ since the leaf that we promote last is at depth $r-1$ within the recursive subtree, and this promotion is weighted by $\frac{1}{2^r}\cdot\sum_{i=2}^{k-1}\frac{1}{2^i}$ (factor of $\frac{1}{2^r}$ follows from Lemma 24). We get that: $p_r = \frac{k+1}{2^k} + p_{r-1}\cdot\left(1-\frac{1}{2^k}\right) + \frac{r}{2^r}\cdot\frac{1}{2}\left(1-\frac{1}{2^{k-2}}\right)$. Then by Lemma 20, with $\alpha = 1-\frac{1}{2^k}$ and $\gamma = \frac{1}{2}(1-\frac{1}{2^{k-2}})$, we get:

$$p_r = (k+1)\cdot(1-\alpha^r) + \delta \quad , \quad \delta \equiv \alpha^r \cdot p_0 + \frac{2\alpha\gamma}{(2\alpha-1)^2}\cdot\left(\alpha^r - \frac{1}{2^r}\right) - \frac{\gamma}{(2\alpha-1)}\cdot\frac{r}{2^r}$$

It remains to show that $0 \le \delta < \alpha^r$. It is simple to see that $\delta = 0$ for $k = 2$, because then $\gamma = 0$ and $p_0 = 0$ is the average weighted promotion in a tree with a single node. For $k \ge 3$, by the definition of α and γ we have that $\frac{2\alpha\gamma}{(2\alpha-1)^2} = \frac{(2^k-1)(2^k-4)}{(2^k-2)^2} = 1 - \frac{1}{2^k-4+\frac{4}{2^k}} \in (\frac{3}{4}, 1)$ and $\frac{\gamma}{2\alpha-1} = \frac{1}{2} - \frac{1}{2^k-2} \in [\frac{1}{3}, \frac{1}{2})$. Substituting these bounds and $p_0 = 0$ into the formula for δ gives $\delta < \alpha^r$. Moreover, δ is positive since $\delta > \frac{3}{4}(\alpha^r - \frac{1}{2^r}) - \frac{1}{2}\cdot\frac{r}{2^r} = \frac{3}{4}(\alpha-\frac{1}{2})\cdot\sum_{i=0}^{r-1}\alpha^i\cdot\left(\frac{1}{2}\right)^{r-1-i} - \frac{r}{2^{r+1}} = \frac{3(\alpha-\frac{1}{2})}{2^{r+1}}\cdot\sum_{i=0}^{r-1}(2\alpha)^i - \frac{r}{2^{r+1}} > \frac{3(\alpha-\frac{1}{2})}{2^{r+1}}\cdot r - \frac{r}{2^{r+1}} = (3(\frac{1}{2}-\frac{1}{2^k})-1)\cdot\frac{r}{2^{r+1}} > 0$ for $k \ge 3$.

Note that indeed the gain from promoting the left-most node of each subtree is negligible, since the effect is merely having $\gamma \neq 0$, which only contributes $0 \leq \delta < \alpha^r < 1$. ◀

▶ **Corollary 27.** *The average cost-per-query of GF on a strongly-stable sequence induced by a (k,r)-tree is larger than the optimal cost by at least $(k+1)\cdot\left(1-\left(1-\frac{1}{2^k}\right)^r\right)$. On any mixed-stable sequence, the difference is at least $k\cdot\left(1-\left(1-\frac{1}{3^k}\right)^r\right)$.*

We are ready to prove Theorem 5.

▶ **Theorem 5.** *For every $n \geq 2$ there exist sequences $X \in [n]^m$ such that $cost(GF,X) = cost(OPT,X) + \Omega(m \cdot \lg\lg n)$. Among these sequences, there exists a sequence whose length is $m = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$. (There exist other longer sequences too.)*

Proof. Let X be the strongly-stable sequence induced by a (k,r)-tree T_r, and for simplicity assume that the initial tree is T_r.[6] By Lemma 25, $n = (2+\frac{2}{k-1})k^r - (1+\frac{2}{k-1})$ therefore $\lg\lg n = \lg r + \lg\lg k + O(1)$.[7] By Corollary 27, $\hat{c}(GF,X) - \hat{c}(OPT,X) \geq \Delta \equiv (k+1)\cdot(1-(1-\frac{1}{2^k})^r)$. By choosing $r = 2^k$ we get that $\Delta = (k+1)\cdot(1-(1-\frac{1}{2^k})^{2^k}) \approx (1-\frac{1}{e})\cdot(k+1)$.[8] We also get that $\lg\lg n = k + \lg\lg k + O(1)$, therefore $\Delta \approx (1-\frac{1}{e})\lg\lg n$ and we conclude that $\hat{c}(GF,X) - \hat{c}(OPT,X) \geq \Omega(\lg\lg n)$.

By Lemma 13 the length of the atomic strongly-stable sequence of T_r is $m = 2^{d(T_r)}$, hence $m = 2^{rk}$ by Lemma 24. By Lemma 25, $\frac{n+(1+2/(k-1))}{2+2/(k-1)} = k^r = 2^{r\lg k}$. Together we get that $m = 2^{rk} = 2^{(r\lg k)\cdot(k/\lg k)} = \left(\frac{n+(1+2/(k-1))}{2+2/(k-1)}\right)^{(k/\lg k)} = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$. ◀

▶ Remark 28. In the proof of Theorem 5, the sequence X does not have to be strongly-stable, and any mixed-stable sequence X induced by a (k,r)-tree T_r works as well. Indeed, Corollary 27 guarantees that $\hat{c}(GF,X) - \hat{c}(OPT,X) \geq \Delta$ for $\Delta = k\cdot\left(1-\left(1-\frac{1}{3^k}\right)^r\right)$, and then by choosing $r = 3^k$ we get that $\Delta = \Theta(k)$, and $k = \Theta(\lg\lg n)$, and $m = 2^{\Theta(rk)} = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$.

▶ Remark 29. The choice of $r = 2^k$ in the proof of Theorem 5 maximizes our lower bound on the additive gap $cost(GF,X) - cost(OPT,X)$ (up to constants) for our (k,r)-trees. Indeed, revisiting the proof, we have that Δ and n are both functions of k and r, and we need to choose r and k to maximize Δ as a function of n. Note that $\Delta = O(k)$ regardless of r, and $\lg\lg(n) = \lg r + \lg\lg k + O(1)$. To simplify and eliminate a parameter we define $r = 2^k \cdot f(k)$ for some monotone function f. Now we get simplified relations: $\Delta = (k+1)\cdot(1-(1-\frac{1}{2^k})^{2^k f(k)}) \approx (k+1)\cdot(1-e^{-f(k)})$ and $\lg\lg n = k + \lg f(k) + \lg\lg k + O(1)$. Consider the following two cases.

- If $f(k) = \Omega(1)$: then $\exists c \in \mathbb{R}$ such that $\forall k \geq 1 : \lg f(k) \geq c$, and therefore $\lg\lg n = \Omega(k)$, written differently $k = O(\lg\lg n)$, which yields $\Delta = O(k) = O(\lg\lg n)$.
- If $f(k) = o(1)$: Being $o(1)$ means that $\lim_{k\to\infty} f(k) = 0$, so for sufficiently large values of k we can use the approximation $e^x \approx 1 + x$ (that holds for small x) to get: $\Delta \approx (k+1)\cdot f(k) = \frac{k+1}{1/f(k)}$. If $\frac{1}{f(k)}$ grows faster than $(k+1)$, we get $\Delta = O(1)$ which does not even grow with n. Therefore $\frac{1}{f(k)}$ is increasing, but at a sub-linear rate. Recall that $\lg\lg n = k - \lg\frac{1}{f(k)} + \lg k + O(1)$. Since $\frac{1}{f(k)}$ is sub-linear, we get that $k = \Theta(\lg\lg n)$, which yields $\Delta = O(k) = O(\lg\lg n)$.

---

[6] We remove this assumption in Remark 34.

[7] $k \geq 2 \Rightarrow k^r \leq n < 4k^r \Rightarrow \lg n = r\lg k + c$ for $c \in [0,2)$, and so $\lg\lg n = \lg r + \lg\lg k + O(1)$.

[8] The approximation is off by less than 10% for $k \geq 2$. (60% and 20% for $k = 0, 1$ respectively.)

▶ **Corollary 30.** *GF is not* $(1, O(m))$*-competitive. If the multiplicative term is* 1*, then the additive term is at least* $\Omega(m \cdot \lg\lg n)$*.*

We have yet to analyze the cost of OPT on a strongly-stable sequence X corresponding to a (k,r)-tree that produces the gap of $\Omega(m \cdot \lg\lg n)$ in Theorem 5. Allegedly, if the cost is cheap, say linear, we would get a large competitive ratio as well. However, by Theorem 22 we expect a competitive ratio of at most 2, and therefore we can conclude without further analysis, that $cost(OPT, X) = \Omega(m \cdot \lg\lg n)$. In fact, we prove that $cost(OPT, X) = \Theta(m \cdot \frac{\lg n}{\lg\lg\lg n})$. It follows that the competitive ratio deteriorates when the additive gap increases.

▶ **Lemma 31.** *Define the constants* $\alpha \equiv 1 - \frac{1}{2^k}$ *and* $\beta \equiv \sum_{j=1}^{k} \frac{j}{2^j} + \frac{k+1}{2^k}$*. Let* X *be a strongly-stable sequence induced by a* (k,r)*-tree. Then* $\hat{c}(GF, X) = 2^k \cdot \beta \cdot (1 - \alpha^r) + \alpha^r$*. In asymptotic terms:* $\hat{c}(GF, X) = \Theta(2^k \cdot (1 - \alpha^r))$*.*

Proof. We write a recurrence for the average cost, c_r, of GF on the strongly-stable sequence induced by T_r. We have $c_0 = 1$, and

$$c_{r+1} = \frac{1+k}{2^k} + \sum_{j=1}^{k} \frac{j + c_r}{2^j} = \Big(1 - \frac{1}{2^k}\Big)c_r + \sum_{j=1}^{k} \frac{j}{2^j} + \frac{1+k}{2^k} \equiv \alpha \cdot c_r + \beta$$

($\frac{1+k}{2^k}$ is due to the actual leaf, and the summation is the contribution of all the T_r subtrees.) By Lemma 20 (with $\gamma = 0$), $c_r = \frac{\beta}{1-\alpha}(1 - \alpha^r) + \alpha^r \cdot c_0 = 2^k \cdot \beta(1 - \alpha^r) + \alpha^r$. Since $\alpha = 1 - \frac{1}{2^k} \in [\frac{3}{4}, 1)$ clearly $\alpha^r < 1$. Furthermore, $\beta = \Theta(1)$. To see this note that β only depends on k. Denote $\beta = \beta(k)$ and observe that: $\beta(k+1) - \beta(k) = \big(\frac{k+1}{2^{k+1}} + \frac{k+2}{2^{k+1}}\big) - \frac{k+1}{2^k} = \frac{1}{2^{k+1}}$. Therefore, $\beta(k) = \beta(2) + \sum_{i=3}^{k} \Big(\beta(i) - \beta(i-1)\Big) = \Big(\frac{1}{2} + \frac{2}{4} + \frac{3}{4}\Big) + \sum_{i=3}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$, and $\beta(k) \in [\frac{7}{4}, 2) \Rightarrow \beta = \Theta(1)$. Because $2^k \cdot (1 - \alpha^r) \geq 2^k \cdot (1 - \alpha) = 1 > \alpha^r$, we conclude that $c_r = \Theta(2^k \cdot (1 - \alpha^r))$. ◀

▶ **Lemma 32.** *Let* X *be a sequence from the family of sequences in Theorem 5, then* $cost(OPT, X) = \Theta(m \cdot \frac{\lg n}{\lg\lg\lg n})$*.*

Proof. Let X be a strongly-stable sequence induced by querying a (k,r)-tree. We know that $\frac{1}{2}\hat{c}(GF, X) < \hat{c}(OPT, X) \leq \hat{c}(GF, X)$ where the lower-bound is by Theorem 22. Therefore, $\hat{c}(OPT, X) = \Theta(2^k \cdot (1 - \alpha^r))$ by Lemma 31. By Lemma 25, $\lg n = r \cdot \lg k + O(1)$, or $r = \frac{\lg n - O(1)}{\lg k}$. When we substitute $r = 2^k$ as in the proof of Theorem 5, we get that $(1 - \alpha^r) = \Theta(1)$ and $2^k = r = \frac{\lg n - O(1)}{\lg k} = \Theta(\frac{\lg n}{\lg\lg\lg n})$. Therefore, $cost(OPT, X) = \Theta(m \cdot \frac{\lg n}{\lg\lg\lg n})$. ◀

As a concluding remark, we recall that the F_r-tree is a (k,r)-tree for $k = 2$. If we substitute $k = 2$ in the formula of Lemma 31 we get that $\alpha = \frac{3}{4}$, $\beta = \frac{7}{4}$, and $\hat{c}(GF, X) = 7 \cdot (1 - (3/4)^r) + (3/4)^r$. By Lemma 26, the average promotion is $3 \cdot (1 - (3/4)^r)$ (for $k = 2$, we have $\delta = 0$). These values are the strongly-stable analogues of Lemma 21, and can be used to show a weaker lower bound of $\frac{7}{4}$, on the competitive ratio of GF.

4 Conclusions and Open Questions

In this paper we gave improved lower bounds on the competitiveness of the Greedy Future (GF) algorithm for serving a sequence of queries by a dynamic binary search tree (BST). In contrast to many of the previous results on GF that are obtained using the geometric-view by studying the equivalent Geometric Greedy (GG) algorithm, we used the standard "tree-view" and the treap-based definition of GF. We showed that the competitive ratio of GF is at

least 2, and that there are sequences $X \in [n]^m$ for which the cost difference (additive gap) between GF and OPT is $\Omega(m \cdot \lg \lg n)$. These lower bounds enabled us to show that if GF is approximately-monotone (Definition 8) with some constant c then $c \geq 2$. Also, the lower bounds show that the cost of GF on a sequence compared to its cost on its reverse, may differ by a factor as close as we like to 2. In contrast, the cost of OPT on a sequence compared to its reverse may differ by at most n.

Our results give new insights on the "tradeoff" between the additive term and the multiplicative term in the competitiveness of GF, showing that the multiplicative term is typically larger when the total cost of the algorithm on the sequence is smaller. Indeed, our best multiplicative term is achieved for a sequence whose average cost per query is 6. This tradeoff is not surprising since a fixed difference implies a larger ratio when the quantities are small. It may be interesting to figure out if this tradeoff hints of some underlying property of GF, or is just an artifact of our technique that requires high costs on average per query in order to increase the additive gap between GF and OPT.

Clearly, these improved lower bounds still don't settle the deeper question of whether GF (and GG) is dynamically-optimal. Our techniques focused on a smaller family of sequences which we named *mixed-stable sequences*, whereas "most" sequences are not stable. While it is possible that an improved lower bound (larger than 2) can be found by a more clever pattern of mixed-stable sequences, it seems more likely to be found by analyzing sequences for which the tree maintained by GF is not static. In addition, we note that GF was not investigated too deeply directly, as most of the work has been done in the geometric view with respect to its counterpart GG. Therefore, studying other problems in tree-view may give complementing insights. One such problem is the deque conjecture, which has been partially settled for GG, in the case when deletions are only allowed on the minimum item [2].

References

1. Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the Strong Wilber 1 Bound for Binary Search Trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 33:1–33:21, 2020.
2. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Greedy is an almost optimal deque. In *14th International Conference on Algorithms and Data Structures (WADS)*, pages 152–165, 2015.
3. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 410–423, 2015.
4. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *arXiv*, abs/1603.04892, 2016. `arXiv:1603.04892`.
5. Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Nidia Obscura Acosta, Akash Pareek, and Sorrachai Yingchareonthawornchai. Improved pattern-avoidance bounds for greedy BSTs via matrix decomposition. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2023.
6. Parinya Chalermsook and Wanchote Po Jiamjitrak. New binary search tree bounds via geometric inversions. In *28th Annual European Symposium on Algorithms (ESA)*, pages 28:1–28:16, 2020.
7. Erik D. Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505, 2009.
8. Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality – almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

9 Kyle Fox. Upper bounds for maximally greedy binary search trees. In *12th International Conference on Algorithms and Data Structures (WADS)*, pages 411–422, 2011.

10 Michael L. Fredman. Two applications of a probabilistic search technique: Sorting X+Y and building balanced search trees. In *7th Annual ACM Symposium on Theory of Computing (STOC)*, pages 240–244, 1975.

11 George F. Georgakopoulos. Chain-splay trees, or, how to achieve and prove loglogN-competitiveness by splaying. *Information Processing Letters*, 106(1):37–43, 2008.

12 Dion Harmon. *New Bounds on Optimal Binary Search Trees.* PhD thesis, Massachusetts Institute of Technology, 2006.

13 Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1989.

14 László Kozma. *Binary search trees, rectangles and patterns.* PhD thesis, Saarland University, 2016.

15 Caleb Levy and Robert Tarjan. *New Paths from Splay to Dynamic Optimality.* PhD thesis, Princeton, 2019.

16 Joan M. Lucas. Canonical forms for competitive binary search tree algorithms. In *Tech. Rep. DCS-TR-250.* Rutgers University, 1988.

17 Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5:287–295, 1975.

18 J. Ian Munro. On the competitiveness of linear search. In *8th Annual European Symposium on Algorithms (ESA)*, pages 338–345. Springer-Verlag, 2000.

19 Hauke Reddmann. On the geometric equivalent of instance optimal binary search tree algorithms. Master's thesis, Universität Hamburg, 2021.

20 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of ACM*, 32(3):652–686, 1985.

21 Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. O(log log n)-competitive dynamic binary search trees. In *17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 374–383, 2006.

22 Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on Computing*, 18(1):56–67, 1989.

A Appendix: Deferred Proofs and Discussions

A.1 Enforcing a Stable Tree for *GF*

We describe how to restructure any initial tree, to a desired tree, when GF is considered. The initial tree cannot simply be re-organized since GF updates the tree in a specific way following each query. Let $P \circ X$ denote the concatenation of the sequences P and X. Note that even if P enforces a desired tree when served alone, serving $P \circ X$ may give a different tree following P when X starts. The reason for this is that GF restructures the tree while serving P according to future queries, therefore the existence of X may affect its decisions while serving P. Nevertheless, we present a simple technique to enforce a tree for GF.

▶ **Theorem 33.** *For any tree T there is a sequence $S(T)$ such that: (1) $|S(T)| = O(n \cdot d(T))$, and more precisely, $|S(T)| \leq \min\left(3n(d(T) - \lfloor \lg n \rfloor + 1), \frac{3}{2}n(n-1)\right)$, and, (2) for any suffix of queries Y, when GF serves $S(T) \circ Y$, its tree when is it done with the last query of $S(T)$ is T. We say that $S(T)$* enforces *T.*

Proof. We enforce the structure of T bottom-up, by first ensuring the position of the leaves, and continuing recursively upwards towards the root. Define $T^{[0]} \equiv T$ and $T^{[i+1]}$ is the tree $T^{[i]}$ stripped of all of its leaves, until the final tree $T^{[h]}$ contains only the root. Observe that $h = d(T)$ (the depth of T). For each tree $T^{[i]}$ we define R_i to be the set of non-leaf nodes in $T^{[i]}$. Note that a non-leaf node may be binary or unary.

We construct $S(T)$ in steps. In step $i \geq 0$ we query R_i in two monotonic phases, where the first phase queries every item twice, and the second phase queries every item once. Denote the queries of this step by S_i. For example, if $R_0 = \{1, 3, 5\}$ then we query in the first step: $S_0 = [1, 1, 3, 3, 5, 5, 1, 3, 5]$. Note that $S_h = \emptyset$ since $R_h = \emptyset$ by definition. The resulting sequence $S(T)$ is the concatenation of the queries in all the (non-empty) steps, that is, $S(T) = S_0 \circ S_1 \circ \ldots \circ S_h$ ($\circ$ for concatenation).

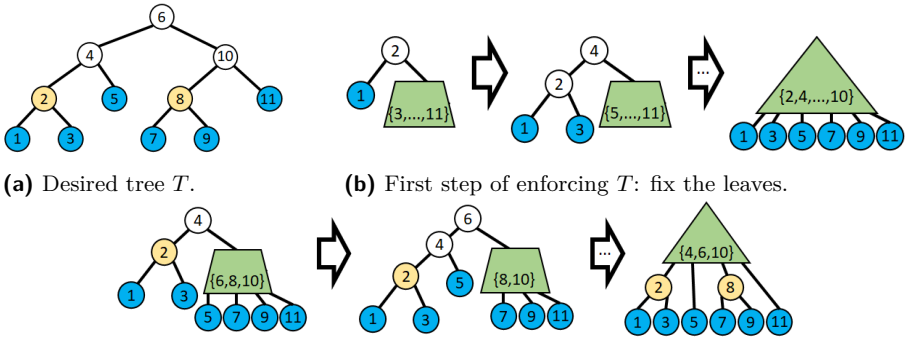
First we analyze $|S(T)|$. Observe that we strip leaves between steps, $|R_{i+1}| \leq |R_i| - 1$. Initially $|R_1| \leq n - 1$ hence $|R_i| \leq n - i$. Also, $h < n$. Therefore we get that: $|S(T)| = \sum_{i=1}^{h} |S_i| = \sum_{i=1}^{h} 3|R_i| \leq 3\sum_{i=1}^{n}(n-i) = \frac{3}{2}n(n-1)$. To get the other bound, observe that $|R_{h-i}| \leq 2^i - 1$, which is meaningful for the last few steps, i.e., for $i \leq \lfloor \lg n \rfloor$. With that in mind: $|S(T)| = 3\sum_{i=1}^{h-\lfloor \lg n \rfloor} |R_i| + 3\sum_{i=h-\lfloor \lg n \rfloor + 1}^{h} |R_i| < 3\sum_{i=1}^{h-\lfloor \lg n \rfloor} n + 3\sum_{j=0}^{\lfloor \lg n \rfloor - 1} 2^j < 3n(h - \lfloor \lg n \rfloor) + 3n$. In conclusion, $|S(T)| < 3n(d(T) - \lfloor \lg n \rfloor + 1)$.

Next we prove that the tree of GF is exactly T when it finishes serving $S(T)$ regardless of any suffix of queries. The proof is by induction on $d(T) = h$. The base case is for $h = 0$. In this case $S(T) = \emptyset$ (because $S_0 = \emptyset$), while also $T^{[h]} = T^{[0]} = T$ contains only the root. Querying nothing is not a problem since there is a unique tree with a single node, and we are done. Now, assume that the claim holds for $h = k$. Consider a tree T of depth $k + 1$. First we show that when GF finishes processing S_0, all the leaves of T are leaves of the tree of GF, and will never be touched (accessed or otherwise), therefore they remain leaves until GF finishes processing $S(T)$. See Figure 7 for a visualization. Indeed, when we query a value u twice in a row, GF brings u to the root when re-ordering the tree after the first query of the couple. Note that since we query R_0 monotonically, by the end of the first phase of S_0 (the queries in pairs), GF has a tree whose left-spine is exactly the items of R_0. Indeed, each item, in turn, is brought to the root and demotes the previous root to the left. Moreover, no value of $T^{[0]} \setminus R_0$ is part of this spine, because of the second phase of queries (monotonously querying the values of R_0). To see this, note that if on the first phase we touch $v \notin R_0$ when r is the root and the pair of queries is to u, such that $r < v < u$, and u is the successor of r in R_0, then the next access to r is closer in the future than that of v, so r will indeed be placed as the left-child of the new root u (and v will be the right child of r). If $r < u < v$ then v does not interfere with r being the left child of u.

Now that we know that all the leaves of $T = T^{[0]}$ are fixed as leaves of the tree of GF when it finishes processing S_0, we can conclude by induction: $S' \equiv S_1 \circ \ldots \circ S_{k+1}$ is exactly the sequence that enforces $T^{[1]}$, and by our inductive assumption, $T^{[1]}$ is enforced correctly. Since the leaves of $T^{[0]}$ are not touched at all during S', we conclude that they must remain hanging off $T^{[1]}$ as "subtrees". The location of each leaf is uniquely determined, and thus we conclude that $S(T)$ indeed enforces $T = T^{[0]}$. ◀

Adding a prefix to our sequence may affect the competitive ratio. However, once we fixed the stable tree, we can repeat the corresponding stable sequence to "amplify" the original competitive ratio making the effect of the prefix negligible. One difficulty raised by repetitions is when we care about the length of the sequence in our claim. This is the case in Theorem 5 where we claim the existence of a sequence of length $n^{\Theta(\frac{\lg \lg n}{\lg \lg \lg n})}$. In the proof of this theorem we assumed for simplicity that we can choose the initial tree. The following remark shows that indeed we can start with an arbitrary initial tree without weakening the theorem.

▶ Remark 34. Let X be an atomic mixed-stable sequence used to prove Theorem 5. Consider the sequence $Z = S \circ X^n$, where S is the prefix (guaranteed by Theorem 33) that is enforcing the desired "initial" tree T that corresponds to X, X^n are n repetitions of X,



(a) Desired tree T.

(b) First step of enforcing T: fix the leaves.

(c) Second step of enforcing T: fix the leaves of $T^{[1]}$.

Figure 7 Example of enforcing a tree as detailed in Theorem 33. The desired tree T is shown in (a). Blue nodes are the leaves of $T^{[0]} = T$ and golden nodes are the leaves of $T^{[1]}$ (leaves of $T^{[2]}$ and $T^{[3]}$ are not colored). The first step of queries is $S_0 = [2, 2, 4, 4, 6, 6, 8, 8, 10, 10, 2, 4, 6, 8, 10]$; (b) shows the states after the first query of 2, the first query of 4, and by the end of S_0. Following the first step, it remains to enforce the remainder of the tree (the inductive step). Concretely, the second step of queries is $S_1 = [4, 4, 6, 6, 10, 10, 4, 6, 10]$; (c) shows the states after the first query of 4, the first query of 6, and by the end of S_1. The green trapezoids and triangles abstract away the structure of some subtrees. The last non-empty step is $S_2 = [6, 6, 6]$ (not shown), after which we get T.

and $\circ$ represents concatenation. There are no unary nodes in T and X queries every leaf at least once so $\frac{n}{2} < \frac{n+1}{2} \leq |X|$. Together with Theorem 33 we get that $n|X| \leq |Z| < n(|X| + \frac{3n}{2}) < 4n|X|$, therefore we have: $|Z| = \Theta(n|X|) = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$ (the second equality is by Theorem 5). Since after processing S the tree of GF is fixed: $cost(GF, Z, T_0) - cost(OPT, Z, T_0) \geq n \cdot (cost(GF, X, T) - cost(OPT, X, T))$. By the proof of Theorem 5 and Remark 28: $cost(GF, X, T) - cost(OPT, X, T) = \Omega(|X| \cdot \lg\lg n)$, and putting everything together we get that: $cost(GF, Z, T_0) - cost(OPT, Z, T_0) = \Omega(|Z| \cdot \lg\lg n)$. Note that if $|X| = \Omega(n^2)$ then it suffices to define $Z = S \circ X$ without repetitions and we get that $|Z| = \Theta(|X|)$, and the rest of the arguments remain the same. There may be additional cases where repetitions are not required, e.g., when the depth of T is $\lg n + O(1)$ (in this case, $|S| = O(n)$).

A.2 Omitted Proofs

In this subsection we restate and prove Lemmas and Theorems that were omitted from the main text. For convenience, we restate the claims in their original numbering.

The proof of Theorem 22 makes use of Wilber's first bound [22]. We use the original presentation of this bound which is a bit tighter than later simplified versions such as [14].

▶ **Definition 35** (Wilber's First Bound [22]). *Let X be a sequence of queries, and let T be a static reference tree such that every query of X is in a leaf of T. An alternation at an inner node u of T is defined to be two queries closest in time such that one accesses either the left or right subtree of u and the other accesses the other subtree of u. Define $ALT(u)$ to be the number of alternations at node u. Then: $cost(OPT, X) \geq m + \frac{1}{2}\sum_{inner\ u \in T} ALT(u)$.*

▶ **Theorem 22.** *Let X be a mixed-stable sequence and let T be the tree that corresponds to it. Then $cost(GF, X, T) < c \cdot cost(OPT, X, T)$ for $c = \frac{5}{2}$. If X is strongly-stable, then $c = 2$.*

Proof. We use the tree that corresponds to the mixed-stable sequence as the reference tree for Wilber's first bound. Arithmetic manipulations will yield an expression that we can tie to the cost of GF, according to the claim.

Let X be a mixed-stable sequence, with a corresponding tree T. Let S be the set of values that are in the leaves of T, and let U be the set of inner nodes, $|U| = \frac{n-1}{2}$. We also denote by $A(i)$ the set of proper ancestors of i. By the definition of the cost of a static tree, we know that $\hat{c}(GF, X) = \sum_{i \in S} (d(i) + 1) \cdot f(i)$ where $d(i)$ is the depth of i and $f(i)$ is the frequency of accessing i. We extend $f(u)$ to refer to the frequency of visiting any node u. Note that $f(u) = \sum_{i \in S \wedge u \in A(i)} f(i)$ and that $\sum_{i \in S} f(i) = 1$.

Now consider Wilber's bound for X, with T as the reference tree. We can use T as the reference tree since X only accesses leaves of T, by definition. We also denote $\alpha_u \equiv \frac{ALT(u)+1}{f(u) \cdot m}$ ($ALT(u)$ is defined in Defintion 35, and note that $0 \leq ALT(u) \leq f(u) \cdot m - 1$). We have $\alpha_u \in (0, 1]$, where $\alpha_u = 1$ corresponds to fully alternating accesses to the subtree rooted at u. The lower bound is $cost(OPT, X) \geq m + \frac{1}{2} \sum_{u \in U} (ALT(u) + 1) - \frac{|U|}{2} = \frac{m}{2} + \frac{m}{2}(1 + \sum_{u \in U} \alpha_u \cdot f(u)) - \frac{n-1}{4} = \left(\frac{m}{2} - \frac{n-1}{4}\right) + \frac{m}{2} \sum_{i \in S} (1 + \sum_{u \in A(i)} \alpha_u) f(i)$. Let $\alpha \leq \min_{u \in U} \alpha_u$, we get that $\hat{c}(OPT, X) \geq \left(\frac{1}{2} - \frac{n-1}{4m}\right) + \frac{\alpha}{2} \sum_{i \in S} (d(i) + 1) \cdot f(i) = \frac{\alpha}{2} \hat{c}(GF, X) + \left(\frac{1}{2} - \frac{n-1}{4m}\right)$ where the equality holds since GF maintains a static tree. Thus $\hat{c}(GF, X) \leq \frac{2}{\alpha} \cdot \hat{c}(OPT, X) - \frac{1}{\alpha}\left(1 - \frac{n-1}{2m}\right) < \frac{2}{\alpha} \cdot \hat{c}(OPT, X)$.

In order to choose a suitable α, recall that a strongly-stable node u has a coefficient of $\alpha_u = 1$, which means that for strongly-stable sequences, in which all inner nodes are stable, we can pick $\alpha = 1$ and conclude that $\hat{c}(GF, X) < 2 \cdot \hat{c}(OPT, X)$. If u is a weakly-stable node, then its coefficient is $\alpha_u = \frac{2}{3}$. So for a mixed-stable sequence we can naively pick $\alpha = \frac{2}{3}$, resulting in $\hat{c}(GF, X) < 3 \cdot \hat{c}(OPT, X)$.

In order to improve from 3 to $\frac{5}{2}$, we observe that by definition, every weakly-stable node has a strongly-stable child. Let u be a weakly-stable node and let w be its (strongly-stable) favored-child (recall Definition 11). Since $ALT(u) = ALT(w)$ (by definition of the access pattern in u), we can present Wilber's bound differently, summing $(ALT(u)+1) \cdot (1+\beta) + (ALT(w)+1) \cdot (1-\beta)$ instead of $(ALT(u)+1) + (ALT(w)+1)$. We get modified coefficients $\alpha'_u = \frac{(ALT(u)+1) \cdot (1+\beta)}{m \cdot f(u)} = \alpha_u \cdot (1 + \beta) = \frac{2(1+\beta)}{3}$ and similarly $\alpha'_w = \alpha_w(1 - \beta) = (1 - \beta)$. Choosing $\beta = \frac{1}{5}$ balances the coefficients: $\alpha'_u = \alpha'_w = \frac{4}{5}$. Now we can choose $\alpha = \frac{4}{5}$, and get $\hat{c}(GF, X) < \frac{5}{2} \cdot \hat{c}(OPT, X)$ for mixed-stable sequences. ◀

▶ **Theorem 6.** *For any $\epsilon > 0$ there exists a sequence X with a subsequence (not necessarily consecutive) $X' \subseteq X$ such that $cost(GF, X') \geq (2 - \epsilon) \cdot cost(GF, X)$.*

Proof. Denote the initial tree by T_0. Let Z be the weakly-stable sequence used for proving Theorem 4. Let T_P be the tree that corresponds to Z and T_Q the optimized tree, in which the leaves are promoted as in Lemma 21. Let P and Q be the sequences that enforce T_P and T_Q by Theorem 33, respectively. Note that ϵ determines Z, P and Q since it tells us how close to a ratio of 2 we need to get.

Revisit Figure 5 to see the (recursive) structures of T_P (on the left) and T_Q (on the right, post-promotions). Observe that T_P remains static when GF serves Z with it, by definition. Moreover, T_Q remains static when GF serves Z with it. Indeed, let r be the root of T_Q. Z queries the item in r every third access and the other accesses are alternating between its left and right subtrees, hence r remains the root of T_Q. The rest of T_Q remains static recursively.

Define $X = P \circ Q \circ Z^k$ for a large k, and $X' = P \circ Z^k \subset X$ ($\circ$ for concatenation). Since GF does not change T_P and T_Q while serving Z we get that $\frac{cost(GF,X')}{cost(GF,X)} = \frac{cost(GF,P,T_0) + k \cdot cost(GF,Z,T_P)}{cost(GF,P \circ Q,T_0) + k \cdot cost(GF,Z,T_Q)}$. This ratio approaches $\frac{cost(GF,Z,T_P)}{cost(GF,Z,T_Q)}$ for large enough k, and

since T_p and T_Q are exactly the trees used in the proof of Lemma 21, we conclude that we can make the resulting ratio as close to 2 as we like (choosing Z, P, Q according to the desired ϵ). ◀

▶ **Theorem 7.** *Let S be a sequence, we define $rev(S)$ to be the sequence S in reverse. For any $\epsilon > 0$ there exists a sequence X such that $cost(GF, rev(X)) \geq (2 - \epsilon) \cdot cost(GF, X)$.*

Proof. The proof is similar to that of Theorem 6, and we define Z, T_0, P, T_P, Q and T_Q the same way. Here we define $X = Q \circ (rev(Z))^{k+1} \circ rev(P)$, for a large k.

We claim that T_Q remains static when GF serves $rev(Z)$ over it, rather than Z, by the same argument as in the proof of Theorem 6, because the interleaving pattern in the root is preserved under reversal. Moreover, $cost(GF, rev(Z), T_Q) = cost(GF, Z, T_Q)$ because the cost on a static tree depends only on the access frequencies. Putting everything together, we get: $\frac{cost(GF,rev(X))}{cost(GF,X)} = \frac{cost(GF,P,T_0)+k\cdot cost(GF,Z,T_P)+cost(GF,Z\circ rev(Q),T_P)}{cost(GF,Q,T_0)+k\cdot cost(GF,rev(Z),T_Q)+cost(GF,rev(Z)\circ rev(P),T_Q)}$. Note that the suffix contains one repetition of Z so that the rest of it ($rev(P)$ or $rev(Q)$) does not affect the restructuring decisions of GF during the earlier repetitions of Z. The limit of this ratio for large k is $\frac{cost(GF,Z,T_P)}{cost(GF,Z,T_Q)}$. We finish the argument as in the proof of Theorem 6. ◀

The Complexity of Translationally Invariant Problems Beyond Ground State Energies

James D. Watson
University College London, UK

Johannes Bausch
University of Cambridge, UK

Sevag Gharibian
Universität Paderborn, Germany

Abstract

The physically motivated quantum generalisation of k-SAT, the k-Local Hamiltonian (k-LH) problem, is well-known to be QMA-complete ("quantum NP"-complete). What is surprising, however, is that while the former is easy on 1D Boolean formulae, the latter remains hard on 1D local Hamiltonians, *even if all constraints are identical* [Gottesman, Irani, FOCS 2009]. Such "translation-invariant" systems are much closer in structure to what one might see in Nature. Moving beyond k-LH, what is often more physically interesting is the computation of *properties of the ground space* (i.e. "solution space") itself. In this work, we focus on two such recent problems: Simulating local measurements on the ground space (APX-SIM, analogous to computing properties of optimal solutions to MAX-SAT formulae) [Ambainis, CCC 2014], and deciding if the low energy space has an energy barrier (GSCON, analogous to classical reconfiguration problems) [Gharibian, Sikora, ICALP 2015]. These problems are known to be $\mathrm{P}^{\mathrm{QMA}[\log]}$- and QCMA-complete, respectively, in the general case. Yet, to date, it is not known whether they remain hard in such simple 1D translationally invariant systems.

In this work, we show that the 1D translationally invariant versions of both APX-SIM and GSCON are intractable, namely are $\mathrm{P}^{\mathrm{QMA}_{\mathrm{EXP}}}$- and $\mathrm{QCMA}_{\mathrm{EXP}}$-complete ("quantum $\mathrm{P}^{\mathrm{NEXP}}$" and "quantum NEXP"), respectively. Each of these results is attained by giving a respective generic "lifting theorem". For APX-SIM we give a framework for lifting any abstract local circuit-to-Hamiltonian mapping H satisfying mild assumptions to hardness of APX-SIM on the family of Hamiltonians produced by H, while preserving the structural properties of H (e.g. translation invariance, geometry, locality, etc). Each result also leverages counterintuitive properties of our constructions: for APX-SIM, we compress the answers to polynomially many parallel queries to a QMA oracle into a single qubit. For GSCON, we show strong robustness, i.e. soundness even against adversaries acting on all but a single qudit in the system.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Oracles and decision trees; Theory of computation → Quantum complexity theory

Keywords and phrases Complexity, Quantum Computing, Physics, Constraint Satisfaction, Combinatorial Reconfiguration, Many-Body Physics

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.54

Related Version *Full Version*: `https://arxiv.org/abs/2012.12717`

Funding *James D. Watson*: EPSRC Centre for Doctoral Training in Delivering Quantum Technologies (grant EP/L015242/1).
Johannes Bausch: Draper's Research Fellowship at Pembroke College.
Sevag Gharibian: DFG grant 432788384

© James D. Watson, Johannes Bausch, and Sevag Gharibian;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

The quantum generalisation of a Boolean constraint satisfaction problem such as a 3-SAT formula is known as a *k-local Hamiltonian*, $H = \sum_i H_i$. Here, a Hermitian operator H acts on N quantum systems of dimension $d \in O(1)$, so that H is a $d^N \times d^N$ complex matrix. Yet, H has a succinct description of poly(N) bits, in that each H_i acts non-trivially[1] on only $k \in O(1)$ qudits, and hence requires $O(1)$ bits to specify. Any Hamiltonian H captures the static and dynamic properties of some many-body quantum system (via the Schrödinger equation), such as its ground state energy, spectral gap, and time-evolution.

For this reason, the complexity of computing properties of local Hamiltonians has seen intense interest in the last two decades (e.g., [32, 15] for surveys). The "benchmark" problem here has been the quantum generalisation of MAX-k-SAT – approximating the ground state energy of H (i.e. smallest eigenvalue $\lambda_{\min}(H)$ of H, which represents the energy level the system settles into when cooled to low temperature). This is the *Local Hamiltonian Problem (LH)*, known to be QMA-complete [26]. (Quantum-Merlin Arthur (QMA) is a quantum generalisation of Merlin-Arthur (MA), with a quantum proof and verifier.) LH remains hard on physically motivated setups, such as qubits on a 2D lattice [31] and 1D chains of local dimension 8 [2, 30, 22]. Amazingly, it is QMA$_{\text{EXP}}$-complete[2] even for *1D translationally invariant, nearest neighbour* systems [21], meaning on a line of N qudits, with each constraint $H_{i,i+1}$ being *identical* for $i \in [N-1]$ and each constraint only acting between neighbouring pairs of qudits. This in stark contrast to 1D *classical* constraint satisfaction, which can be efficiently solved via divide-and-conquer in time polynomial in the length of the chain (even in non-translationally invariant systems).

Beyond ground state energies. From a physical perspective, what is often more interesting than ground state energies is computing *properties of the ground space* itself – a problem analogous to computing properties of optimal MAX-k-SAT assignments. In this direction, recent works have studied determining a system's density of states [10, 34]; minimising interaction terms yielding frustrated ground spaces [16]; deciding if a ground space has an energy barrier [18, 20]; simulating local measurements on ground spaces [3, 19, 17]; estimating spectral gaps of local Hamiltonians [3, 12, 19]; and "universal" Hamiltonian models simulating other quantum many-body systems [8, 11, 33, 27]. Here, we focus on two of these problems: Simulating local measurements on ground spaces (APX-SIM) [3] and deciding if a ground space has an energy barrier (GSCON) [18].

APX-SIM. The first problem, *Approximate Simulation (APX-SIM)*, asks: How difficult is it to simulate a local measurement on a ground state of a local Hamiltonian? Given that much of condensed matter physics is devoted to determining the low-energy properties of materials, and that local measurements are the only tools available to experimentalists to examine these systems, this is an extremely important problem.

▶ **Definition 1** (APX-SIM$(H, A, k, l, a, b, \delta)$ [3]). *Given k-local Hamiltonian $H = \sum_i H_i$ on N qubits, l-local observable A, and $a, b, \delta \in \mathbb{R}$ such that $b - a \geq N^{-c}$ and $\delta \geq N^{-c'}$, for $c, c' > 0$ constant, decide:*
YES. *If H has a ground state $|\psi\rangle$ satisfying $\langle\psi| A |\psi\rangle \leq a$.*
NO. *If for all $|\psi\rangle$ satisfying $\langle\psi| H |\psi\rangle \leq \lambda_{\min}(H) + \delta$, it holds that $\langle\psi| A |\psi\rangle \geq b$.*

[1] Formally, if H_i acts on a subset S_i of qudits, it is specified via $H_i \otimes I_{[N]\setminus S_i}$, for $[N] = \{1, \ldots, N\}$.
[2] QMA$_{\text{EXP}}$ (Definition 20) is a quantum analogue of NEXP, meaning an exp-length quantum proof and exp-time quantum verifier.

We will also be interested in the version of the problem with a translationally invariant Hamiltonian.

▶ **Definition 2** (TI-APX-SIM$(N, H_i, A, k, l, a, b, \delta)$ [3]). *Defined similarly to Definition 1 except the input is now the systems size N described in $n = O(\log(N))$ many bits, and a local interaction term H_i. The overall Hamiltonian is a translationally invariant Hamiltonian on N qudits $H = \sum_i H_i$, with parameters $b - a \geq N^{-c} = O(2^{-cn})$, $\delta \geq N^{-c'} = O(2^{-c'n})$, for $c, c' > 0$.*

APX-SIM was originally shown $\mathrm{P}^{\mathrm{QMA[log]}}$-complete for 5-local Hamiltonians with 1-local measurements [3, 19]. Here, $\mathrm{P}^{\mathrm{QMA[log]}}$ is the quantum analogue of $\mathrm{P}^{\mathrm{NP[log]}}$, meaning all languages decidable by a P machine making logarithmically many queries to a QMA oracle. It was subsequently shown that APX-SIM remains $\mathrm{P}^{\mathrm{QMA[log]}}$-complete on physically motivated 2D models, and on (non-translationally invariant) 1D chains [17]. While it is unlikely that $\mathrm{QMA} = \mathrm{P}^{\mathrm{QMA[log]}}$ (since $\mathrm{P}^{\mathrm{QMA[log]}}$ trivially contains co-QMA), $\mathrm{P}^{\mathrm{QMA[log]}}$ is "not too much harder" than QMA, in that $\mathrm{P}^{\mathrm{QMA[log]}} \subseteq \mathrm{PP}$ [19]. Finally, in this work we use $\mathrm{P}^{\mathrm{QMA[log]}} = \mathrm{P}^{\|\mathrm{QMA}}$ [17], where $\mathrm{P}^{\|\mathrm{QMA}}$ allows polynomially many parallel queries to the QMA oracle.

GSCON. The second problem we study is *Ground State Connectivity (GSCON)* [18], which asks: given two ground states $|\psi\rangle$ and $|\phi\rangle$ of a local Hamiltonian, is there a low energy path connecting $|\psi\rangle$ to $|\phi\rangle$? Physically, this captures the question of determining if a ground space has an energy barrier.

▶ **Definition 3** (Ground State Connectivity (GSCON $(H, \eta_1, \eta_2, \eta_3, \eta_4, \delta, b, m, U_\psi, U_\phi, |\psi\rangle, |\phi\rangle)$)[18]). *Let $H = \sum_i H_i$ be a k-local Hamiltonian on N qubits. Consider parameters $\eta_1, \eta_2, \eta_3, \eta_4, \delta \in \mathbb{R}$, and $m \in \mathbb{Z}^+$, with $\eta_2 - \eta_1, \eta_4 - \eta_3 \geq \delta$. Let U_ψ and U_ϕ be* poly(N)*-size quantum circuits generating "start" and "target" states $|\psi\rangle = U_\psi |0 \cdots 0\rangle$ and $|\phi\rangle = U_\phi |0 \cdots 0\rangle$, respectively, satisfying $\langle\psi| H |\psi\rangle \leq \eta_1$ and $\langle\phi| H |\phi\rangle \leq \eta_1$. Output:*

YES: *If there exists a sequence of b-local unitaries $(U_i)_{i=1}^m$ such that:*

1. *(Intermediate states remain in low energy space) For all $i \in [m]$ and intermediate states $|\psi_i\rangle := U_i \cdots U_2 U_1 |\psi\rangle$, one has $\langle\psi_i| H |\psi_i\rangle \leq \eta_1$, and*
2. *(Final state close to target state) $\|U_m \cdots U_1 |\psi\rangle - |\phi\rangle\|_2 \leq \eta_3$.*

NO: *If for all b-local sequences of unitaries $(U_i)_{i=1}^m$, either:*

1. *(Intermediate state obtains high energy) There exists $i \in [m]$ and an intermediate state $|\psi_i\rangle := U_i \cdots U_2 U_1 |\psi\rangle$, such that $\langle\psi_i| H |\psi_i\rangle \geq \eta_2$, or*
2. *(Final state far from target state) $\|U_m \cdots U_1 |\psi\rangle - |\phi\rangle\|_2 \geq \eta_4$.*

The translationally invariant version of GSCON can be specified in a similar way, where instead the Hamiltonian is specified by a TI local interaction terms and the systems size.

GSCON is QCMA-complete for 5-local Hamiltonians [18], and remarkably (and in contrast to LH) remains hard even for *commuting* Hamiltonians (i.e. when H_i and H_j commute for all pairs) [20]. (Quantum-Classical Merlin Arthur (QCMA) is QMA, except with a *classical* witness.) GSCON is motivated via quantum memories and stabilizer codes. For example, a Hamiltonian H for a YES instance of GSCON has a short sequence of local unitaries mapping between low energy states $|\psi\rangle$ and $|\phi\rangle$ through the low energy space of H. In a quantum memory, $|\psi\rangle$ and $|\phi\rangle$ may encode logical states. As errors in physical systems are often local, this implies H might not be a good quantum memory – not only can a short adversarial circuit corrupt $|\psi\rangle$ to $|\phi\rangle$ (since there is no energy barrier "separating" $|\psi\rangle$ from $|\phi\rangle$), but this corrupting process takes place completely in the low energy space, meaning such errors are not easily detectable.

When are these problems relevant to physics? From a quantum complexity theoretic standpoint, a key goal is to show that even local Hamiltonian systems mirroring those in Nature can encode "hard" problems. *Translationally-invariant* Hamiltonians, in particular, not only possess symmetries seen in Nature, but are also believed simpler than general Hamiltonians. Intuitively, due to the spatial invariance of the system, the degrees of freedom available to encode complex behaviour appears limited. A second physically motivated restriction is *spatial structure*, such as one-dimensional systems, which can sometimes be more tractable [7, 29, 1, 36, 28, 14]. Combining these two properties, we arrive at one of the simplest systems imaginable – 1D, translationally-invariant (TI) systems. Are these "more tractable" than their higher-dimensional counterparts? As mentioned earlier, LH on such systems surprisingly turns out to be $\mathrm{QMA}_{\mathrm{EXP}}$-complete [21], and the question of existence of a spectral gap undecidable [5]. Yet for natural problems such as APX-SIM and GSCON, the verdict is still open.

1.1 Results

In this work, we give generic "lifting frameworks", which we then apply to show (a) that APX-SIM and GSCON remain "hard" in the 1D TI setting and (b) the Local Hamiltonian problem determines the complexity of these problems. We now discuss these results in depth.

A lifting framework for APX-SIM

We begin with a generic framework for "lifting" hardness results about ground state energies (i.e. LH) to hardness results for APX-SIM. Formally achieved via our Lifting Lemma (Lemma 10) and applications thereof in Section 2.3, the general premise is informally:

▶ **Theorem 4** (LH to APX-SIM (informal))**.** *If a family of Hamiltonians $\mathcal{F}$ admits a circuit-to-Hamiltonian mapping H_{w} such that approximating the ground state energy for $\mathcal{F}$ is* C*-hard (for complexity class C), then APX-SIM for $\mathcal{F}$ is $\mathrm{P}^{\mathrm{C}[\log]}$- or P^{C}-hard for non-TI and TI Hamiltonians, respectively.*

The key point of the lifting map underlying Theorem 4 is that it automatically *preserves structural properties of $\mathcal{F}$*, such as locality, geometry, translational invariance, etc. This has two advantages: First, it obviates the need to reprove hardness for APX-SIM each time a new physically motivated circuit-to-Hamiltonian construction H_{w} is discovered (modulo mild assumptions on H_{w} as per Definition 9). Second, it reveals that LH itself fundamentally characterises the complexity of computing properties, such as simulating measurements on the low energy states of a family of Hamiltonians.

For clarity, throughout this work, 1D TI versions of computational problems assume the input size is n, whereas the length of the 1D chain is $N \in O(\exp(n))$. This is because in the TI setup, it is standard for the input to be given succinctly by (1) the length of the chain in binary and (2) a description of the single $H_{i,i+1}$ term to be repeated along the chain [21].

▶ **Theorem 5.** *APX-SIM is $\mathrm{P}^{\mathrm{QMA}_{\mathrm{EXP}}}$-complete for 1D TI, nearest neighbour, Hamiltonians on N qudits of local dimension* 44*, for $\delta = \Omega(1/\operatorname{poly}(N))$, $b - a = \Omega(1/\operatorname{poly}(N))$.*

We also obtain PSPACE-completeness for 1D TI APX-SIM when $\delta, b - a \in \Omega(1/\exp(N))$. We thus find the first known hardness result for APX SIM in the TI setting. Two points worth highlighting: First, counter-intuitively, our construction "stores" the answers to m QMA queries into a *single* qubit. A similar phenomenon is trivially impossible classically. This "compression" is what allows us to make our setup so generic. Second, one of the steps

to establishing Theorem 5 is to show $\mathrm{EXP}^{||\mathrm{QMA}} = \mathrm{P}^{\mathrm{QMA}_{\mathrm{EXP}}}$. In other words, an exp-time Turing machine making exponentially many parallel QMA queries is equivalent to a poly-time machine making poly-many adaptive queries to $\mathrm{QMA}_{\mathrm{EXP}}$.

Hardness of Ground State Connectivity (GSCON)

Via different techniques, we next give a Lifting Lemma for GSCON (Lemma 15), obtaining the first GSCON hardness result in a physically motivated setting:

▶ **Theorem 6.** GSCON *is* $\mathrm{QCMA}_{\mathrm{EXP}}$*-complete for 1D TI Hamiltonians on* N *qudits of constant local dimension, for* $m \in \mathrm{poly}(N)$*,* $\delta \in \Theta(1/\mathrm{poly}(N))$*, and any* $b \in \{2, \ldots, N-1\}$*.*

Here, $\mathrm{QCMA}_{\mathrm{EXP}}$ (Definition 21) is to QCMA as $\mathrm{QMA}_{\mathrm{EXP}}$ is to QMA, i.e. one has an exp-long classical proof and exp-time quantum verifier. Worth highlighting is that, perhaps surprisingly, Theorem 6 holds even for $b = N-1$. In words, even if an adversary can act jointly on *all but a single qudit* per time step, our construction remains sound. This is significantly more robust than [18], and is tight, since an adversary acting on all N qudits can trivially cheat by mapping $|\psi\rangle$ to $|\phi\rangle$ via a single N-qudit unitary. We remark that the applicability of our lifting theorem for GSCON is not as wide as that for APX-SIM; details in Section 1.2.

1.2 Techniques

Circuit-to-Hamiltonian Mappings

We begin with a brief overview of circuit-to-Hamiltonian mappings. In the literature, a *circuit-to-Hamiltonian mapping* roughly means a map which takes as input any quantum circuit $U = U_T U_{T-1} \ldots U_1$ (e.g. consisting of 2-qubit gates U_i), where U acts on some Hilbert space $\mathcal{H}_q$, and outputs a so-called "history state Hamiltonian" H, whose low energy space "encodes" U. The prototypical example is Kitaev's construction [26], which outputs $H = H_{\mathrm{in}} + H_{\mathrm{prop}} + H_{\mathrm{out}} + H_{\mathrm{stab}}$, where H_{in} forces the input of U to be initialised correctly, H_{prop} that each gate of U follows correctly after all previous gates are applied, H_{out} checks that U accepts, and H_{stab} ensures the clock register is encoded correctly. (The Cook-Levin theorem has analogous Boolean formulae for $H_{\mathrm{in}}, H_{\mathrm{prop}}, H_{\mathrm{out}}$, but does not require H_{stab}, as time is explicitly encoded via rows of the table.) Of these, the most relevant to our discussion is

$$H_{\mathrm{prop}} = \sum_{t=0}^{T-1} h_t \quad \text{where} \quad h_t := \sum_{|e\rangle} \big(|t\rangle|e\rangle - |t+1\rangle U_t |e\rangle \big)\big(\langle t|\langle e| - \langle t+1|\langle e| U_i^\dagger \big),$$

where we sum over a basis $\{|e\rangle\}$ for $\mathcal{H}_q$. The "history states" are then any state $|\psi\rangle$ of the following form (which span the null space of H_{prop})

$$|\psi\rangle = \frac{1}{\sqrt{T}} \sum_{t=0}^{T} |t\rangle |\psi_t\rangle \quad \text{where} \quad |\psi_t\rangle := U_t U_{t-1} \cdots U_1 |\psi_0\rangle \tag{1}$$

for any $|\psi_0\rangle \in \mathcal{H}_q$. For our purposes, we formulate precise definitions of such mappings (Definition 9 and Definition 26) in order to rigorously prove our lifting theorems with as broad generality as possible.

For APX-SIM

To make our lifting framework as general as possible, instead of focusing on $\mathrm{P}^{\|\mathrm{QMA}}$ (which equals $\mathrm{P}^{\mathrm{QMA[log]}}$ [17]), we consider arbitrary classes $\mathrm{D}^{\|\mathrm{Q}}$. Briefly, D is any "deterministic class" (e.g. P, EXP; Definition 22), Q any "existentially quantified quantum verification classes (QVClass)" Q (e.g. NP, QCMA, QMA; Definition 23). We allow almost arbitrary "local circuit-to-Hamiltonian mappings" H_{w} (e.g. Kitaev [25]; Definition 9), the primary requirement of which is that a measurement in the standard basis on the first and second output qubits of any unitary U can be simulated by a measurement on the low energy space of Hamiltonian $H_{\mathrm{w}}(U)$. Formal requirements with all minor assumptions in Definition 9.

With just this basic property of measuring two qubits of the ground state in hand, we give a Lifting Lemma (Lemma 7) which takes any $\mathrm{D}^{\|\mathrm{Q}}$ computation and embeds it in an APX-SIM instance, while automatically preserving structural properties of the circuit-to-Hamiltonian mapping H_{w}. At a high level, we begin with an idea similar to the 1D non-TI $\mathrm{P}^{\mathrm{QMA[log]}}$-hardness result of [17] by replacing all parallel oracle calls to Q with explicit *verification circuits* for Q. In contrast to [17], we then "count" the number of YES Q-queries via a single qubit – each time a Q-verifier outputs YES, we rotate a designated "flag qubit" by a small fixed amount. We then push this entire "bootstrapped" computation through the circuit-to-Hamiltonian mapping H_{w}, followed by use of the "primary requirement" above to simulate a penalty on the flag qubit. Remarkably, by carefully adjusting the weight on this one flag qubit, with high probability we can force *all* Q-queries to simultaneously be answered correctly. *A priori*, this is perhaps surprising; for example, Holevo's theorem [23] says that n qubits cannot transmit more than n bits of information, and yet here we are cramming polynomially many query answers into a single flag qubit, while still meaningfully utilising the information therein.

This flag qubit construction now allows us to circumvent the 1D TI restrictions (since there is only a *single* flag qubit to keep track of, we are not worried about how it is arranged geometrically within the final system). Additionally, a key part of the soundness analysis is an exchange argument (Lemma 11), which may be of independent interest: given a joint entangled proof $|w_{1\cdots m}\rangle$ to m Q-verifiers V_i, if the i^{th} local component of $|w_{1\cdots m}\rangle$ is ϵ-suboptimal for verifier V_i, we give a rigorous lower bound on the deviation from the optimal "counted sum" on the flag qubit.

For GSCON

Using different techniques, we next give a generic Lifting Lemma for GSCON (Lemma 15), although less generic than what we are able to achieve for APX-SIM. Namely, we restrict attention to quantum verification classes such as QCMA or $\mathrm{QCMA}_{\mathrm{EXP}}$ (since we require the ability to prepare low energy/history states efficiently in the YES case), and to general 1D TI circuit-to-Hamiltonian mappings (again, with mild restrictions; see Definition 26). We then apply Lemma 15 to the 1D TI Gottesman-Irani construction [21] to obtain $\mathrm{QCMA}_{\mathrm{EXP}}$-hardness of GSCON on 1D TI systems (Theorem 6).

At a high level, the starting setup for our lifting framework is similar to [18], which we briefly review: given a QCMA verification circuit V, apply Kitaev's circuit-to-Hamiltonian construction to obtain local Hamiltonian $H = \sum_i H_i$, such that if V is a YES (NO) instance, $\lambda_{\min}(H)$ is small (large). Then, attach a "switch gadget" to H to obtain a new Hamiltonian H', so that any polynomial-length traversal of the low energy space of H', from start state $|\psi\rangle$ to target state $|\phi\rangle$, forcibly "switches on" all terms of H. In the NO case, switching on H incurs a large energy penalty, i.e. we hit the claimed energy barrier.

Extending this to the 1D TI setting presents various challenges. First, the construction of [18] is highly non-local geometrically, as each switch qubit is coupled to *all* local terms H_i of H. In order to maintain this level of coupling in 1D, we first use an idea reminiscent of the space-time circuit-to-Hamiltonian construction of Breuckmann and Terhal [9], and instead endow each qudit on the 1D chain with its *own* "local switch qudit". We then wish to add "string constraints" on these "local switch qubits" to force a prover to "switch on" each term $H_{i,i+1}$ of the 1D Hamiltonian one at a time on the chain. But here we face an additional pair of challenges: first, any naïve implementation of string constraints allows a cheating prover to switch on only $N - b$ of the chain's local terms $H_{i,i+1}$ – this is because the prover is allowed to apply arbitrary b-local unitaries in each step, allowing it to "shortcut" the last b switch qudits in one step. Second, we cannot satisfy the desired completeness properties for GSCON by simply switching on local terms $H_{i,i+1}$ iteratively from left to right. Rather, we must allow a *non-linear* order of activation.

It turns out that, not only can both the second and third challenges above be addressed in a unified black-box fashion, but the unified fix will also make the construction remarkably robust from a soundness perspective. Specifically, we first increase the local switch Hilbert space dimension to 7, which roughly will allow non-linear activation orders when switching on the local constrain terms. We then carefully construct our string constraints so that any ground space evolution satisfying said constraints becomes "trapped" in a low-dimensional joint switch subspace on all qudits. This low-dimensional space is precisely set up to achieve two things: (1) force *all* local terms of H to be simultaneously switched on, and (2) be "logically protected" from any switch subspace deviating from property (1) by a "string" of local unitaries of length $\Theta(N)$. The formal proof of correctness uses, among other tools, the Traversal Lemma (Lemma 14) of [18, 20]. Perhaps counterintuitively, soundness holds even if a cheating prover can apply $(N-1)$-local unitaries in each step, i.e. can act on *all but one* qudit of the chain per step.

1.3 Open questions

For APX-SIM, our lifting framework not only simplifies existing $\mathrm{P}^{\mathrm{QMA}[\log]}$-hardness proofs of APX-SIM [3, 19, 17], but also yields new hardness results, notably for 1D TI systems: $\mathrm{P}^{\mathrm{QMA}_{\mathrm{EXP}}}$-completeness and PSPACE-completeness for inverse polynomial and inverse exponential precision (with respect to the length of the chain), respectively. Can our techniques, such as "compressing" multiple queries into a single qubit (Lemmas 7) and 11, find use elsewhere in studying quantum oracle classes? Can our APX-SIM results be generalised to yet more physical Hamiltonians (e.g. using Hamiltonian simulation techniques [17])? For GSCON, does our "logically protected" switch subspace design have applications beyond complexity theory, e.g. to robust quantum memories? Can our GSCON lifting framework be generalised to the broader class of "local circuit-to-Hamiltonian constructions" (Definition 9), as in APX-SIM? Most interestingly, do there exist non-trivial classes of Hamiltonians for which GSCON is easy?

Organisation

Section 2 and Section 3 give detailed proof sketches of our main results.

Notation

$\mathrm{Herm}(\mathcal{X})$ and $\mathrm{U}(\mathcal{X})$ denote the sets of linear and unitary operators acting on space $\mathcal{X}$, respectively. For $H \in \mathrm{Herm}(\mathcal{X})$, $\lambda_{\min}(H)$ is its smallest eigenvalue, and $\Delta(H)$ its spectral gap (i.e. gap between two smallest distinct eigenvalues of A), i.e. $\Delta(A) \coloneqq \lambda_1(A) - \lambda_{\min}(A)$. $\mathrm{Null}(A)$ is the null-space/kernel of A.

2 Hardness via Lifting for APX-SIM

We now prove our lifting results for APX-SIM. Where appropriate, full proofs are deferred or can be found in [35].

2.1 Reducing $\mathrm{P}^{\|\mathrm{QMA}}$ to a single quantum verification circuit

While it is instructive to view $\mathrm{P}^{\|\mathrm{QMA}}$ as the "guiding example" for this section, our statements below apply more generally to classes of form $\mathrm{D}^{\|\mathrm{Q}}$. Namely, let D be a *Deterministic Decision Class* if the set of languages it contains can be deterministically decided with some time and space resources as a function of the input length (e.g. P, PSPACE, EXP, etc., Definition 22). Let Q be an *Existentially quantified quantum verification class (QVClass)* if it consists of promise problems verifiable by a uniform family of quantum verifiers given access to a quantum proof $|\psi\rangle$, and with some completeness/soundness parameters c and s (e.g. NP, QMA, $\mathrm{QMA}_{\mathrm{EXP}}$. Definition 23 makes no restrictions on c, s, uniformity resources, etc).

The first step of our construction is to map an arbitrary $\mathrm{D}^{\|\mathrm{Q}}$ computation to a single "verification circuit".

▶ **Lemma 7.** *Let $x \in \{0,1\}^n$ be an instance of a problem in $\mathrm{D}^{\|\mathrm{Q}}$, which is decided by $\mathrm{D}^{\|\mathrm{Q}}$ machine U. There exists an efficiently computable (in encoding sizes of x and U) quantum circuit V satisfying:*

1. *V takes as input $m+2$ registers: input register A containing $x \in \{0,1\}^n$, m proof registers B_i containing joint quantum proof $|w_{1\cdots m}\rangle$, with register B_i to be verified by a Q-circuit V_i (Figure 1). Without loss of generality, each verifier V_i has the same completeness and soundness parameters c and s, respectively.*
2. *V has two designated output wires: q_{out} encodes the output of U, and q_{flag}, the state of which encodes the number of Q queries made by U which were YES instances. Let $|\psi\rangle$ denote the output state of V, given joint proof $|w_{1\cdots m}\rangle$.*
   Let S_0 and S_1 partition $\{0,1\}^m$ such that the D machine underlying U rejects (accepts) given a string of query responses $y \in S_0$ ($y \in S_1$). Define $p_{y,w} \coloneqq \Pr\left(\bigwedge_{i=1}^m V_i \text{ outputs } y_i \,\middle|\, |w_{1\cdots m}\rangle\right)$. Then,

$$\mathrm{Tr}\left(|\psi\rangle\langle\psi| \cdot |1\rangle\langle 1|_{q_{\mathrm{out}}}\right) = \sum_{y\in S_1} p_{y,w} \tag{2}$$

$$\mathrm{Tr}\left(|\psi\rangle\langle\psi| \cdot |1\rangle\langle 1|_{q_{\mathrm{flag}}}\right) = \sum_{y\in\{0,1\}^m} p_{y,w} \cdot \sin^2\left(\frac{\sqrt{3}}{2m}\cdot \mathrm{HW}(y)\right). \tag{3}$$

   where $HW(y)$ is the Hamming weight of $y \in \{0,1\}^m$.

Proof. As depicted in Figure 1, V is constructed by translating the D machine underlying U into a quantum circuit U', and then "simulating" the m (parallel) oracle calls U makes as sub-routines by executing their Q-verification circuits V_i on the relevant subsets of $|w_{1\cdots m}\rangle$. Note U' is diagonal in the standard basis, and U computes the inputs $|q_i\rangle$ to Q-verification circuits V_i on-the-fly given x. Gate $R(\theta)$ in Figure 1 denotes 2×2 the rotation matrix $R(\theta) = (\cos\theta, -\sin\theta; \sin\theta, \cos\theta)$.

Let X, Y, Z denote the input registers to U' holding input $x \in \{0,1\}^m$, query response string $y = y_1 \cdots y_m$, and ancilla (initialised to all zeroes), respectively. Since U' is a classical circuit, without loss of generality it maps any $|x\rangle_X |y\rangle_Y |0\cdots 0\rangle_Z \mapsto |x\rangle_X |y\rangle_Y |0\cdots 0 f(y)\rangle_Z$, where $f(y)$ is the output of U' given query response string $y \in \{0,1\}^m$. If F denotes the flag qubit register, the output $|\psi\rangle$ of V is given by

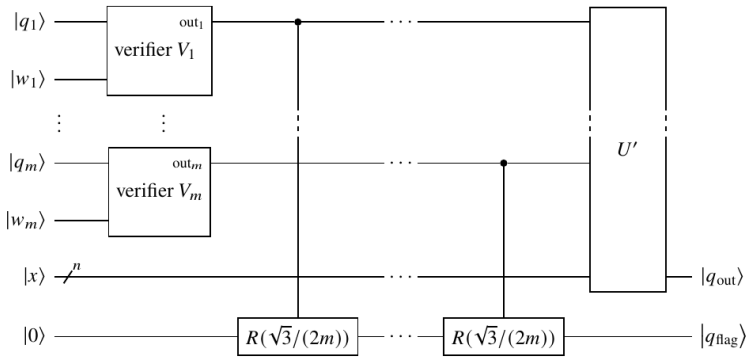


Figure 1 The circuit V constructed in Lemma 7. The V_i are Q-verifiers, each taking input $|q_i\rangle$ and proof/witness $|w_i\rangle$. For simplicity, states $|w_i\rangle$ are illustrated in tensor product, but our proofs treat the general case of entangled joint proof $|w_{1\cdots m}\rangle$. U' denotes the host postprocessing circuit in the original $\mathrm{D}^{\|\mathrm{Q}}$ circuit U, which takes the Q-query responses and outputs U's final answer. We omit any preprocessing needed by U to compute inputs $|q_i\rangle$ and ancilla register C.

$$|\psi\rangle = \sum_{y\in\{0,1\}^m} \alpha_y |x\rangle_X |y\rangle_Y |0\cdots 0 f(y)\rangle_Z \left(\cos\left(\frac{\sqrt{3}}{2m}\cdot \mathrm{HW}(y)\right)|0\rangle + \sin\left(\frac{\sqrt{3}}{2m}\cdot \mathrm{HW}(y)\right)|1\rangle\right)_F, \quad (4)$$

where we omit registers such as those containing proof $|w_{1\cdots m}\rangle$, $\mathrm{HW}(y)$ is the Hamming weight of y, and $|\alpha_y|^2 = \Pr\left(\bigwedge_{i=1}^m V_i \text{ outputs } y_i \,\middle|\, |w_{1\cdots m}\rangle\right)$. This immediately yields Equations (2) and (3). ◀

▶ Remark. The flag qubit q_{flag} does not use binary to store the number of queries which output YES, but rather the number is stored in the angle the qubit is rotated by from its initial state.

▶ Remark. It is *not* true in Figure 1 that the optimal strategy of a dishonest prover is to send the *optimal* proof $|w_i^*\rangle$ for each verifier V_i. This is because intentionally sending a rejecting proof $|w_i\rangle$ to V_i (even if q_i is a YES instance) sets $y_i = 0$, which may cause U' to incorrectly output 1 (whereas setting $y_i = 1$ might cause U' to output 0). Indeed, if sending $|w_i^*\rangle$ *was* the malicious prover's optimal strategy, then Lemma 7 itself maps an arbitrary $\mathrm{P}^{\|\mathrm{QMA}}$ computation to a single QMA instance V, implying $\mathrm{P}^{\|\mathrm{QMA}} = \mathrm{QMA}$.

2.2 Generic Hardness Constructions via a Lifting Lemma

We will in particular be interested in Hamiltonians which satisfy the following condition:

▶ **Definition 8** (Conformity). *Let H be a Hamiltonian with some well-defined structure S (such as k-local interactions, all constraints drawn from a fixed finite family, with a fixed geometry such as 1D, translational invariance, etc). We say a Hermitian operator P* conforms *to H if $H + P$ also has structure S.*

For example, if H is a 1D translationally invariant Hamiltonian on qubits, then P conforms to H if $H + P$ is also 1D translationally invariant.

▶ **Definition 9** (Local Circuit-to-Hamiltonian Mapping). *Let $\mathcal{X} = (\mathbb{C}^2)^{\otimes m}$ and $\mathcal{Y} = (\mathbb{C}^2)^{\otimes n}$. A map $H_{\mathrm{w}} : \mathrm{U}(\mathcal{X}) \mapsto \mathrm{Herm}(\mathcal{Y})$ is a* local circuit-to-Hamiltonian mapping *if, for any $T > 0$ and any sequence of 2-qubit unitary gates $U = U_T U_{T-1} \cdots U_1$, the following hold:*

1. *(Overall structure)* $H_{\mathrm{w}}(U) \succeq 0$ *has a non-trivial null space, i.e.* $\mathrm{Null}\left(H_{\mathrm{w}}(U)\right) \neq \{0\}$. *This null space is spanned by (some appropriate notion of) "correctly initialised computation history states"*[3]*, i.e. with ancillae qubits set "correctly" and gates in* U *"applied" sequentially.*
2. *(Local penalisation and measurement) Let* q_1 *and* q_2 *be the first two output wires of* U *(each a single qubit), respectively. Let* $S_{\mathrm{pre}} \subseteq \mathcal{X}$ *and* $S_{\mathrm{post}} \subseteq \mathcal{Y}$ *denote the sets of input states to* U *satisfying the structure enforced by* $H_{\mathrm{w}}(U)$ *(e.g. ancillae initialised to zeroes), and null states of* $H_{\mathrm{w}}(U)$*, respectively. Then, there exist projectors* M_1 *and* P_T*, projector* M_2 *conforming to* $H_{\mathrm{w}}(U)$*, and a bijection* $f : S_{\mathrm{pre}} \mapsto S_{\mathrm{post}}$*, such that for all* $i \in \{1,2\}$ *and* $|\phi\rangle \in S_{\mathrm{pre}}$*, the state* $|\psi\rangle = f(|\phi\rangle)$ *satisfies*

$$\mathrm{Tr}\left(|0\rangle\langle 0|_i \left(U_T U_{T-1} \dots U_1\right) |\phi\rangle\langle\phi| \left(U_T U_{T-1} \dots U_1\right)^\dagger \right) = \mathrm{Tr}\left(|\psi_T\rangle\langle\psi_T| \, M_i \right), \tag{5}$$

where $|\psi_T\rangle = P_T |\psi\rangle \,/\, \|P_T |\psi\rangle\|_2$ *is* $|\psi\rangle$ *postselected on measurement outcome* P_T *(we require* $P_T |\psi\rangle \neq 0$*). Moreover, there exists a function* $g : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{R}$ *such that*

$$\|P_T |\psi\rangle\|_2^2 = g(m,T) \text{ for all } |\psi\rangle \in \mathrm{Null}\left(H_{\mathrm{w}}(U)\right), \tag{6}$$

$$M_i = P_T M_i P_T. \tag{7}$$

The map H_{w}*, and all operators/functions above (*M_1*,*M_2*,*P_T*,*f*,*g*) are computable given* U*.*

We next embed the circuits V constructed in Lemma 7 into a "local circuit-to-Hamiltonian construction", and carefully penalise the *flag* qubit – not the output qubit – to encourage the ground space of $H_{\mathrm{w}}(V)$ to encode correct query answers made by the D machine to the Q oracle. The latter is necessary since the reduction of Lemma 7 is not *sound*, meaning a NO instance of $\mathrm{P}^{\|\mathrm{QMA}}$ is not necessarily mapped to a "NO QMA circuit" V. For this, we formalise and use a broad notion of "local circuit-to-Hamiltonian mapping" H_{w} in Definition 9.

Coupling our single-qubit flag register setup from Lemma 7 with generic black-box usage of "local circuit-to-Hamiltonian mappings H_{w}" allows us to give the main workhorse of this section, the Lifting Lemma for APX-SIM. Crucially, Definition 8 and Definition 9 ensure H below automatically maintains desirable structural properties of H_{w} (such as translational invariance, geometric constraints, etc).

▶ **Lemma 10** (Lifting Lemma for APX-SIM). *Let* $x \in \{0,1\}^n$ *be an instance of an arbitrary* $\mathrm{D}^{\|\mathrm{Q}}$ *problem,* U *a* $\mathrm{D}^{\|\mathrm{Q}}$ *machine deciding* x*, and* V *the verification circuit output by Lemma 7. Fix a local circuit-to-Hamiltonian mapping* H_{w}*, and assume the notation in Definition 9. Fix any* $\alpha : \mathbb{N} \mapsto \mathbb{N}$ *such that* $\alpha > \max\left(\frac{4\|M_2\|}{\Delta(H_{\mathrm{w}}(V))}, \frac{\Delta(H_{\mathrm{w}}(V))}{3\|M_2\|^2}, 1\right)$*. Then, for any* ϵ *satisfying* $0 \leq \epsilon \leq \frac{1}{\alpha}\left(\frac{1}{\alpha} + \frac{12\|M_2\|^2}{\Delta(H_{\mathrm{w}}(V))}\right)\left(\frac{8m^2}{3g(m,T)}\right)$*. the Hamiltonian* $H \coloneqq \alpha(n) H_{\mathrm{w}}(V) + M_2$ *satisfies:*

- *If* x *is a YES instance, then for all* $|\psi\rangle$ *with* $\langle\psi| H |\psi\rangle \leq \lambda_{\min}(H) + \frac{1}{\alpha^2}$*,* $\langle\psi| M_1 |\psi\rangle \leq g(m,T) \cdot m \cdot \max(1 - c + \epsilon, s) + \frac{12\|M_2\|}{\alpha\Delta}$*.*
- *If* x *is a NO instance, then for all* $|\psi\rangle$ *with* $\langle\psi| H |\psi\rangle \leq \lambda_{\min}(H) + \frac{1}{\alpha^2}$*,*

$$\langle\psi| M_1 |\psi\rangle \geq g(m,T)\left(1 - m \cdot \max(1 - c + \epsilon, s)\right) - \frac{12\,\|M_2\|}{\alpha\Delta}.$$

[3] We are intentionally being vague here, as the only formal requirement on the null space of $H_{\mathrm{w}}(U)$ is that of the following bullet on "local penalisation and measurement". Intuitively, this would appear to necessitate the null space of $H_{\mathrm{w}}(U)$ to indeed encode "correct initialisations" and "correct gate applications", as do all known circuit-to-Hamiltonian constructions.

Table 1 Completeness of APX-SIM for families of many-body systems. Measurement precision is relative to system size N, not input size n. NN = 2-local nearest-neighbour interactions. TI = translationally-invariant.

circuit-to-Ham. construction	interaction topology	measurement precision	completeness
[31]	2D planar, NN, local dim 2	$\frac{1}{\text{poly}}$	$\text{P}^{\|\text{QMA}}$
[2]	1D line, NN, local dim 12		
[13]	3-local, local dim 2	$1/\exp$	PSPACE
[4]	TI, 1D line, NN, local dim 44	$\frac{1}{\text{poly}}$	$\text{EXP}^{\|\text{QMA}}$
[6]	TI, 3D fcc lattice, 4-local, local dim 4		$\text{EXP}^{\|\text{QMA}}$
[27]	TI, 1D line, NN, local dim 42	$1/\exp$	PSPACE

Note the Lifting Lemma's sole degree of freedom is the function α. All other quantities stem from the choices of H_{w}, D, and Q. M_1 plays the role of observable A from Definition 24.

Proof sketch. We proceed via three steps: *(i)* We first use the Extended Projection Lemma (Lemma 25) to show that for α as in the lemma claim, for any $|\psi\rangle$ such that $\langle\psi| H |\psi\rangle \leq \lambda_{\min}(H) + \delta$, there exists a uniform history state $|\phi\rangle \in \text{Null}(H_{\text{w}}(V))$ such that $\| |\psi\rangle\langle\psi| - |\phi\rangle\langle\phi| \|_{\text{tr}} \leq \frac{12\|M_2\|}{\alpha\Delta}$ and where $|\phi\rangle$ has energy $\langle\phi| H |\phi\rangle \leq \lambda_{\min}(H) + \delta + \frac{12\|M_2\|^2}{\alpha\Delta}$. *(ii)* These history states correctly simulate V from Lemma 7 (as given in Equation (1)) on any claimed proof $|w_{1\cdots m}\rangle$ to the parallel Q-verifiers. However, the M_2 term in H, which simulates penalising the flag qubit of V, enforces that any such low energy history state must in fact send the *locally optimal* proofs $|w_{1\cdots m}\rangle = |w_1^*\rangle \otimes \cdots \otimes |w_m^*\rangle$, ensuring all Q-queries are answered correctly. The main technical ingredient behind rigorously proving this is the following lemma.

▶ **Lemma 11.** *Assume the notation of Lemma 7, which showed* $\text{Tr}(|\psi\rangle\langle\psi| \cdot |0\rangle\langle 0|_{q_{\text{flag}}} = \sum_{y\in\{0,1\}^m} p_{y,w} \cdot \cos^2\left(\text{HW}(y)\sqrt{3}/(2m)\right)$, *where* $|\psi\rangle$ *denoted the output of* V *given joint proof* $|w_{1\cdots m}\rangle$, *and* $p_{y,w} = \Pr\left(\bigwedge_{i=1}^m V_i \text{ outputs } y_i \mid |w_{1\cdots m}\rangle\right)$. *Suppose there exists an* $i \in \{1,\ldots,m\}$ *and* $\epsilon > 0$ *such that* $|w_{1\cdots m}\rangle$ *is "*ϵ*-suboptimal on proof* i*", meaning there exists a local proof* $|w_i'\rangle$ *such that* $\Pr(V_i \text{ outputs } 1 \mid |w_{1\cdots m}\rangle) = \Pr(V_i \text{ outputs } 1 \mid |w_i'\rangle) - \epsilon$. *Then there exists a proof* $|w_{1\cdots m}'\rangle = |w_1'\rangle \otimes \cdots \otimes |w_m'\rangle$ *which causes* V *to output* $|\psi'\rangle$ *satisfying* $\text{Tr}\left(|\psi\rangle\langle\psi| \cdot |0\rangle\langle 0|_{q_{\text{flag}}}\right) \geq \text{Tr}\left(|\psi'\rangle\langle\psi'| \cdot |0\rangle\langle 0|_{q_{\text{flag}}}\right) + \frac{3}{8m^2}\epsilon$.

Lemma 11 ties the energy penalty on the flag qubit of V to the optimality of all Q-queries. It is proven via a careful exchange argument involving a pair of recursions – the delicacy lies in the fact that we require a rigorous deviation bound (i.e. $3\epsilon/8m^2$), and for this we must take conditional probabilities into account due to potential entanglement between proofs. A full proof is provided in the full version of the paper [35]. ◀

2.3 Applying the Lifting Lemma

With Lemma 10 in hand, we show the hardness results depicted in the rightmost column of Table 1. Technically, our construction gives a slightly stronger result, namely hardness for $\forall$-APX-SIM (Definition 24), for which the YES case reads "*for all* $|\psi\rangle$ satisfying $\langle\psi| H |\psi\rangle \leq \lambda_{\min}(H) + \delta$, $\langle\psi| A |\psi\rangle \leq a$". This immediately implies hardness for APX-SIM as well, since $\forall$-APX-SIM trivially reduces to it.

▶ **Corollary 12.** *$\forall$-APX-SIM is complete for the Hamiltonians and respective classes in Table 1. In particular, $\forall$-APX-SIM (and hence APX-SIM) is $\mathrm{EXP}^{\|\mathrm{QMA}}$-hard for a 1D TI Hamiltonian on qudits of local dimension 44, 1-local observable A, $\delta = 1/\operatorname{poly}(N)$, $b - a = \Omega(1/\operatorname{poly}(N))$.*

▶ Remark. (1) In TI settings, one need be slightly careful in how Lemma 10 is applied, as we cannot explicitly write out the full circuit of Figure 1. (2) We also obtain PSPACE-completeness for inverse exponential promise gap (relative to system size N). Previously, LH with such promise gap was shown PSPACE-complete [13]. (3) We also recover existing $\mathrm{P}^{\|\mathrm{QMA}}$-hardness results of [3, 19, 17] except for $\mathrm{P}^{\|\mathrm{StoqMA}}$-completeness [17] (Lemma 10 requires error reduction, not known to hold for StoqMA).

3 Hardness via Lifting for GSCON

We now show Theorem 6, which recall says GSCON is $\mathrm{QCMA}_{\mathrm{EXP}}$-complete for 1D TI Hamiltonians on N qudits. To begin, we require the following tools.

▶ **Definition 13** (b-orthogonal states and subspaces [18])**.** *For $b \geq 1$, a pair of states $|v\rangle, |w\rangle \in (\mathbb{C}^d)^{\otimes N}$ is* b-orthogonal *if for all b-qudit unitaries U, we have $\langle w| U |v\rangle = 0$. We call subspaces $S, T \subseteq (\mathbb{C}^d)^{\otimes N}$* b-orthogonal *if any pair of vectors $|v\rangle \in S$ and $|w\rangle \in T$ are b-orthogonal.*

▶ **Lemma 14** (Traversal Lemma [18, 20])**.** *Let $S, T \subseteq (\mathbb{C}^d)^{\otimes N}$ be b-orthogonal subspaces. Fix arbitrary states $|v\rangle \in S$ and $|w\rangle \in T$, and consider a sequence of b-qudit unitaries $(U_i)_{i=1}^m$ such that $\| |w\rangle - U_m \cdots U_1 |v\rangle \|_2 \leq \epsilon$ for some $0 \leq \epsilon < 1/2$. Define $|v_i\rangle := U_i \cdots U_1 |v\rangle$ and $P := I - \Pi_S - \Pi_T$. Then, there exists an $i \in [m]$ such that $\langle v_i| P |v_i\rangle \geq ((1-\epsilon)/m)^2$.*

3.1 Generic hardness constructions via a Lifting Lemma

We now give a black-box mapping for "lifting" 1D TI circuit-to-Hamiltonian constructions to $\mathrm{QCMA}_{\mathrm{EXP}}$-hardness results for GSCON. While the goal is similar to the Lifting Lemma for APX-SIM, here we restrict attention to a broad class of 1D TI circuit-to-Hamiltonian mappings we denote *TI-standard* (Definition 26), with two primary properties: (1) In the YES case, there exists a low energy state $|\psi_{\mathrm{low}}\rangle$ preparable in time $\operatorname{poly}(N)$, for N the length of the chain (hence our focus on $\mathrm{QCMA}_{\mathrm{EXP}}$, not $\mathrm{QMA}_{\mathrm{EXP}}$), and (2) the local H_i need not be positive, but the set of H_i with $\langle \psi_{\mathrm{low}}| H_i |\psi_{\mathrm{low}}\rangle < 0$ is computable in time $\operatorname{poly}(N)$. Again, these assumptions are rather mild, and satisfied by most, if not all known circuit-to-Hamiltonian constructions.

▶ **Lemma 15** (Lifting Lemma for GSCON)**.** *Let V be a verifier for a $\mathrm{QCMA}_{\mathrm{EXP}}$ promise problem. Fix any TI-standard circuit-to-Hamiltonian mapping which produces 1D TI Hamiltonians on qudits of local dimension d, and any $b \in \{2, \ldots, N-1\}$. Then, there exists a $\operatorname{poly}(\log N)$-time many-one reduction mapping any instance x for V to a 1D TI GSCON instance H on N qudits of local dimension $7d$ with b-local unitaries U_i, such that $m \in \operatorname{poly}(N)$ and $\delta \in \Theta(1/\operatorname{poly}(N))$.*

3.1.1 Proof of Lemma 15

The construction is stated below – Definition 16 gives the Hamiltonian, and the remaining parameters are set subsequently. The intuition was outlined in Section 1, and is fleshed out in the full version in multiple steps. Here, we recount the most crucial points, and point the

	0	1	2	3	4	5	6
0	✓	✓	✗	✗	✗	✗	✗
1	✓	✓	✓	✗	✗	✗	✗
2	✗	✗	✓	✗	✗	✗	✗
3	✗	✗	✓	✓	✓	✗	✗
4	✗	✗	✗	✗	✓	✗	✗
5	✗	✗	✗	✗	✓	✓	✓
6	✗	✗	✗	✗	✗	✓	✓

Phase	Switch register contents
Warm up	$0000 \mapsto 1000 \mapsto 1010$
Full blast	$1010 \mapsto 1110 \mapsto 1111$
Left deke	$1111 \mapsto 1112 \mapsto 1122 \mapsto 1222 \mapsto 2222$
Right deke	$2222 \mapsto 3222 \mapsto 3322 \mapsto 3332 \mapsto 3333$
Left deke	$3333 \mapsto 3334 \mapsto 3344 \mapsto 3444 \mapsto 4444$
Right deke	$4444 \mapsto 5444 \mapsto 5544 \mapsto 5554 \mapsto 5555$
Cool down	$5555 \mapsto 5556 \mapsto 5656$
Complete shutdown	$5656 \mapsto 5666 \mapsto 6666$

Figure 2 *Left*: A ✓ (✗) at position (i, j) means symbol i can (cannot) appear immediately to the left of j in joint clock space $\mathcal{B}$. *Right*: An honest prover's sequence of states in $\mathcal{B}$, organised by phase, and for $N = 4$. In this example, for concreteness we assume $F = \{1, 3\}$ (see Definition 26).

reader to the completeness analysis of Appendix C.2 for further insight: (1) Each qudit in our system is given its own "clock register" to permit a 1D construction, and (2) the Hamiltonian H, start/end states $|\psi\rangle$ and $|\phi\rangle$ are designed to force any low-energy space evolution to effectively "wind through" a pre-defined path in the "clock space", along which lies a carefully placed "bottleneck" which "switches" on a simulation of the $\mathrm{QCMA}_{\mathrm{EXP}}$ verifier V.

▶ **Definition 16** (Lifted Hamiltonian). *Let x be an instance of a $\mathrm{QCMA}_{\mathrm{EXP}}$ promise problem, with verification circuit V such that for any YES instance x, V accepts some classical proof with probability at least $1-\epsilon$, and for all NO instances x, V accepts all proofs with probability at most ϵ. Let $H' = \sum_{i=1}^{N-1} H'_{i,i+1}$, be the Hamiltonian generated by applying a TI-standard construction (Definition 26) to V. Define E as the 2-local projector onto the set of forbidden 2-local nearest-neighbour substrings in figure 2 (left), where the $\mathcal{B}_i$ are the local clock registers. Define the lifted Hamiltonian as $H = \sum_{i=1}^{N-1} H_{i,i+1}$, where:*

$$H_{i,i+1} \quad := \quad (H'_{i,i+1})_{\mathcal{A}_i,\mathcal{A}_{i+1}} \otimes (|1\rangle\langle 1| + |2\rangle\langle 2| + |3\rangle\langle 3| + |4\rangle\langle 4| + |5\rangle\langle 5|)_{\mathcal{B}_i} + \Delta E_{\mathcal{B}_i,\mathcal{B}_{i+1}} \quad (8)$$

for $\Delta \in \mathbb{R}$ to be chosen as needed. Note each $\mathcal{A}_i \otimes \mathcal{B}_i$ is viewed jointly as qudit i.

The start and final states are $|\psi\rangle = \bigotimes_{i=1}^{N} |0\rangle_{\mathcal{A}_i} |0\rangle_{\mathcal{B}_i}$ and $|\phi\rangle = \bigotimes_{i=1}^{N} |0\rangle_{\mathcal{A}_i} |6\rangle_{\mathcal{B}_i}$, respectively. Set $\eta_1 = \alpha$, $\eta_2 = \beta/(8m^2)$, $\eta_3 = 0$, $\eta_4 = 1/2$, and $m = 2L + 7N$, where $L \in \mathrm{poly}(N)$ is the size of the circuit preparing the ground state of H' in the YES case. Set $\delta = (\eta_1 + \eta_2)/2$, which by definition of TI-standard is at least inverse polynomial in N (since β is at least inverse polynomial in N). Finally, set any $b \in \{2, \ldots, N-1\}$ (b the locality of each U_i); $b = 2$ suffices to show completeness, and soundness holds for all $b \in \{2, \ldots, N-1\}$.

Completeness

Suppose x is a YES instance. The following lemma shows there is a short path through the low energy subspace between states $|\psi\rangle$ and $|\phi\rangle$, as desired.

▶ **Lemma 17** (Completeness). *Let $\lambda_{\min}(H') \leq \alpha$. There exists a circuit $U = U_m \ldots U_2 U_1$ of 2-local gates U_i such that $U|\psi\rangle = |\phi\rangle$, and all intermediate states $|\psi_i\rangle = U_i \ldots U_2 U_1 |\psi\rangle$ satisfy $\langle\psi_i| H |\psi_i\rangle \leq \eta_1$.*

Proof sketch. The full analysis is given in Appendix C. The high-level idea is as follows: Since x is a YES instance, H' has a low energy history state $|\psi_{\mathrm{low}}\rangle$, which by Definition 26 can be prepared in $\mathrm{poly}(N)$ time. To traverse the low-energy space of H from $|\psi\rangle$ to $|\phi\rangle$,

prepare $|\psi_{\text{low}}\rangle$ in $\mathcal{A}$, map the clock registers from $|0\rangle^{\otimes N}$ to $|6\rangle^{\otimes N}$ according to the rules of Figure 2 (left); an $N = 4$ example is in Figure 2 (right). Roughly, the Warm up and Full blast (and by symmetry, Cool down, Complete shutdown) phases allow the honest prover to "switch on" local terms of H_i in an arbitrary order, required since we cannot assume $H'_i \succeq 0$ for all i in Definition 26 (necessary for [21]). The remaining four phases are for soundness (Lemma 18). ◀

Soundness. Suppose x is a NO instance. The following lemma shows that any short path from $|\psi\rangle$ to $|\phi\rangle$ must leave the low-energy subspace.

▶ **Lemma 18.** *Let $\lambda_{\min}(H') \geq \beta$, and fix any $b \in \{2, \ldots, N-1\}$. Consider any sequence $U = U_m \cdots U_1$ of b-local unitary operators acting on $\bigotimes_{i=1}^{N} \mathcal{A}_i \otimes \mathcal{B}_i$. Then, either there exists $i \in [m]$ such that intermediate state $|\psi_i\rangle \coloneqq U_i \cdots U_2 U_1 |\psi\rangle$ satisfies $\langle\psi_i| H |\psi_i\rangle \geq \frac{1}{2}\left(\frac{\beta}{4m^2}\right) = \eta_2$, or $\| |\psi_m\rangle - |\phi\rangle \|_2 \geq 1/2 = \eta_4$.*

Proof sketch. The full analysis is given in Appendix C. At a high level, in the NO case, H' does not have a low-energy state $|\psi_{\text{low}}\rangle$ to prepare in $\mathcal{A}$. Thus, the aim is to force the cheating prover to switch on all terms H'_i, which inflicts energy penalty at least β on register $\mathcal{A}$. The catch is that the prover can apply b-local unitaries, potentially attempting to bypass the last b switches on the chain via a single unitary. Via a careful application of the Traversal Lemma, we show that there exists a time step i, such that intermediate state $|\psi_i\rangle$ has non-trivial overlap in $\mathcal{B}_{N-b} \otimes \cdots \otimes \mathcal{B}_N$ on regular expression $33^*(2^* \cup 4^*)$. By the rules of Figure 2 (left), we deduce that all switch qudits to the left of $\mathcal{B}_{N-b}$ are also set to $|3\rangle$; but this guarantees all terms of H'_i are switched on, as desired. Note the Traversal Lemma alone cannot ensure that *all* H'_i are turned on; it is the delicate combination of the rules of Figure 2 (left) and the Traversal Lemma which make this possible. ◀

3.2 Proof of QCMA$_{\text{EXP}}$-completeness

With the Lifting Lemma for GSCON (Lemma 15) in hand, we obtain our QCMA$_{\text{EXP}}$-completeness result.

▶ **Theorem 19.** *GSCON is* QCMA$_{\text{EXP}}$*-complete for 1D, nearest neighbour, translationally invariant Hamiltonians on N qudits, for $m \in \text{poly}(N)$, $\delta \in \Theta(1/\text{poly}(N))$, and any $b \in \{2, \ldots, N-1\}$.*

Proof sketch. Containment in QCMA$_{\text{EXP}}$ for $\delta \in \Omega(1/\text{poly}(N))$ is immediate since GSCON $\in$ QCMA for any interaction graph [18]. QCMA$_{\text{EXP}}$-hardness of GSCON follows from plugging GI into Lemma 15. Note GI itself is not TI-standard, as it does not satisfy the requirement $\beta \geq 16(2L + 7N)\alpha \geq 0$, but this is easily addressed via energy shifts. ◀

References

1 Ian Affleck, Tom Kennedy, Elliott H. Lieb, and Hal Tasaki. Rigorous results on valence-bond ground states in antiferromagnets. *Physical Review Letters*, 59(7):799–802, August 1987. doi:10.1103/PhysRevLett.59.799.

2 Dorit Aharonov, Daniel Gottesman, Sandy Irani, and Julia Kempe. The power of quantum systems on a line. *Communications in Mathematical Physics*, 287(1):41–65, May 2009. doi:10.1007/s00220-008-0710-3.

3 Andris Ambainis. On physical problems that are slightly more difficult than QMA. In *29th IEEE Conference on Computational Complexity (CCC)*, pages 32–43, 2014.

4 Johannes Bausch, Toby Cubitt, and Maris Ozols. The Complexity of Translationally Invariant Spin Chains with Low Local Dimension. *Annales Henri Poincaré*, 18(11):3449–3513, November 2017. doi:10.1007/s00023-017-0609-7.

5 Johannes Bausch, Toby S. Cubitt, Angelo Lucia, and David Perez-Garcia. Undecidability of the Spectral Gap in One Dimension. *Physical Review X*, 10(3):031038, August 2020. doi:10.1103/PhysRevX.10.031038.

6 Johannes Bausch and Stephen Piddock. The complexity of translationally invariant low-dimensional spin lattices in 3D. *Journal of Mathematical Physics*, 58(11):111901, November 2017. doi:10.1063/1.5011338.

7 H Bethe. Zur Theorie der Metalle. *Zeitschrift für Physik*, 71(3–4):205–226, 1931.

8 S. Bravyi and M. Hastings. On complexity of the quantum Ising model. *Communications in Mathematical Physics*, 349(1):1–45, 2017. doi:10.1007/s00220-016-2787-4.

9 Nikolas P. Breuckmann and Barbara M. Terhal. Space-time circuit-to-Hamiltonian construction and its applications. *Journal of Physics A Mathematical General*, 47(19):195304, May 2014. doi:10.1088/1751-8113/47/19/195304.

10 Brielin Brown, Steven T. Flammia, and Norbert Schuch. Computational Difficulty of Computing the Density of States. *PRL*, 107(4):040501, July 2011. doi:10.1103/PhysRevLett.107.040501.

11 Toby S Cubitt, Ashley Montanaro, and Stephen Piddock. Universal quantum hamiltonians. *Proceedings of the National Academy of Sciences*, 115(38):9497–9502, 2018. arXiv:1701.05182.

12 Toby S. Cubitt, David Perez-Garcia, and Michael M. Wolf. Undecidability of the spectral gap. *Nature*, 528(7581):207–211, December 2015. doi:10.1038/nature16059.

13 Bill Fefferman and Cedric Yen-Yu Lin. A Complete Characterization of Unitary Quantum Space. In *Leibniz International Proceedings in Informatics (LIPIcs)*. 9th Innovations in Theoretical Computer Science Conference, April 2016. doi:10.4230/LIPIcs.ITCS.2018.4.

14 Fabio Franchini. *An Introduction to Integrable Techniques for One-Dimensional Quantum Systems*, volume 940 of *Lecture Notes in Physics*. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-48487-7.

15 S. Gharibian, Y. Huang, Z. Landau, and S. W. Shin. Quantum Hamiltonian complexity. *Foundations and Trends® in Theoretical Computer Science*, 10(3):159–282, 2014. doi:10.1561/0400000066.

16 S. Gharibian and J. Kempe. Hardness of approximation for quantum problems. In *39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 387–398, 2012.

17 Sevag Gharibian, Stephen Piddock, and Justin Yirka. Oracle Complexity Classes and Local Measurements on Physical Hamiltonians. In *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*, volume 154, pages 20:1–20:37, 2020.

18 Sevag Gharibian and Jamie Sikora. Ground state connectivity of local Hamiltonians. In *42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 617–628, 2015.

19 Sevag Gharibian and Justin Yirka. The complexity of simulating local measurements on quantum systems. *Quantum*, 3:189, 2019. doi:10.22331/q-2019-09-30-189.

20 David Gosset, Jenish C. Mehta, and Thomas Vidick. QCMA hardness of ground space connectivity for commuting Hamiltonians. *Quantum*, 1:16, July 2017. doi:10.22331/q-2017-07-14-16.

21 Daniel Gottesman and Sandy Irani. The Quantum and Classical Complexity of Translationally Invariant Tiling and Hamiltonian Problems. *Theory of Computing*, 9(1):31–116, May 2009. doi:10.4086/toc.2013.v009a002.

22 S. Hallgren, D. Nagaj, and S. Narazanaswami. The Local Hamiltonian problem on a line with eight states is QMA-complete. *Quantum Information & Computation*, 13(9&10):0721–0750, 2013.

23 A. S. Holevo. Bounds for the quantity of information transmitted by a quantum communication channel. *Problems of Information Transmission*, 9(3):177–183, 1973.

24 Julia Kempe, Alexei Yu. Kitaev, and Oded Regev. The Complexity of the Local Hamiltonian Problem. *SIAM Journal on Computing*, 35(5):1070–1097, January 2006. doi:10.1137/S0097539704445226.

25 A. Kitaev, A. Shen, and M. Vyalyi. *Classical and Quantum Computation*. American Mathematical Society, 2002.

26 Alexei Yu. Kitaev, Alexander Shen, and Mikhail N. Vyalyi. Classical and quantum computing. In *Quantum Information*, pages 203–217. Springer New York, New York, NY, 2002. doi:10.1007/978-0-387-36944-0_13.

27 Tamara Kohler, Stephen Piddock, Johannes Bausch, and Toby Cubitt. Translationally invariant universal quantum hamiltonians in 1d. *Annales Henri Poincaré*, 23(1):223–254, 2021. doi:10.1007/s00023-021-01111-7.

28 Zeph Landau, Umesh Vazirani, and Thomas Vidick. A polynomial time algorithm for the ground state of one-dimensional gapped local Hamiltonians. *Nature Physics*, 11(7):566–569, June 2015. doi:10.1038/nphys3345.

29 Elliott Lieb, Theodore Schultz, and Daniel Mattis. Two soluble models of an antiferromagnetic chain. *Annals of Physics*, 16(3):407–466, December 1961. doi:10.1016/0003-4916(61)90115-4.

30 Daniel Nagaj. *Local Hamiltonians in Quantum Computation*. PhD thesis, Massachusetts Institute of Technology, Boston, 2008. Available at arXiv.org quant-ph/0808.2117v1.

31 Roberto I. Oliveira and Barbara M. Terhal. The complexity of quantum spin systems on a two-dimensional square lattice. *Quantum Information and Computation*, 8(10):1–23, April 2005. arXiv:0504050.

32 T. J. Osborne. Hamiltonian complexity. *Reports on Progress in Physics*, 75(2):022001, 2012. URL: http://stacks.iop.org/0034-4885/75/i=2/a=022001.

33 Stephen Piddock and Johannes Bausch. Universal Translationally-Invariant Hamiltonians. *arXiv*, January 2020. arXiv:2001.08050.

34 Y. Shi and S. Zhang. Note on quantum counting classes. Available at http://www.cse.cuhk.edu.hk/syzhang/papers/SharpBQP.pdf.

35 James D. Watson, Johannes Bausch, and Sevag Gharibian. The Complexity of Translationally Invariant Problems beyond Ground State Energies. *arXiv e-prints*, page arXiv:2012.12717, December 2020. arXiv:2012.12717.

36 Steven R. White. Density matrix formulation for quantum renormalization groups. *Physical Review Letters*, 69(19):2863–2866, November 1992. doi:10.1103/PhysRevLett.69.2863.

A Additional Definitions

▶ **Definition 20** ($\mathsf{QMA_{EXP}}$ [21]). *A promise problem $\Pi = (A_{\text{yes}}, A_{\text{no}})$ is in $\mathrm{QMA_{EXP}}$ if and only if there exists a $k \in \mathrm{O}(1)$ and a Quantum Turing Machine M such that for any input $x \in \{0,1\}^n$, and any proof $|\psi\rangle \in (\mathbb{C}^2)^{\otimes 2^{n^k}}$, on input $(x, |\psi\rangle)$, M halts in 2^{n^k} steps. Furthermore,*

- *(Completeness) If $x \in A_{\text{yes}}$, $\exists\ |\psi\rangle \in (\mathbb{C}^2)^{\otimes 2^{n^k}}$ such that M accepts $(x, |\psi\rangle)$ with probability $\geq 2/3$.*
- *(Soundness) If $x \in A_{\text{no}}$, then $\forall\ |\psi\rangle \in (\mathbb{C}^2)^{\otimes 2^{n^k}}$, M accepts $(x, |\psi\rangle)$ with probability $\leq 1/3$.*

We take care to distinguish $\mathrm{QMA_{EXP}}$ from the class $\mathsf{QMA_{exp}}$ of [13] which is for an exponentially small promise gap in the input size, but polynomial length run time, also called PreciseQMA.

Next, QMA with a classical witness yields the complexity class QCMA.

▶ **Definition 21** ($\mathsf{QCMA}_{\mathsf{EXP}}$). *A promise problem $\Pi = (A_{\text{yes}}, A_{\text{no}})$ is in* $\mathrm{QCMA}_{\mathrm{EXP}}$ *if and only if there exists $k \in \mathrm{O}(1)$ and an exponential-time uniform family of quantum circuits $\{Q_n\}$, where Q_n takes as input a string $x \in \Sigma^n$, a classical proof $y \in \{0,1\}^{2^{n^k}}$, and 2^{n^k} ancilla qubits in state $|0\rangle^{\otimes 2^{n^k}}$, such that:*

- *(Completeness) If $x \in A_{\text{yes}}$, $\exists y \in \{0,1\}^{2^{n^k}}$ such that Q_n accepts (x, y) with probability $\geq 2/3$.*
- *(Soundness) If $x \in A_{\text{no}}$,$\forall y \in \{0,1\}^{2^{n^k}}$, Q_n accepts (x, y) with probability $\leq 1/3$.*

▶ **Definition 22** (Deterministic Decision class). *A set C of languages is a* deterministic decision class *if, for any language $L \in C$, there exists a deterministic Turing machine M which can decide L under the resource constraints specified by C. Formally, given any input $x \in \{0,1\}^n$, M halts after using $R(n)$ resources (where R may specify bounds on time or space), and accepts if $x \in L$ or rejects if $x \notin L$.*

Standard examples of deterministic classes include P, PSPACE, and EXP.

▶ **Definition 23** (Existentially quantified quantum verification class (QVClass)). *A set C of promise problems is an* existentially quantified quantum verification class *if any promise problem $A = (A_{\text{yes}}, A_{\text{no}}, A_{\text{inv}})$ in C satisfies the following. There exist computable functions $f, g, h : \mathbb{N} \mapsto \mathbb{N}$, as well as a deterministic Turing machine M which, for any input $x \in \{0,1\}^n$, uses $R(n)$ resources to produce a quantum verification circuit V (consisting of 1- and 2-qubit gates) and thresholds $c, s \in \mathbb{R}^+$ such that $c - s > 1/h(n)$. Here, $R(x)$ refers to resources such as time, space, etc, as required by C. The circuit V takes in a quantum proof $|\psi\rangle$ on $f(n)$ qubits, $g(n)$ ancilla qubits initialised to all zeroes, and has a designated output qubit, such that:*

- *(YES case) If $x \in A_{\text{yes}}$, there exists a quantum proof $|\psi\rangle$ on $f(n)$ qubits such that measuring the output qubit of $V|\psi\rangle|0\cdots0\rangle$ in the standard basis yields 1 with probability at least c.*
- *(NO case) If $x \in A_{\text{no}}$, for all quantum proofs $|\psi\rangle$ on $f(n)$ qubits, measuring the output qubit of $V|\psi\rangle|0\cdots0\rangle$ in the standard basis yields 1 with probability at most s.*

Without loss of generality, we assume the output qubit of V is the first wire exiting V.

In this way, classes such as NP, NEXP, QCMA, QMA, and so forth are examples of a QVClass.

▶ **Definition 24** ($\forall$-APX-SIM$(H, A, k, l, a, b, \delta)$ [17]). *Given a k-local Hamiltonian $H = \sum_i H_i$ acting on N qubits, an l-local observable A, and real numbers a, b, and δ such that $b - a \geq N^{-c}$ and $\delta \geq N^{-c'}$, for $c, c' > 0$ constant, decide:*
YES. *If for all $|\psi\rangle$ satisfying $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H) + \delta$, it holds that $\langle\psi|A|\psi\rangle \leq a$.*
NO. *If for all $|\psi\rangle$ satisfying $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H) + \delta$, it holds that $\langle\psi|A|\psi\rangle \geq b$.*

B Complexity of APX-SIM

B.1 Useful Lemmas

▶ **Lemma 25** (Extended Projection Lemma ([24, 19])). *Let $H = H_1 + H_2$ be the sum of two Hamiltonians operating on some Hilbert space $\mathcal{H} = \mathcal{S} + \mathcal{S}^\perp$. The Hamiltonian H_1 is such that $\mathcal{S}$ is a zero eigenspace and the eigenvectors in $\mathcal{S}^\perp$ have eigenvalue at least $J > 2\|H_2\|_\infty$. Let $K \coloneqq \|H_2\|_\infty$. Then, for any $\delta \geq 0$ and $|\psi\rangle$ satisfying $\langle\psi|H|\psi\rangle \leq \lambda_{\min}(H) + \delta$, there exists a $|\psi'\rangle \in \mathcal{S}$ such that the ground state energy is bounded as $\lambda_{\min}(H_2|_{\mathcal{S}}) - \frac{K^2}{J-2K} \leq$*

$\lambda_{\min}(H) \leq \lambda_{\min}(H_2|_{\mathcal{S}})$, where $\lambda_{\min}(H_2|_{\mathcal{S}})$ denotes the smallest eigenvalue of H_2 restricted to space $\mathcal{S}$. Furthermore, the ground state is perturbed as $|\langle\psi|\psi'\rangle|^2 \geq 1 - \left(\frac{K+\sqrt{K^2+\delta(J-2K)}}{J-2K}\right)^2$, and satisfies $\langle\psi'|H|\psi'\rangle \leq \lambda_{\min}(H) + \delta + 2K\frac{K+\sqrt{K^2+\delta(J-2K)}}{J-2K}$.

C Complexity of GSCON

C.1 Definitions for GSCON

▶ **Definition 26** (TI-standard). *A circuit-to-Hamiltonian mapping from verification circuits V to 1D, nearest neighbor, translationally invariant Hamiltonians $H = \sum_{i=1}^{N-1} H_{i,i+1}$ is* TI-standard *if it satisfies the following conditions. Below, N denotes the number of qudits H acts on, and α and β the completeness/soundness (a.k.a. "low energy" and "high energy") parameters for H, respectively:*

1. *$H \succeq 0$, although the local terms may satisfy $H_{i,i+1} \not\succeq 0$.*
2. *In the YES case, if the optimal proof to verifier V is a classical string $y \in \{0,1\}^{\mathrm{poly}(N)}$, then there exists a (potentially non-uniformly generated) quantum circuit of size $L \in \mathrm{poly}(N)$ preparing a low energy state $|\psi_{\mathrm{low}}\rangle$ for H, i.e. $\langle\psi_{\mathrm{low}}|H|\psi_{\mathrm{low}}\rangle \leq \alpha$.*
3. *In the YES case, the subset of indices $F \subseteq [N-1]$ for which $H_{i,i+1}$ contributes negative energy to the low-energy state, i.e. all i for which $H_{i,i+1}$ satisfies $\langle\psi_{\mathrm{low}}|H_{i,i+1}|\psi_{\mathrm{low}}\rangle < 0$, is computable in $\mathrm{poly}(N)$ time*[4].
4. *In the NO case, $\lambda_{\min}(H) \geq \beta$. Here, we require $|\alpha - \beta| \geq 1/\mathrm{poly}(N)$ (which is standard in the literature) and $\beta \geq 16(2L+7N)\alpha \geq 0$ (which is specific to our construction).*

All of these assumptions are rather mild, as we now clarify.

Remarks regarding Definition 26.

- Assumptions 1 and 4 must be taken together (otherwise, $H \succeq 0$ can always be achieved by adding a multiple of the identity).
- The setting of $H \succeq 0$ but $H_i \not\succeq 0$ arises when applying our construction to the Gottesman-Irani 1D TI mapping (henceforth GI) [21] in Section 3.2. Specifically, GI is not TI-standard in its original form, since it violates the final requirement $\beta \geq 16(2L+7N)\alpha$, which is crucial to our use of the Traversal Lemma.
- Assumption 3 is vacuously true when all $H_i \succeq 0$. When $H_i \not\succeq 0$ for some i, however, this is also generally a mild assumption, since it only cares about energies against $|\psi_{\mathrm{low}}\rangle$, which is typically a history state of some form.

C.2 Proof of GSCON Lifting Lemma

We now prove the Lifting Lemma for GSCON, Lemma 15.

C.2.1 Completeness

Suppose x is a YES instance. The following lemma shows there is a short path through the low energy subspace between states $|\psi\rangle$ and $|\phi\rangle$, as desired.

[4] One can replace the < 0 condition here with $< -1/p(N)$ for some sufficiently small polynomial p; we omit this for simplicity.

▶ **Lemma 27.** *Using the notation of Definition 16, let* $\lambda_{\min}(H') \leq \alpha$. *There exists a circuit* $U = U_m \dots U_2 U_1$ *of 2-local gates* U_i *such that* $U|\psi\rangle = |\phi\rangle$, *and all intermediate states* $|\psi_i\rangle = U_i \dots U_2 U_1 |\psi\rangle$ *satisfy* $\langle\psi_i| H |\psi_i\rangle \leq \eta_1$.

Proof. By definition of TI-standard, since the optimal $\mathrm{QCMA}_{\mathrm{EXP}}$ proof is a classical string of size $\mathrm{poly}(N)$, there exists a $\mathrm{poly}(N)$-length sequence $U' = U_L \cdots U_1$ of L 1-and 2-qudit unitaries (acting on $\bigotimes_{i=1}^N \mathcal{A}_i$) which prepares a low energy state $|\psi_{\mathrm{low}}\rangle$ of H'. The circuit U of the claim now acts as follows (see Figure 2 for an explicit example when $N = 4$).

1. *Prepare low energy state.* Compute $U'_{\mathcal{A}} \otimes I_{\mathcal{B}} |\psi\rangle_{\mathcal{A},\mathcal{B}}$, i.e. perform the mapping
$$|\psi\rangle_{\mathcal{A},\mathcal{B}} \mapsto |\psi_{\mathrm{low}}\rangle_{\mathcal{A}} |0 \cdots 0\rangle_{\mathcal{B}}.$$
2. *"Warm up".* Let $F \subseteq [N-1]$ denote the set of indices i for which $\langle\psi_{\mathrm{low}}| H'_{i,i+1} |\psi_{\mathrm{low}}\rangle < 0$, which is efficiently computable by definition of TI-standard. One at a time, map $|0\rangle_{\mathcal{B}_i} \mapsto |1\rangle_{\mathcal{B}_i}$ for each $i \in F$, in any order.
3. *"Full blast".* One at a time, map $|0\rangle_{\mathcal{B}_i} \mapsto |1\rangle_{\mathcal{B}_i}$ for all $i \in [N] \setminus F$, in any order.
4. *"Left deke".* Map $|1\rangle_{\mathcal{B}_i} \mapsto |2\rangle_{\mathcal{B}_i}$ for all i in sequence $(N, \dots, 1)$ (i.e. right to left).
5. *"Right deke".* Map $|2\rangle_{\mathcal{B}_i} \mapsto |3\rangle_{\mathcal{B}_i}$ for all i in sequence $(1, \dots, N)$ (i.e. left to right).
6. *"Left deke".* Map $|3\rangle_{\mathcal{B}_i} \mapsto |4\rangle_{\mathcal{B}_i}$ for all i in sequence $(N, \dots, 1)$ (i.e. right to left).
7. *"Right deke".* Map $|4\rangle_{\mathcal{B}_i} \mapsto |5\rangle_{\mathcal{B}_i}$ for all i in sequence $(1, \dots, N)$ (i.e. left to right).
8. *"Cool down".* One at a time, map $|5\rangle_{\mathcal{B}_i} \mapsto |6\rangle_{\mathcal{B}_i}$ for all $i \in [N] \setminus F$, in any order.
9. *"Complete shut down".* One at a time, map $|5\rangle_{\mathcal{B}_i} \mapsto |6\rangle_{\mathcal{B}_i}$ for each $i \in F$, in any order.
10. *Uncompute low energy state.* Apply $(U')^\dagger_{\mathcal{A}} \otimes I_{\mathcal{B}}$ to our state.

Analysing each step above shows that for each step satisfy $\langle\psi_i| H |\psi_i\rangle \leq \eta_1$. ◀

C.2.2 Soundness

Suppose x is a NO instance. The following lemma shows that any short path from $|\psi\rangle$ to $|\phi\rangle$ must leave the low-energy subspace, as desired.

▶ **Lemma 28.** *Using the notation of Definition 16, let* $\lambda_{\min}(H') \geq \beta$, *and fix any* $b \in \{2, \dots, N-1\}$. *Consider any sequence* $U = U_m \cdots U_1$ *of b-local unitary operators acting on* $\bigotimes_{i=1}^N \mathcal{A}_i \otimes \mathcal{B}_i$. *Then, either there exists* $i \in [m]$ *such that intermediate state* $|\psi_i\rangle := U_i \cdots U_2 U_1 |\psi\rangle$ *satisfies* $\langle\psi_i| H |\psi_i\rangle \geq \frac{1}{2}\left(\frac{\beta}{4m^2}\right) = \eta_2$, *or* $\| |\psi_m\rangle - |\phi\rangle \|_2 \geq 1/2 = \eta_4$.

Proof. Assume, for sake of contradiction, that $\| |\psi_m\rangle - |\phi\rangle \|_2 < 1/2$, and that $\langle\psi_i| H |\psi_i\rangle < \eta_2$ for all $i \in [m]$. Define b-orthogonal subspaces

$$\begin{aligned} S_{012} &= I_{\mathcal{A},\mathcal{B}_{1,\dots,N-b-1}} \otimes \mathrm{Span}\left(\left\{|s\rangle_{\mathcal{B}_{N-b,N}} \mid s \in \{0,1,2\}^{b+1}\right\}\right) \\ S_{456} &= I_{\mathcal{A},\mathcal{B}_{1,\dots,N-b-1}} \otimes \mathrm{Span}\left(\left\{|s\rangle_{\mathcal{B}_{N-b,N}} \mid s \in \{4,5,6\}^{b+1}\right\}\right), \end{aligned}$$

and recall that we set

$$H_{i,i+1} := H'_{i,i+1} \otimes (|1\rangle\langle 1| + |2\rangle\langle 2| + |3\rangle\langle 3| + |4\rangle\langle 4| + |5\rangle\langle 5|)_{\mathcal{B}_i} + \Delta E_{\mathcal{B}_i,\mathcal{B}_{i+1}} \tag{9}$$

for E the projector forbidding the 2-local substrings depicted in Figure 2 (left). We first show that, for sufficiently large Δ, $|\psi_i\rangle$ has almost all its amplitude on a state in the null space of $H_E := \Delta \sum_{i=1}^N E_{\mathcal{B}_i,\mathcal{B}_{i+1}}$.

▶ **Lemma 29.** *Assume* $\langle\psi_i| H |\psi_i\rangle < \eta_2$ *for all* $i \in [m]$*. Then, there exists* $i \in [m]$ *such that* $|\psi_i\rangle$ *can be written as* $|\psi_i\rangle = \gamma_1 |\gamma_1\rangle + \gamma_2 |\gamma_2\rangle$*, with* $\langle\gamma_1|\gamma_2\rangle = 0$*,* $|\gamma_1\rangle \in \mathrm{Null}\,(H_E)$*, and*

$$\left\| |\psi_i\rangle\langle\psi_i| - |\gamma_1\rangle\langle\gamma_1| \right\|_{\mathrm{tr}} < 2\sqrt{\frac{\eta_2}{\Delta}}, \tag{10}$$

$$\langle\gamma_1| I - \Pi_{S_{012}} - \Pi_{S_{456}} |\gamma_1\rangle > \frac{1}{4m^2} - 2\sqrt{\frac{\eta_2}{\Delta}}. \tag{11}$$

The proof of Lemma 29 follows from Lemma 14. We draw the following conclusions:

1. Recalling that S_{012} (S_{456}) projects onto $\{0,1,2\}^*$ ($\{4,5,6\}^*$) in $\mathcal{B}_{N-b,\ldots,N}$, respectively, Equation (11) implies $|\gamma_1\rangle = \chi_1 |\chi_1\rangle + \chi_2 |\chi_2\rangle$ for orthonormal vectors $\{|\chi_1\rangle, |\chi_2\rangle\}$, $|\chi_1|^2 > (4m^{-2}) - 2\sqrt{\eta_2/\Delta}$, such that $|\chi_1\rangle$ has registers $\mathcal{B}_{N-b,\ldots,N}$ supported solely on the intersection of two sets:
   - All strings in the null space of H_E (since $|\gamma_1\rangle \in \mathrm{Null}\,(H_E)$ implies $|\chi_1\rangle \in \mathrm{Null}\,(H_E)$), and
   - all strings in the null space of $I - \Pi_{S_{012}} - \Pi_{S_{456}}$ (by Equation (11)).

   But the intersection of these two sets has precisely the regular expression

   $$33^*(2^* \cup 4^*), \tag{12}$$

   where the first 3 is located in $\mathcal{B}_{N-b}$. This follows since by Figure 2 (left), a 3 can only have a 2, 3, or 4 to its right, and once we put down a 2 to the right (resp. 4), we can only put down more 2's (resp. 4's).

   Similarly, $|\chi_2\rangle$ is supported in registers $\mathcal{B}_{N-b,\ldots,N}$ solely on the span of strings from set $\{0,1,2\}^{b+1} \cup \{4,5,6\}^{b+1}$ (note Figure 2 (left) disallows a digit from set $\{0,1,2\}$ to be neighbours with a digit from $\{4,5,6\}$). Thus, $|\chi_1\rangle$ and $|\chi_2\rangle$ are orthogonal on the last $b+1$ switch qudits (since the former must have a $|3\rangle$ on these qudits, but the latter cannot).
2. Again since $|\chi_1\rangle \in \mathrm{Null}\,(H_E)$, combining Equation (12) with Figure 2 (left) now implies, in fact, that *all* switch qudits "to the left" of $\mathcal{B}_{N-b}$ are also set to $|3\rangle$, i.e.

   $$|\chi_1\rangle = |3\cdots3\rangle_{\mathcal{B}_{1,\ldots,N-b}} \otimes |\chi_1'\rangle_{\mathcal{A},\mathcal{B}_{N-b+1,\ldots,N}},$$

   for some unit vector $|\chi_1'\rangle$. Together with Equation (12), this implies the entire register $\mathcal{B}$ of $|\chi_1\rangle$ is supported only on symbols from $\{2,3,4\}$.
3. Since all of $\mathcal{B}$ is now supported on symbols from $\{2,3,4\}$, it follows from Equation (9) that *all* terms of H' are switched on (thus resolving Obstacle 2). Hence,

   $$\begin{aligned}
   \langle\gamma_1| H |\gamma_1\rangle &= \langle\gamma_1| \sum_{i=1}^{N-1} H'_{i,i+1} \otimes (|1\rangle\langle1| + |2\rangle\langle2| + |3\rangle\langle3| + |4\rangle\langle4| + |5\rangle\langle5|)_{\mathcal{B}_i} |\gamma_1\rangle \\
   &> \left(\frac{1}{4m^2} - 2\sqrt{\frac{\eta_2}{\Delta}}\right) \langle\chi_1| H |\chi_1\rangle \\
   &= \left(\frac{1}{4m^2} - 2\sqrt{\frac{\eta_2}{\Delta}}\right) \mathrm{Tr}(H' \,\mathrm{Tr}_{\mathcal{B}}(|\chi_1\rangle\langle\chi_1|)) \\
   &\geq \left(\frac{1}{4m^2} - 2\sqrt{\frac{\eta_2}{\Delta}}\right) \beta,
   \end{aligned} \tag{13}$$

   where the first statement follows since $|\gamma_1\rangle \in \mathrm{Null}\,(H_E)$, the second since (1) $H' \succeq 0$ and (2) since

   $$\langle\chi_1| H'_{i,i+1} \otimes (|1\rangle\langle1| + |2\rangle\langle2| + |3\rangle\langle3| + |4\rangle\langle4| + |5\rangle\langle5|)_{\mathcal{B}_i} |\chi_2\rangle = 0,$$

since $|\chi_1\rangle$ and $|\chi_2\rangle$ are orthogonal on the last $b+1$ switch qudits (even when projected down onto $\mathrm{Span}(|1\rangle, |2\rangle, |3\rangle, |4\rangle, |5\rangle)$), the third statement since all of register $\mathcal{B}$ is supported on symbols $\{2,3,4\}$, and the last statement since $\lambda_{\min}(H') \geq \beta$ by assumption.

We conclude that $|\gamma_1\rangle$ is high energy against H. We now show a similar result for $|\psi_i\rangle$, giving the desired contradiction. To do so, we follow the proof of the Projection Lemma of [24]. For brevity, define $H_1 \coloneqq \sum_{i=1}^{N-1} H'_{i,i+1} \otimes (|1\rangle\langle 1| + |2\rangle\langle 2| + |3\rangle\langle 3| + |4\rangle\langle 4| + |5\rangle\langle 5|)_{\mathcal{B}_i}$ so that $H = H_1 + H_E$. Then, for $\Delta > 2\left\|H'\right\|_\infty = 2\left\|H_1\right\|_\infty$, recalling that $H_E|\gamma_1\rangle = 0$,

$$\begin{aligned}
\langle\psi_i| H |\psi_i\rangle &\geq \left[(1-|\gamma_2|^2)\langle\gamma_1| H_1 |\gamma_1\rangle + 2\mathrm{Re}(\gamma_1\gamma_2\langle\gamma_1| H_1 |\gamma_2\rangle) + |\gamma_2|^2\langle\gamma_2| H_1 |\gamma_2\rangle\right] + \Delta|\gamma_2|^2 \\
&\geq \langle\gamma_1| H_1 |\gamma_1\rangle + (\Delta - 2\left\|H_1\right\|_\infty)|\gamma_2|^2 - 2\left\|H_1\right\|_\infty|\gamma_2| \\
&> \langle\gamma_1| H |\gamma_1\rangle - 2\left\|H_1\right\|_\infty\sqrt{\frac{\eta_2}{\Delta}} \\
&> \frac{1}{4m^2}\beta - 2\sqrt{\frac{\eta_2}{\Delta}}\left(\beta + \left\|H'\right\|_\infty\right),
\end{aligned}$$

where the first statement follows since $|\gamma_1|^2 + |\gamma_2|^2 = 1$ and $H_E|\gamma_1\rangle = 0$, the second since $|\gamma_1| \leq 1$, the third when $\Delta > 2\left\|H_1\right\|_\infty$ and since $|\gamma_2|^2 < \eta_2/\Delta$, and the last by Equation (13) and since $\left\|H_1\right\|_\infty = \left\|H'\right\|_\infty$. Crucially, note that H' is independent of Δ (recall H' is the TI-standard Hamiltonian we have plugged in as a black-box). Thus, we may set Δ to a sufficiently large fixed polynomial in N so that $\langle\psi_i| H |\psi_i\rangle > \frac{1}{2}\left(\frac{\beta}{4m^2}\right) = \eta_2$. This yields the desired contradiction. ◀

Restless Temporal Path Parameterized Above Lower Bounds

Philipp Zschoche
Faculty IV, Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

Reachability questions are one of the most fundamental algorithmic primitives in temporal graphs – graphs whose edge set changes over discrete time steps. A core problem here is the NP-hard Short Restless Temporal Path: given a temporal graph $\mathcal{G}$, two distinct vertices s and z, and two numbers δ and k, is there a δ-restless temporal s-z path of length at most k? A temporal path is a path whose edges appear in chronological order and a temporal path is δ-restless if two consecutive path edges appear at most δ time steps apart from each other. Among others, this problem has applications in neuroscience and epidemiology. While Short Restless Temporal Path is known to be computationally hard, e.g., it is NP-hard even if the temporal graph consists of three discrete time steps and it is W[1]-hard when parameterized by the feedback vertex number of the underlying graph, it is fixed-parameter tractable when parameterized by the path length k. We improve on this by showing that Short Restless Temporal Path can be solved in (randomized) $4^{k-d}|\mathcal{G}|^{O(1)}$ time, where d is the minimum length of a temporal s-z path.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases temporal graphs, FPT, above-lower-bound parameterization

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.55

Acknowledgements I am grateful to Rolf Niedermeier, Till Fluschnik, and anonymous reviewers for constructive and detailed feedback.

1 Introduction

Susceptible-Infected-Recovered. These are the three states of the *SIR-model* – a canonical spreading model for diseases where recovery confers lasting resistance [6, 31, 39]. Here, an individual is at first susceptible (S) to get a certain disease, can devolve to be infected (I), and ends up resilient after recovery (R). We study one of the most fundamental algorithmic questions in this model: given a set of individuals with a list of physical contacts over time, and two individuals s and z, is it possible to have a chain of infections from s to z? As the timing of the physical contacts is crucial in this scenario, we use a *temporal graph* $\mathcal{G} := (V, (E_i)_{i=1}^{\tau})$ consisting of a set V of vertices and an edge set that changes over discrete time steps described by a chronologically ordered sequence $(E_i)_{i=1}^{\tau}$ of edge sets over V. Here, the number τ denotes the *lifetime* of of the temporal graph $\mathcal{G}$. A temporal path is a path whose edges appear in chronological order. In particular, a sequence $P := ((e_i, t_i))_{i=1}^{m}$ of time-edges from $\mathcal{E}(\mathcal{G}) := \bigcup_{i=1}^{\tau} E_i \times \{i\}$ is a *temporal s-z path* of length m if $(\bigcup_{i=1}^{m} e_i, \{e_i \mid i \in [m]\})$ is an s-z path (no vertex is visited twice) and $t_i \leq t_{i+1}$ for all $i \in [m-1]$. If we construct a temporal graph where the vertices are individuals and an edge $e \in E_t$ represents a physical contact of two individuals at time step t, then a chain of infections is represented by a temporal path. However, not every temporal path yields a potential chain of infections, as an infected person might recover before the next individual is met. To represent infection chains in the SIR-model by temporal paths, we restrict the waiting time at each intermediate vertex to a prescribed duration – that is, the time until an individual becomes resilient after infection. These temporal paths are called restless. In particular, the temporal s-z path P is *δ-restless* if $t_i \leq t_{i+1} \leq t_i + \delta$ for all $i \in [m-1]$ where $\delta \in \mathbb{N} \cup \{0\}$. Hence, restless temporal paths model

© Philipp Zschoche;
licensed under Creative Commons License CC-BY 4.0

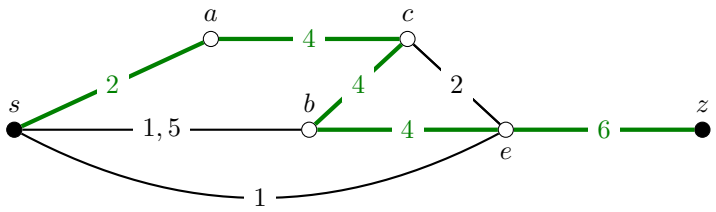


Figure 1 An illustration of a temporal graph with vertices s, a, b, c, e, and z. The labels on the edges denote at which time steps the edges are present. The time-edges of a 2-restless temporal s-z path in this temporal graph are marked by thick (green) edges. In fact, this is the only 2-restless temporal s-z path in this temporal graph, as we cannot visit a vertex twice and two consecutive time-edges have to be at most two time steps apart.

infection transmission routes of diseases that grant immunity upon recovery [28]. Other applications of restless temporal paths appear in the context of delay-tolerant networking with time-aware routing tables [13], and in the context of finding signaling pathways in brain networks [41]. Consider Figure 1 for an illustration of a temporal graph with a 2-restless temporal s-z path.

The central problem of this work is as follows.

Short Restless Temporal Path
Input: A temporal graph $\mathcal{G}$, a source vertex $s \in V$, a destination vertex $z \in V$, and integers $\delta, k \in \mathbb{N}$.
Question: Is there a δ-restless temporal s-z path in $\mathcal{G}$ of length at most k?

Casteigts et al. [13] showed that Short Restless Temporal Path is NP-hard even if $\delta = 1$, $\tau = 3$, every edge appears only once, and the underlying graph has a maximum degree of six. Moreover, they showed that it is W[1]-hard when parameterized by the distance to disjoint paths of the underlying graph[1]. Among other things, this presumably excludes exact algorithms for Short Restless Temporal Path running in $f(x) \cdot |\mathcal{G}|^{O(1)}$ time, where x is the distance to disjoint paths of the underlying graph and f is any computable function.

Hence, Short Restless Temporal Path is presumably not fixed-parameter tractable when parameterized by a wide range of well-known parameters of the underlying (static) graph, e.g., feedback vertex number, pathwidth, or cliquewith. However, Short Restless Temporal Path is fixed-parameter tractable when parameterized by k or the treedepth of the underlying graph or the feedback edge number of the underlying graph [13]. Thejaswi et al. [41] showed that for every $p \in \mathbb{R}$ with $0 < p < 1$ there is a randomized $O(2^k k|\mathcal{G}|\delta \log(k \cdot 1/p))$-time algorithm for Short Restless Temporal Path that has a one-sided error probability of at most p. More precisely, if the algorithm returns *yes*, then the given instance I of Short Restless Temporal Path is a *yes*-instance, and if the algorithm returns *no*, then the probability that I is a *yes*-instance is at most p. They conducted experiments on large synthetic and real-world data sets and showed that their algorithm performs well as long as the parameter k is small. For example, one can solve Short Restless Temporal Path with $k \leq 9$ and a temporal graph with 36 million time-edges in less than one hour with customary desktop hardware. On the data set used in the experiments, k seems to be the

[1] That is, the minimum number of vertices we need to remove from a graph such that the remaining graph consists of a set of vertex-disjoint paths.

only useful parameter for which we know that Short Restless Temporal Path is fixed-parameter tractable; all other known parameters (i.e., timed feedback vertex number [13], treedepth of the underlying graph, and feedback edge number of the underlying graph) are too large to be eligible in practice [41]. Hence, the current algorithms are not satisfactory when it comes to computing long restless temporal paths in real-world temporal networks.

The parameter k of Short Restless Temporal Path can be seen as the *solution size* and is thus a natural and well-motivated parameter from a parameterized algorithmics point of view. However, as we observed before, FPT-algorithms regarding the solution size are not necessarily practical, e.g., if all solutions are large. To address this problem, one can investigate *parameterizations above guaranteed lower bounds* [4, 8, 14, 25, 27, 33, 34, 29]: that is, the *difference* between the smallest size of a solution and a guaranteed lower bound for the solution size. We refer to Gutin and Mnich [26] for a survey on graph problems parameterized above and below guarantees.

In the case of Short Restless Temporal Path, we are looking for a guaranteed lower bound d on the length of any δ-restless temporal s-z path such that Short Restless Temporal Path admits an FPT-algorithm when parameterized by $k - d$. Note that if d exceeds k, then this is a *no*-instance. Towards the choice of a lower bound, we see the following three intermediate options.

The distance from s to z: The minimum length of an s-z path in the underlying graph. This is lower bound for the length of any δ-restless temporal s-z path, as each δ-restless temporal s-z path induces an s-z path in the underlying graph.

The temporal distance from s to z: The minimum length of a temporal s-z path. Clearly, this is a lower bound for the length of any δ-restless temporal s-z path. Note that, the temporal distance from s to z is at least the distance from s to z in the underlying graph. Hence, the parameter $k - d_t$ is *at least as good* as the parameter $k - d_u$ for Short Restless Temporal Path, where d_u is the distance between s and z and d_t is the temporal distance between s and z.

The δ-restless temporal distance from s to z: The minimum length of a δ-restless temporal s-z walk. Herein, a sequence $W := ((e_i, t_i))_{i=1}^m$ of time-edges is a *temporal s-z walk* of length m if the edges $(e_i)_{i=1}^m$ induce an s-z walk and $t_i \leq t_{i+1}$ for all $i \in [m-1]$. Moreover, W is δ-restless if $m = 1$ or $t_{i+1} - t_i \leq \delta$. As any δ-restless temporal s-z path is also a δ-restless temporal s-z walk, the minimum length of a δ-restless temporal s-z walk is a lower bound for the length of any δ-restless temporal s-z path. Moreover, this lower bound is at least as good as the temporal distance from s to z.

For the sake of brevity, we say for an instance $(\mathcal{G}, s, z, \delta, k)$ of Short Restless Temporal Path that the δ-restless temporal distance from s to z, the temporal distance from s to z, and the distance from s to z are always below $k + 1$, as otherwise $(\mathcal{G}, s, z, \delta, k)$ is a trivial *no*-instance.

Unfortunately, a closer look at the NP-hardness reductions of Casteigts et al. [13] reveals that, unless P=NP, there is no $|\mathcal{G}|^{f(k-d_r)}$-time algorithm for Short Restless Temporal Path, where d_r is the δ-restless temporal distance from s to z and f is any computable function. However, we show that one can determine in polynomial time whether there is a δ-restless temporal s-z path which has the same length as a shortest temporal s-z path. Moreover, we show that Short Restless Temporal Path admits an FPT-algorithms when parameterized above the (temporal) distance from s to z.

Our contributions. We show that Short Restless Temporal Path can be solved in randomized $4^{k-d}|\mathcal{G}|^{O(1)}$ time, where d is the temporal distance from s to z. To the best of our knowledge, this is the first above-lower-bound FPT-algorithm on temporal graphs.

More precisely, we show that for every $p \in \mathbb{R}$ with $0 < p < 1$ there is a randomized $O(4^\ell \cdot \ell^2|\mathcal{G}|^3\delta \log(k/p(\ell+1)))$-time algorithm for Short Restless Temporal Path with a one-sided error probability of at most p, where $\ell := k - d$ and d is the temporal distance from s to z. Informally speaking, the parameter ℓ is minimum length of the detour a δ-restless temporal s-z path has to take compared to a (non-δ-restless) temporal s-z path. The main technical contribution behind this is a geometrical perspective onto temporal graphs which seems applicable to other temporal graph problems when parameterized above the temporal distance between vertices. In the resulting algorithm, the only subroutine with a super-polynomial running time is the algorithm of Thejaswi et al. [41] that we employ to find δ-restless temporal path of length at most $2(k - d) + 1$. In fact, this subroutine can be replaced by a deterministic algorithm of Casteigts et al. [13] – this leads to a $2^{O(k-d)}|\mathcal{G}|^3\delta$-time deterministic algorithm for Short Restless Temporal Path. The running time overhead induced by our technique is $O(|\mathcal{G}|^2\ell)$ in the deterministic case and $O(|\mathcal{G}|^2\ell \log(k/(\ell+1)p))$ if we use the algorithm of Thejaswi et al. [41], where $\ell := k - d$ and d is the temporal distance from s to z. The overhead with the randomized algorithm is larger as we need that the error probability of several calls of the randomized algorithm accumulate to p. Even though the running time overhead of our technique is slightly larger with the randomized algorithm of Thejaswi et al. [41], the exponential part of running time of the overall algorithm for finding restless temporal paths is better than with the algorithm of Casteigts et al. [13].

Further related work. In the literature, waiting-time constraints are studied from various angles. Himmel et al. [7] studied a variant of restless temporal paths where multiple visits of vertices are permitted, i.e., restless temporal *walks*. In contrast to restless temporal paths, they showed that such walks can be computed in polynomial time. Pan and Saramäki [40] empirically studied the correlation between waiting times of temporal paths and the ratio of the network reached in spreading processes. Akrida et al. [1] studied flows in temporal networks with "vertex buffers", which however pertains to the quantity of information that a vertex can store, rather than a duration.

Algorithmic reachability questions are one of the most thriving research topics in temporal graphs. Bui-Xuan et al. [11] and Wu et al. [42] studied the computation of temporal paths that satisfy certain optimality criteria and show that shortest, fastest, and foremost temporal path can be computed in polynomial time. In the temporal setting, reachability is not an equivalence relation among vertices and the reachability relation between vertices is not even transitive – this makes many problems computationally harder than their counterpart on static graphs. Michail and Spirakis [36] studied the NP-hard question of whether a temporal graph contains a temporal walk that visits each vertex at least once. This problem remains computationally hard even if the underlying graph is a star [3, 12]. If the underlying graph is connected at each time step and the walk can only contain one edge in each time step, then a fast exploration is guaranteed [20, 22, 21]. However, on these so-called always-connected temporal graphs, the decision problem remains NP-hard, even if the underlying graph has pathwidth two [10]. Kempe et al. [30] studied whether there are k vertex-disjoint temporal paths between two given vertices. While the classical analogue of this on static graphs is polynomial-time solvable, it becomes NP-hard in the temporal setting. Moreover, this problem remains NP-hard on a single underlying path, when we are looking for a set of temporal paths which is only pairwise vertex-disjoint at any point in time [32]. Furthermore, the related problem of finding small separators in temporal graphs becomes computationally hard [23, 30], even on quite restricted temporal graph classes [23]. Bhadra and Ferreira [9] showed that finding a maximum temporally connected component is NP-hard. Furthermore, a temporal graph may not have a sparse spanner [5], and computing a spanner with a minimum number of time-edges is NP-hard [2, 35].

Related to spreading processes, Enright et al. [18, 19], Deligkas and Potapov [16], and Molter et al. [38] studied restricting the set of reachable vertices via various temporal graph modifications – all described decision problems are NP-hard in rather restricted settings.

2 Preliminaries

We denote by $\mathbb{N}$ and $\mathbb{N}_0$ the natural numbers excluding and including zero, respectively. By $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$ we denote the real numbers, rational numbers, and the integers, respectively. Moreover, $[a,b] := \{i \in \mathbb{Z} \mid a \leq i \leq b\}$, $[n] := [1,n]$, $\mathbb{R}_+ := \{x \in \mathbb{R} \mid x \geq 0\}$, and $\mathbb{Q}_+ := \{x \in \mathbb{Q} \mid x \geq 0\}$. We denote by $\log(x)$ the ceiling of the binary logarithm of x ($\lceil \log_2(x) \rceil$), where $x \in \mathbb{R}$.

Let $(a_i)_{i=1}^n := (a_1, a_2, \ldots, a_n)$ be a sequence of length n and let $(b_i)_{i=1}^m$ be a sequence of length m. We denote by $x \in (a_i)_{i=1}^n$ that there is an $i \in [n]$ such that $x = a_i$. We denote by $(a_i)_{i=1}^n \subseteq (b_i)_{i=1}^m$ that $(a_i)_{i=1}^n$ is a *subsequence* of $(b_i)_{i=1}^m$. That is, there is an injective function $\sigma\colon [n] \to [m]$ such that $a_i = b_{\sigma(i)}$ for all $i \in [n]$ and $\sigma(i) < \sigma(j)$ for all $i,j \in [n]$ with $i < j$. Moreover, for a set S, we denote by $(a_i)_{i=1}^n \setminus S$ the subsequence of $(a_i)_{i=1}^n$ where an element a_i is removed if and only if $a_i \in S$, for all $i \in [n]$. Appending an element x to sequence $(a_i)_{i=1}^n$ results in the sequence $(a_i)_{i=1}^{n+1}$, where $a_{n+1} = x$.

A *randomized (Monte-Carlo) algorithm* has additionally access to an oracle that, given some number $n \in \mathbb{N}$, draws a value $x \in [n]$ uniformly at random in constant time. A (randomized) algorithm with error probability p is a randomized algorithm that returns the correct answer with probability $1-p$. For a finite alphabet Σ and a language $L \subseteq \Sigma^*$, a (randomized) algorithm for L with a one-sided error probability p is a randomized algorithm that returns for every input $x \in \Sigma^*$ either *yes* or *no*, and one of the following is true:

- If *yes* is returned, then $x \in L$ with probability 1. If *no* is returned, then $x \in L$ with probability p.
- If *yes* is returned, then $x \notin L$ with probability p. If *no* is returned, then $x \notin L$ with probability 1.

We refer to Mitzenmacher and Upfal [37] for more material on randomized algorithms. If it is not stated otherwise, then we use standard notation from graph theory [17]. Graphs are simple and undirected by default.

Temporal graphs. A temporal graph $\mathcal{G} := (V, (E_i)_{i=1}^\tau)$ consists of a set of vertices $V(\mathcal{G}) := V$ and a sequence of edge sets $(E_i)_{i=1}^\tau$. The number τ is the *lifetime* of $\mathcal{G}$. The elements of $\mathcal{E}(\mathcal{G}) := \bigcup_{i\in[\tau]} E_i \times \{i\}$ are called the *time-edges* of $\mathcal{G}$. We say that time-edge $(e,t) \in \mathcal{E}(\mathcal{G})$ has time stamp t and is in time step t. The graph (V, E_i) is called *layer* i of temporal graph $\mathcal{G}$, for all $i \in [\tau]$. The *underlying graph* of $\mathcal{G}$ is the (static) graph $(V, \bigcup_{i=1}^\tau E_i)$. For every $v \in V$ and every $t \in [\tau]$, we denote the *appearance of vertex v at time t* by the pair (v,t). For a time-edge $(\{v,w\},t)$ we call the vertex appearances (v,t) and (w,t) its *endpoints*. We assume that the *size* of $\mathcal{G}$ is $|\mathcal{G}| := |V| + \sum_{i=1}^\tau \max\{1, |E_i|\}$, that is, we do not assume to have compact representations of temporal graphs. For a vertex set $X \subseteq V$ of a temporal graph $\mathcal{G} := (V, (E_i)_{i=1}^\tau)$, we denote by $\mathcal{G}[X]$ the temporal graph $(X, (E'_i)_{i=1}^\tau)$, where $E'_i := \{e \in E_i \mid e \subseteq X\}$. Moreover, we denote the temporal graph $\mathcal{G}$ without the vertices X by $\mathcal{G} - X := \mathcal{G}[V \setminus X]$. For a time-edge set Y, we denote by $\mathcal{G} \setminus Y$ the temporal graph where $V(\mathcal{G} \setminus Y) := V(\mathcal{G})$ and $\mathcal{E}(\mathcal{G} \setminus Y) := \mathcal{E}(\mathcal{G}) \setminus Y$.

The set of vertices of the temporal path $P = (e_i = (\{v_{i-1}, v_i\}, t_i))_{i=1}^m$ is denoted by $V(P) = \{v_i \mid i \in [m] \cup \{0\}\}$. We say that P *visits* the vertex v_i at time t if $t \in [t_i, t_{i+1}]$, where $i \in [m-1]$. The *departure* (or *starting*) *time* of P is t_1 and the *arrival time* of P is t_m. A (δ-restless) temporal s-z path of length m in a temporal graph $\mathcal{G}$ is a *shortest* (δ-restless) temporal s-z path if each temporal s-z path in $\mathcal{G}$ is of length at least m.

A *solution* of an instance $(\mathcal{G}, s, z, \delta, k)$ of Short Restless Temporal Path is a δ-restless temporal s-z path of length at most k in $\mathcal{G}$.

Parameterized complexity. Let Σ be a finite alphabet. A *parameterized problem* L is a subset $L \subseteq \Sigma^* \times \mathbb{N}_0$. The size of an instance $(x, k) \in \Sigma^* \times \mathbb{N}_0$ is denoted by $|x|$ and usually we have that $|x| + k \in O(|x|)$. An instance $(x, k) \in \Sigma^* \times \mathbb{N}_0$ is a *yes*-instance of L if and only if $(x, k) \in L$ (otherwise it is a *no*-instance). A parameterized problem L is *fixed-parameter tractable* (in FPT) if there is an (FPT-)algorithm that decides for every input $(x, k) \in \Sigma^* \times \mathbb{N}_0$ in $f(k) \cdot |x|^{O(1)}$ time whether $(x, k) \in L$, where f is some computable function only depending on k. By slightly abusing the FPT-terminology, we sometimes say that a parameterized problem is fixed-parameter tractable even if the FPT-algorithm has a constant one-sided error probability. A parameterized problem L is in XP if for every input (x, k) one can decide in $|x|^{f(k)}$ time whether $(x, k) \in L$, where f is some computable function only depending on k.

The parameterized analogous of NP and NP-hardness are the W-hierarchy FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq \cdots \subseteq$ W[P] $\subseteq$ XP and W[t]-hardness, where $t \in \mathbb{N} \cup \{\mathrm{P}\}$ and all inclusions are conjectured to be strict. If some W[t]-hard parameterized problem is in FPT, then FPT=W[t]. We refer to Cygan et al. [15] for more material on parameterized complexity.

3 The Algorithm

In this section, we show that Short Restless Temporal Path can be solved in $4^{k-d} \cdot |\mathcal{G}|^{O(1)}$ time with a constant one-sided error probability, where d is the minimum length of a temporal s-z path. More precisely, we show the following.

▶ **Theorem 1.** *For every $p \in \mathbb{R}$ with $0 < p < 1$, there is a randomized $O(4^\ell \cdot \ell^2 |\mathcal{G}|^3 \delta \log(k/p(\ell+1)))$-time algorithm for Short Restless Temporal Path, where $\ell := k - d$ and d is the minimum length of a temporal s-z path. If this algorithm*

- *returns* yes, *then the given instance is a* yes*-instance, and if it*
- *returns* no, *then with probability of at least $1 - p$ the given instance is a* no*-instance.*

The formal proof of Theorem 1 is deferred to Section 3.3. In a nutshell, we only check for δ-restless temporal subpaths whose length are upper-bounded by $2(k-d)+1$ and then puzzle them together to ultimately find a δ-restless temporal s-z path, where d is the minimum length of a temporal s-z path. To detect δ-restless temporal paths of some given length, we employ the algorithm of Thejaswi et al. [41].

▶ **Proposition 2** ([41])**.** *For every $p \in \mathbb{R}$ with $0 < p < 1$ there is a randomized $O(2^k \cdot k|\mathcal{G}|\delta \log(k \cdot 1/p))$-time algorithm that takes as input a temporal graph $\mathcal{G}$, two vertices s, z, and two integers δ, k. If this algorithm*

- *returns* yes, *then there is a δ-restless temporal s-z path of length exactly k in $\mathcal{G}$, and if it*
- *returns* no, *then with probability at least $1 - p$ there is no δ-restless temporal s-z path of length exactly k in $\mathcal{G}$.*

In our algorithm, Proposition 2 can be replaced by any algorithm to find δ-restless temporal paths of length k. For example, with the deterministic $2^{O(k)} \cdot |\mathcal{G}|\delta$-time algorithm of Casteigts et al. [13] instead of Proposition 2, we would end up with a $2^{O(k-d)} \cdot |\mathcal{G}|^3\delta$-time algorithm for Short Restless Temporal Path that is deterministic. Here, the exponential part of the running time is worse than with the randomized subroutine Proposition 2.

First, we set up the geometric perspective on temporal graph based on the temporal distance between vertices. This is the main contribution of this paper and might be of independent interest, as the ideas seem to be transferable to other path finding problems

in temporal graphs where the distance between two vertices is a lower bound. Afterwards, we design a dynamic program to solve Short Restless Temporal Path in $4^{k-d} \cdot |\mathcal{G}|^{O(1)}$ time, where d is the minimum length of a temporal s-z path.

3.1 Geometric Perspective on Temporal Graphs Based on the Temporal Distance

In this section, we present the key ideas of the algorithm behind Theorem 1. To this end, we need some notation. Let $\mathcal{G} := (V, (E_i)_{i=1}^{\tau})$ be a temporal graph with two distinct vertices $s, z \in V(\mathcal{G})$ and $\delta, k \in \mathbb{N}$. We define the distance function $d_{\mathcal{G}} \colon V(\mathcal{G}) \times [\tau] \to \mathbb{N}_0 \cup \{\infty\}$ which maps a vertex $v \in V(\mathcal{G})$ and time $t \in [\tau]$ to the length of a shortest temporal v-z path in $\mathcal{G}$ that departs at a time at least t. If such a temporal path does not exist, then $d_{\mathcal{G}}(v,t) = \infty$. We drop the subscript $\mathcal{G}$ if it is clear from the context.

Intuitively, we now arrange all vertex appearances (v,t) in the plane where the x-axis describes the distance (via temporal paths) of v to z at time t and the y-axis describes the time. Thus, (v,t) gets the point $(d(v,t),t)$. Consider Figure 2 for a moment. We want to visualize a temporal s-z path P in this figure. To this end, we say that P visits vertex appearance (v,t) if P visits v in time step t. Hence, we can depict a temporal path P by connecting the vertex appearances which are visited by P in the visiting order. Note that no temporal v-z path or walk moves downwards. Moreover, among all temporal v-z paths that depart at a time t or later, the shortest of them move with each time-edge further towards z (i.e., to the left). For example, the dotted line in Figure 2 depicts the trajectory of a shortest temporal s-z path with a departure time t. The temporal path departs at time t and arrives at time τ. This is not the case for a shortest δ-restless temporal s-z path P – such a temporal path can move to the right or stay at the same point while visiting multiple vertices. For example, the solid (blue) line in Figure 2 depicts the trajectory of a shortest δ-restless temporal s-z path. Let $\ell := k - d(s,1)$. A crucial observation now is that if P moves "too far" to the right or stays for "too long" at the same spot in the x-axis while visiting multiple vertices, then P would be too far away from z (in terms of temporal paths distance) such that P cannot be of length at most k. This will lead us to the observation that for at least every $(2\ell+1)$-st vertex v which is visited by P (at time t), the vertex appearance (v,t) has the following separation property:

- **(i)** each vertex appearance (u,t') that P visits before v (hence, $v \neq u$) is to the right of (v,t) and thus further away from z than (v,t), and
- **(ii)** each vertex appearance (u,t') that P visits after v (hence, $v \neq u$) is to the left of (v,t) and thus closer to z than (v,t).

Moreover, we will observe that two consecutive vertex appearances which have this separation property, have a similar distance to z – the distances differ by at most $\ell+1$. In Figure 2, these special vertex appearances are at the top-left and bottom-right corners of each gray area. Our dynamic program tries to guess these vertex appearances and then constructs for each gray area in Figure 2 a temporal graph that contains the δ-restless path from the bottom-right corner to the top-left corner of this area. Since we know that these δ-restless temporal paths have length at most $2\ell+1$, we can use Proposition 2 to find them.

Another crucial observation we are going to make is that two δ-restless temporal paths from the bottom-right corner to the top-left corner of two distinct gray areas in Figure 2 cannot visit the same vertex (except for their endpoints). This is the case because the distance of a vertex v to z may only increase as time goes by. Thus, if we find for each gray area in Figure 2 a δ-restless temporal path from the bottom-right corner to the top-left corner, then this gives us a δ-restless temporal s-z path. Henceforth the details follow.

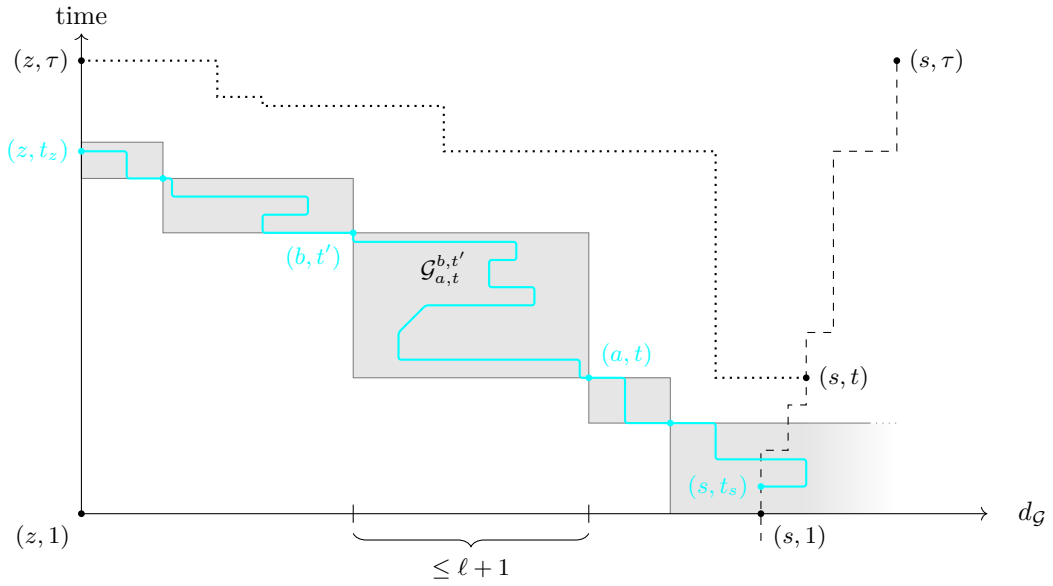


Figure 2 Illustration of the idea behind the dynamic programming table which is used to show Theorem 1. The y-axis describes the time. The x-axis describes the distance to z (via temporal paths). In this plane, a vertex appearance (v,t) gets the position $(d(v,t),t)$. The positions of the vertex appearances of s are on the dashed line. A shortest (non-δ-restless) temporal s-z path that departs at time t is depicted by the dotted line. The trajectory of a shortest δ-restless temporal s-z path which departs at time t_s and arrives at time t_z is depicted by the solid (blue) line. Each gray area depicts a temporal subgraph which we use to compute δ-restless paths from the vertex appearance on the bottom-right corner to the vertex appearance on the top-left corner, e.g., the temporal graph $\mathcal{G}_{a,t}^{b,t'}$.

Before we describe the dynamic programming table in Section 3.2, we define the temporal graph that contains all (shortest) δ-restless paths in a gray area of Figure 2. To this end, we first define sets containing all vertex apperances of such a gray area. For vertex appearances $(a,t),(b,t') \in V(\mathcal{G}) \times [\tau]$, we define

$$\mathcal{A}_{a,t}^{b,t'} := \{(w,t^*) \in V(\mathcal{G}) \times [\tau] \mid d(b,t') < d(w,t^*) < d(a,t), t^* \in [t,t']\} \text{ and}$$
$$\mathcal{A}^{b,t'} := \{(w,t^*) \in V(\mathcal{G}) \times [\tau] \mid \infty > d(w,t^*) > d(b,t'), t^* \leq t'\}.$$

Now, the temporal graph $\mathcal{G}_{a,t}^{b,t'}$ for the gray area between (a,t) and (b,t') with $t \leq t'$ is defined by

$$\begin{aligned}\mathcal{E}\left(\mathcal{G}_{a,t}^{b,t'}\right) := &\left\{(\{v,u\},t^*) \in \mathcal{E}(\mathcal{G}) \;\middle|\; (v,t^*),(u,t^*) \in \mathcal{A}_{a,t}^{b,t'}\right\} \cup \\ &\left\{(\{a,v\},t) \in \mathcal{E}(\mathcal{G}) \;\middle|\; (v,t) \in \mathcal{A}_{a,t}^{b,t'}\right\} \cup \\ &\left\{(\{v,b\},t^*) \in \mathcal{E}(\mathcal{G}) \;\middle|\; t'-\delta \leq t^*, (v,t^*) \in \mathcal{A}_{a,t}^{b,t'} \cup \{(a,t)\}\right\}\end{aligned}$$

and

$$V\left(\mathcal{G}_{a,t}^{b,t'}\right) := \left\{v \in V(\mathcal{G}) \;\middle|\; \exists (e,t^*) \in \mathcal{E}\left(\mathcal{G}_{a,t}^{b,t'}\right) : v \in e\right\}.$$

For the gray area containing s we have to adjust the definition of the corresponding temporal graph slightly. To this end, we define $\mathcal{G}^{b,t'}$ with

$$\mathcal{E}\left(\mathcal{G}^{b,t'}\right) := \left\{(\{v,u\},t^*) \in \mathcal{E}(\mathcal{G}) \;\middle|\; (v,t^*),(u,t^*) \in \mathcal{A}^{b,t'}\right\} \cup \left\{(\{v,b\},t^*) \in \mathcal{E}(\mathcal{G}) \;\middle|\; t'-\delta \leq t^*, (v,t^*) \in \mathcal{A}^{b,t'}\right\}$$

and

$$V\left(\mathcal{G}^{b,t'}\right) := \left\{v \in V(\mathcal{G}) \;\middle|\; \exists (e,t^*) \in \mathcal{E}\left(\mathcal{G}^{b,t'}\right) : v \in e\right\}.$$

In the reminder of this paper, we use these definitions to formulate a dynamic programming table and eventually solve Short Restless Temporal Path.

3.2 The Dynamic Programming Table

In this section, we describe the table T which we are going to use for the dynamic programming, and show its correctness.

Intuitively, the table T has for each vertex appearance (u,t') an entry, and if this entry contains a number $p < \infty$, then p is the length of the shortest δ-restless temporal s-u path that only visits vertex appearances which are, in Figure 2, below and to the right of (u,t').

Let $I := (\mathcal{G}, s, z, \delta, k)$ be an instance of Short Restless Temporal Path, where $k = d(s,1) + \ell$. For all $(u,t') \in V(\mathcal{G}) \times [\tau]$ such that there is an $e \in E_{t'}$ with $v \in e$, we define T as follows. If $d(s,1) - d(u,t') \leq \ell$, then

$$T[u,t'] := \begin{cases} 0, & \text{if } u = s; \\ \ell', & \text{if } u \neq s \text{ and } \ell' \in [2\ell] \text{ is the length of a} \\ & \text{shortest } \delta\text{-restless } s\text{-}u \text{ path in } \mathcal{G}^{u,t'}; \\ \infty, & \text{otherwise.} \end{cases} \tag{1}$$

If $d(s,1) - d(u,t') > \ell$, then

$$T[u,t'] := \min\left(\{\infty\} \cup \left\{T[v,t] + \ell' \;\middle|\; \begin{array}{r} t \in [t'], e \in E_t, v \in e, \text{ where} \\ d(v,t) > d(u,t') \geq d(v,t) - \ell - 1 \\ \text{and } \ell' \in [2\ell+1] \text{ is the length of a} \\ \text{shortest } \delta\text{-restless } v\text{-}u \text{ path in } \mathcal{G}^{u,t'}_{v,t} \end{array}\right\}\right) \tag{2}$$

In the end, we will report that I is a *yes*-instance if and only if there is a $t \in [\tau]$ such that $T[z,t] \leq k$. We will show the correctness of this in the following lemmata. We start with the backwards direction.

▶ **Lemma 3.** *Let $(\mathcal{G}, s, z, \delta, k)$ be an instance of Short Restless Temporal Path. If $T[z,t_z] \leq k < \infty$ (defined in (1) and (2)), then there is a δ-restless temporal s-z path of length at most k in $\mathcal{G}$.*

Proof. We show by induction on the distance to z that if $T[u,t'] = k' < \infty$, then there is a δ-restless s-u path of length k' in $\mathcal{G}^{u,t'}$ which arrives at u at some time step in $[t'-\delta, t']$.

Note that all temporal s-u paths in $\mathcal{G}^{u,t'}$ arrive at some time in $[t'-\delta, t']$. By (1), for each vertex appearance (u,t') with $d(s,1) - d(u,t') \leq \ell$ the induction hypothesis is true – this is our base case.

Now let (u,t') be a vertex appearance with $T[u,t'] = k' < \infty$. Assume that for all vertex appearances (v,t) with $d(v,t) > d(u,t')$ we have that if $T[v,t] = k'' < \infty$, then there is a δ-restless temporal s-v path of length k'' in $\mathcal{G}^{v,t}$ which arrives at v at some time step

in $[t-\delta, t]$. Since $T[u,t'] = k'$, we know by (2) that there is a vertex appearance (v,t) with $T[v,t] = k''$, $t \leq t'$, and $d(v,t) > d(u,t')$. Moreover, there is a δ-restless temporal v-u path P_2 in $\mathcal{G}_{v,t}^{u,t'}$ of length $\ell' = k' - k''$. By the definition of $\mathcal{G}_{v,t}^{u,t'}$, P_2 departs at time t and arrives at some time in $[t'-\delta, t']$. By assumption, there is a δ-restless temporal s-v path P_1 of length k'' in $\mathcal{G}^{v,t}$ which arrives at v at some time step in $[t-\delta, t]$. We now append the time-edges of P_2 to the time-edges of P_1 and claim that the resulting time-edge sequence P is a δ-restless temporal s-u path of length k' which arrives at u at some time in $[t'-\delta, t']$. Observe that P is a δ-restless temporal s-u walk of length $k' = k'' + \ell'$, as P_1 is δ-restless, of length k'', and arrives at v at some time $t^* \in [t-\delta, t]$, and P_2 is of length ℓ' and departs at time t. Moreover, the arrival time of P is the same as the arrival time of P_2.

It remains to show that P does not visit a vertex twice. To see this, we show that $V(\mathcal{G}^{v,t}) \cap V(\mathcal{G}_{v,t}^{u,t'}) = \{v\}$. This will complete the proof, since we know that $V(P_1) \subseteq V(\mathcal{G}^{v,t})$, $V(P_2) \subseteq V(\mathcal{G}_{v,t}^{u,t'})$, P_1 ends at vertex v, and P_2 starts at vertex v. By definition, we have that $v \in (V(\mathcal{G}^{v,t}) \cap V(\mathcal{G}_{v,t}^{u,t'}))$. Assume towards a contradiction that there is a vertex $w \in (V(\mathcal{G}^{v,t}) \cap V(\mathcal{G}_{v,t}^{u,t'})) \setminus \{v\}$. Then, there must be time steps t_1, t_2 such that $(w,t_1) \in \mathcal{A}^{v,t} \cup \{(u,t')\}$ and $(w,t_2) \in \mathcal{A}_{v,t}^{u,t'}$. Note that $d(w,t_1) > d(v,t) > d(w,t_2)$ and hence each temporal w-z path in $\mathcal{G}$ that departs not earlier than t_1 is longer than a shortest w-z path in $\mathcal{G}$ that departs not earlier than t_2. This is a contradiction because $t_1 \leq t \leq t_2$. ◀

To show the forward direction of the correctness, we introduce further notation. Recall from the definition of the dynamic programming table T in (1) and (2) that $\ell = k - d(s,1)$. Assume the input instance I is a *yes*-instance. Thus there is a δ-restless temporal s-z path $P = ((\{v_{i-1}, v_i\}, t_i))_{i=1}^{k}$ of length at most $d(s,1) + \ell$ in $\mathcal{G}$. Let $s = v_0, v_1, \ldots, v_k = z$ be the order in which P visits the vertices in $V(P)$. For simplicity, let $t_0 := 1$ and $t_{k+1} := t_k$. For all $i \in [0,k]$, we say that v_i is a *distance separator* if (i) $d(v_i, t_{i+1}) < d(v_j, t_{j+1})$ for all $j \in [0, i-1]$, and (ii) $d(v_i, t_{i+1}) > d(v_j, t_{j+1})$ for all $j \in [i+1, k]$.

Before we show the forward direction of the correctness of the dynamic programming table T, we show that P visits a distance separator on regular basis.

▶ **Lemma 4.** *For all $i \in [0,k]$ there is a $j \in [0, 2\ell]$ such that v_{i+j} is a distance separator.*

Proof. We show this statement with a reverse induction on the length of P. As z is clearly a distance separator, the claim is true for all values in $[\max\{0, k-2\ell\}, k]$. This is the base case of our induction.

Let $k - 2\ell > 0$ and let $i \in [0, k-2\ell-1]$ and assume that for all $i' \in [i+1, k]$ there is a $j' \in [0, 2\ell]$ such that $v_{i'+j'}$ is a distance separator. Let $n \in [i', k]$ be the smallest possible number such that v_n is a distance separator. Let $f \in [k]$ be the smallest possible number such that $d(v_f, t_{f+1}) - d(v_n, t_{n+1}) = \ell + 1$. Note that if such an f does not exist, then the claim is true. Hence, we assume that such an f exists. Note that $f \leq i$, otherwise n is not the smallest possible number.

We now show that $n - f \leq 2\ell + 1$. Assume towards a contradiction that the temporal v_f-v_n path contained in P has length $n - f > 2\ell + 1$. This is a lower bound for the length of P. We get the following.

$$d(s,1) - d(v_f, t_{f+1}) + 2\ell + 1 + d(v_n, t_{n+1}) < k = d(s,1) + \ell$$
$$\implies d(v_n, t_{n+1}) - d(v_f, t_{f+1}) + \ell + 1 < 0 \implies \ell + 1 < d(v_f, t_{f+1}) + d(v_n, t_{n+1})$$

This is a contradiction to $d(v_f, t_{f+1}) - d(v_n, t_{n+1}) = \ell + 1$.

Next, we show that, between v_f and v_{n-1}, P must visit a distance separator. Assume towards a contradiction that $v_{n-j''}$ is not a distance separator, for all $j'' \in [n-f]$. Hence, for all $p \in [d(v_n, t_{n+1}) + 1, d(v_f, t_{f+1})]$, there are two distinct $q, r \in [f, n-1]$ such that

$d(v_r, t_{r+1}) = d(v_q, t_{q+1}) = p$. Since $[d(v_n, t_{n+1}) + 1, d(v_f, t_{f+1})] = \ell$, we get by the pigeonhole principle that the temporal v_f-v_n path contained in P has length $n - f > 2\ell + 1$ – a contradiction. ◀

Finally, we are set to show the forward direction and Theorem 1 afterwards.

▶ **Lemma 5.** *Let $(\mathcal{G}, s, z, \delta, k)$ be an instance of Short Restless Temporal Path. If there is a δ-restless temporal s-z path in $\mathcal{G}$ of length at most k with arrival time t_k, then $T[z, t_k] \leq k$ (defined in (1) and (2)).*

Proof. Let $P = ((\{v_{i-1}, v_i\}, t_i))_{i=1}^{k}$ be a shortest δ-restless temporal s-z path of length at most $k = d(s, 1) + \ell$ in $\mathcal{G}$. Let $s = v_0, v_1, \ldots, v_k = z$ be the order in which P visits the vertices in $V(P)$. For simplicity, let $t_0 := 1$ and $t_{k+1} := t_k$. Moreover, let m be the number of distance separators visited by P and let $\sigma \colon [m] \to [0, k]$ be an injective function such that $v_{\sigma(i)}$ is the i-th distance separator which is visited by P (from s to z), for all $i \in [m]$. Note that, the vertex $v_{\sigma(i)}$ is the i-th distance separator visited by P and thus $\sigma(i)$ also describes the length of the δ-restless temporal s-$v_{\sigma(i)}$ subpath contained in P.

We now show that for all $i \in [m]$ we have that $T[v_{\sigma(i)}, t_{\sigma(i)+1}] \leq \sigma(i)$. If $\sigma(1) = 0$, then s is a distance separator and the claim is clearly true, see (1). Otherwise, by Lemma 4, we have $\sigma(1) \leq 2\ell$. Hence, P contains a δ-restless temporal s-$v_{\sigma(1)}$ path of length $\sigma(1) \leq 2\ell$ which is contained in $\mathcal{G}^{v_{\sigma(1)}, t_{\sigma(1)+1}}$. Thus, $T[v_{\sigma(1)}, t_{\sigma(1)+1}] \leq \sigma(1)$.

Now assume that for some $i \in [2, m]$ we have that $T[v_{\sigma(i-1)}, t_{\sigma(i-1)+1}] \leq \sigma(i-1)$. Observe that $t_{\sigma(i-1)+1} \leq t_{\sigma(i)+1}$ and that $d(v_{\sigma(i-1)}, t_{\sigma(i-1)+1}) > d(v_{\sigma(i)}, t_{\sigma(i)+1})$. By Lemma 4, we have that $\sigma(i) - \sigma(i-1) \leq 2\ell + 1$ and that the δ-restless temporal $v_{\sigma(i-1)}$-$v_{\sigma(i)}$ path Q contained in P is of length $\sigma(i) - \sigma(i-1) \leq 2\ell + 1$. As all of the at most 2ℓ vertices in $V(Q) \setminus \{v_{\sigma(i-1)}, v_{\sigma(i)}\}$ are not distance separators, we have by the pigeonhole principle that $d(v_{\sigma(i-1)}, t_{\sigma(i-1)+1}) - d(v_{\sigma(i)}, t_{\sigma(i)+1}) \leq \ell + 1$. Moreover, note that Q in $\mathcal{G}^{v_{\sigma(i)}, t_{\sigma(i)+1}}_{v_{\sigma(i-1)}, t_{\sigma(i-1)+1}}$, because $v_{\sigma(i-1)}$ and $v_{\sigma(i)}$ are distance separators. Hence, by (2), we have that $T[v_{\sigma(i)}, t_{\sigma(i)+1}] \leq T[v_{\sigma(i-1)}, t_{\sigma(i-1)+1}] + \sigma(i) - \sigma(i-1) \leq \sigma(i)$, as we have $T[v_{\sigma(i-1)}, t_{\sigma(i-1)+1}] \leq \sigma(i-1)$ by assumption.

Since $z = v_k$, we have that k is the only number in $[0, k]$ with $d(v_k, t_{k+1}) = 0$. Hence, v_k is the last distance separator and thus $\sigma(m) = k$. Finally, by our induction, we have that $T[z, t_{k+1}] \leq k$. ◀

3.3 Putting the Pieces Together

In this section, we use use the tools from previous sections to show first show Theorem 1. Beforehand, we show for the running the analysis that all necessary values of our distances function $d(\cdot, \cdot)$ are computable in linear time.

▶ **Lemma 6.** *Given a temporal graph $\mathcal{G} := (V, (E_i)_{i=1}^{\tau})$ and a vertex z, one can compute in $O(|\mathcal{G}|)$ time the value $d(v, t)$, for all $v \in V$ and $t \in [\tau]$ where v is not isolated in the graph (V, E_t).*

Proof. We will construct a directed graph D where each arc has either weight zero or one such that the weight of a shortest z-v_t path equals the value of $d(v, t)$, for all $v \in V$ and $t \in [\tau]$ where v is not isolated in the graph (V, E_t). Then, a slightly modified breadth-first search will do the task.

We compute the set $\mathcal{V}$ of non-isolated vertex appearances. That is, $\mathcal{V} := \{(v, t) \in V(\mathcal{G}) \times [\tau] \mid \exists e \in E_t \colon v \in e\}$. Note that this can be done in $O(|\mathcal{G}|)$ time and that $|\mathcal{V}| \leq 2|\mathcal{G}|$. Now we are ready to define D by $V(D) := \{z\} \cup \{v_t \mid (v, t) \in \mathcal{V}\}$ and

$$E(D) := \{(v_t, u_t), (u_t, v_t) \mid \{v, u\} \in E_t, (v, t), (u, t) \in \mathcal{V} \text{ and } u \neq v\} \cup$$
$$\{(v_{t_2}, v_{t_1}) \mid v_{t_1}, v_{t_2} \in V(D) \text{ and } t_2 = \min\{t \mid (v, t) \in \mathcal{V} \text{ and } t > t_1\}\} \cup$$
$$\{(z, z_t) \mid z_t \in V(D) \text{ and } t = \max\{t' \mid (z, t') \in \mathcal{V}\}\}.$$

Now all arcs in $\{(v_t, u_t), (u_t, v_t) \mid (v, t), (u, t) \in \mathcal{V} \text{ and } u \neq v\}$ get weight one, while all the other arcs get weight zero. Note that $V(D) + E(D) \in O(|\mathcal{G}|)$ and that D can be constructed in $O(|\mathcal{G}|)$ time. Observe that for every temporal v-z path P in $\mathcal{G}$ with departure time t there is a z-v_t path in D whose accumulated edge-weight equals the length of P. Hence, if we know the minimum edge-weight of the paths from z to all vertices in D, then we also know the value $d(v, t)$, for all $v \in V$ and $t \in [\tau]$ where v is not isolated in the graph (V, E_t). Thus, we employ a breadth-first search that starts at z and only explores an arc of weight one if there is currently no arc of weight zero which could be explored instead. At each vertex $v_t \in V(D)$ we store the edge-weight $d(v, t)$ of the path from z to this vertex. Hence, the overall running time of this procedure is $O(|\mathcal{G}|)$ time. ◀

Finally, we are set to show Theorem 1: For every $p \in \mathbb{R}$ with $0 < p < 1$, there is a randomized $O(4^\ell \cdot \ell^2 |\mathcal{G}|^3 \delta \log(k/p(\ell+1)))$-time algorithm for Short Restless Temporal Path, where $\ell := k - d$ and d is the minimum length of a temporal s-z path. If this algorithm

- returns *yes*, then the given instance is a *yes*-instance, and if it
- returns *no*, then with probability of at least $1 - p$ the given instance is a *no*-instance.

Proof of Theorem 1. Let $I := (\mathcal{G}, s, z, \delta, k)$ be an instance of Short Restless Temporal Path. We perform the following. First, we compute the set $\mathcal{V}$ of non-isolated vertex appearances. That is, $\mathcal{V} := \{(v, t) \in V(\mathcal{G}) \times [\tau] \mid \exists e \in E_t \colon v \in e\}$. Note that this can be done in $O(|\mathcal{G}|)$ time and that $|\mathcal{V}| \leq 2|\mathcal{G}|$. By Lemma 6, we compute $d(v, t)$ for all $(v, t) \in \mathcal{V}$ in $O(|\mathcal{G}|)$ time. We may assume that there is a temporal s-z path in $\mathcal{G}$ and that a shortest of them has length at most k, otherwise I is clearly a *no*-instance. We set $\ell := k - d(s, 1) = k - d(s, t)$, where $t = \min\{t' \in [\tau] \mid (s, t') \in \mathcal{V}\}$. Note that the table T, defined in (1) and (2), has $O(|\mathcal{G}|)$ entries – one for each element in $\mathcal{V}$. To compute one entry in T, we consider at most $O(|\mathcal{G}|)$ other entries in T and for each of them we have to check at most $2\ell + 1$ times whether a temporal graph of size $O(|\mathcal{G}|)$ has a δ-restless temporal path of length $\ell' \in [2\ell + 1]$ between two distinct vertices. We answer each of these checks by Proposition 2 with a one-sided error probability of at most p' in $O(4^\ell \cdot \ell |\mathcal{G}| \delta \log(\ell \cdot 1/p'))$ time. How we set the error probability p' will be determined in a moment.

We say that I is a *yes*-instance if and only if there is a $(z, t) \in \mathcal{V}$ such that $T[z, t] \leq k$. If Proposition 2 reports *yes*, then with probability one, there is such a δ-restless temporal path in question. Hence, by Lemma 3, if our overall algorithm reports *yes*, then I is a *yes*-instance – the error probability is zero in this case. If our overall algorithm reports *no*, then the probability that I is a *yes*-instance shall be at most $1 - p$. By Lemma 5, it remains to determine p'. Recall from Lemma 4 that a δ-restless temporal s-z path P of length at most k visits at least every $2\ell + 1$ vertices one distance separator. Hence, we can identify at most $\lceil k/\lceil \ell+1/2 \rceil \rceil \in O(k/\ell+1)$ vertex appearances which are visited by P and thus $O(k/\ell+1)$ calls of the algorithm behind Proposition 2 such that if these calls are answered correctly then this causes our overall algorithm to report *yes*, as $T[z, t] \leq k$ for some $t \in [\tau]$. Hence, there is a $p' \in O(p(\ell+1)/k)$ such that we have an error probability of at most p in the case our overall algorithm answers *no*. Thus, we can compute all entries of T in $O(4^\ell \cdot \ell^2 |\mathcal{G}|^3 \delta \log(k/p(\ell+1)))$ time, where $\ell := k - d$ and d is the minimum length of a temporal s-z path. ◀

On a more practical note, one can observe that in order to compute one entry for vertex appearances (u, t) of table T, we only consider table entries of vertex appearances with low temporal distance to (u, t). Thus, for temporal graphs that are nowhere dense in terms of the temporal distance, it seems reasonable that the presented algorithm does not induce a quadratic running time, in terms of the temporal graph size, on top of running time of Proposition 2 (e.g., in contact networks where mass events are prohibited). Moreover, a δ-restless temporal path of length k has a time horizon of at most $(k-1)\delta$. Hence, with an overhead of $O(\tau)$ one could guess the departure time t of the δ-restless s-z path and discard all time-edge (e, t') with $t' < t$ or $t' > t + (k-1)\delta + 1$. This potentially decreases the parameter $k - d$ and thus the exponential part of the running time substantially, where d is the minimum length of a temporal s-z path.

4 Conclusion

We showed that Short Restless Temporal Path can be solved in $4^{k-d} \cdot |\mathcal{G}|^{O(1)}$ time with a one-sided error probability of at most $2^{-|\mathcal{G}|}$, where d is the minimum length of a temporal s-z path. In the corresponding algorithm, we have only one subroutine with a super-polynomial running time: an algorithm to find a δ-restless temporal path of length at most $2(k-d)+1$. As this is the only subroutine with a non-zero error probability, our algorithm becomes deterministic $2^{O(k-d)} \cdot |\mathcal{G}|^{O(1)}$-time algorithm, if this subroutine is replaced by the algorithm of Casteigts et al. [13]. Note that the deterministic algorithm does not run in polynomial space and that the running time upper-bound is worse.

We believe that our algorithmic approach opens new research directions to advance further: First, we wonder how good our algorithm performs in an experimental setup comparable to the one of Thejaswi et al. [41]. Second, one could study in detail the temporal subgraphs on which we employ Proposition 2. In principle, Proposition 2, could be replaced with any other algorithm for Short Restless Temporal Path. Do these specific temporal subgraphs admit structural properties which are algorithmically useful? Third, we believe that our geometric perspective can be applied to other temporal graph problems. In particular, for temporal graph problems which ask for specific temporal paths, e.g., temporal paths that obey certain robustness properties [24], or temporal paths that visit all vertices at least once [20, 21, 36] parameterized by the temporal diameter, that is, the length of the longest shortest temporal path between two arbitrary vertices.

References

1 Eleni C. Akrida, Jurek Czyzowicz, Leszek Gąsieniec, Łukasz Kuszner, and Paul G. Spirakis. Temporal flows in temporal networks. *Journal of Computer and System Sciences*, 103:46–60, 2019. doi:10.1016/j.jcss.2019.02.003.

2 Eleni C. Akrida, Leszek Gąsieniec, George B. Mertzios, and Paul G. Spirakis. The complexity of optimal design of temporally connected graphs. *Theory of Computing Systems*, 61(3):907–944, 2017. doi:10.1007/s00224-017-9757-x.

3 Eleni C. Akrida, George B. Mertzios, Paul G. Spirakis, and Christoforos L. Raptopoulos. The temporal explorer who returns to the base. *Journal of Computer and System Sciences*, 120:179–193, 2021. doi:10.1016/j.jcss.2021.04.001.

4 Noga Alon, Gregory Z. Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving MAX-r-SAT above a tight lower bound. *Algorithmica*, 61(3):638–655, 2011. doi:10.1007/s00453-010-9428-7.

5 Kyriakos Axiotis and Dimitris Fotakis. On the size and the approximability of minimum temporally connected subgraphs. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *Leibniz International Proceedings in Informatics*, pages 149:1–149:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.149.

6 Albert-László Barabási. *Network Science*. Cambridge University Press, 2016.

7 Matthias Bentert, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. Efficient computation of optimal temporal walks under waiting-time constraints. *Applied Network Science*, 5(72):1–26, 2020. doi:10.1007/s41109-020-00311-0.

8 Ivona Bezáková, Radu Curticapean, Holger Dell, and Fedor V. Fomin. Finding detours is fixed-parameter tractable. *SIAM Journal on Discrete Mathematics*, 33(4):2326–2345, 2019. doi:10.1137/17M1148566.

9 Sandeep Bhadra and Afonso Ferreira. Computing multicast trees in dynamic networks and the complexity of connected components in evolving graphs. *Journal of Internet Services and Applications*, 3(3):269–275, 2012. doi:10.1007/s13174-012-0073-z.

10 Hans L. Bodlaender and Tom C. van der Zanden. On exploring always-connected temporal graphs of small pathwidth. *Information Processing Letters*, 142:68–71, 2019. doi:10.1016/j.ipl.2018.10.016.

11 Binh-Minh Bui-Xuan, Afonso Ferreira, and Aubin Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003. doi:10.1142/S0129054103001728.

12 Benjamin M. Bumpus and Kitty Meeks. Edge exploration of temporal graphs. In *Proceedings of the 32st International Workshop on Combinatorial Algorithms (IWOCA)*, volume 12757 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2021. doi:10.1007/978-3-030-79987-8_8.

13 Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. Finding temporal paths under waiting time constraints. *Algorithmica*, 83(9):2754–2802, 2021. doi:10.1007/s00453-021-00831-w.

14 Robert Crowston, Michael R. Fellows, Gregory Z. Gutin, Mark Jones, Eun Jung Kim, Fran Rosamond, Imre Z. Ruzsa, Stéphan Thomassé, and Anders Yeo. Satisfying more than half of a system of linear equations over GF(2): A multivariate approach. *Theory of Computing Systems*, 80(4):687–696, 2014. doi:10.1016/j.jcss.2013.10.002.

15 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.

16 Argyrios Deligkas and Igor Potapov. Optimizing reachability sets in temporal graphs by delaying. *Information and Computation*, 285(Part):104890, 2022. doi:10.1016/j.ic.2022.104890.

17 Reinhard Diestel. *Graph Theory*, volume 173. Springer, 5 edition, 2016. doi:10.1007/978-3-662-53622-3.

18 Jessica Enright, Kitty Meeks, George B. Mertzios, and Viktor Zamaraev. Deleting edges to restrict the size of an epidemic in temporal networks. *Journal of Computer and System Sciences*, 119:60–77, 2021. doi:10.1016/j.jcss.2021.01.007.

19 Jessica Enright, Kitty Meeks, and Fiona Skerman. Assigning times to minimise reachability in temporal graphs. *Journal of Computer and System Sciences*, 115:169–186, 2021. doi:10.1016/j.jcss.2020.08.001.

20 Thomas Erlebach, Michael Hoffmann, and Frank Kammer. On temporal graph exploration. *Journal of Computer and System Sciences*, 119:1–18, 2021. doi:10.1016/j.jcss.2021.01.005.

21 Thomas Erlebach, Frank Kammer, Kelin Luo, Andrej Sajenko, and Jakob T. Spooner. Two moves per time step make a difference. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *Leibniz International Proceedings in Informatics*, pages 141:1–141:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.141.

22 Thomas Erlebach and Jakob T. Spooner. Faster exploration of degree-bounded temporal graphs. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 117 of *Leibniz International Proceedings in Informatics*, pages 36:1–36:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.36.

23 Till Fluschnik, Hendrik Molter, Rolf Niedermeier, Malte Renken, and Philipp Zschoche. Temporal graph classes: A view through temporal separators. *Theoretical Computer Science*, 806:197–218, 2020. doi:10.1016/j.tcs.2019.03.031.

24 Eugen Füchsle, Hendrik Molter, Rolf Niedermeier, and Malte Renken. Delay-robust routes in temporal graphs. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS)*, Leibniz International Proceedings in Informatics. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. To appear.

25 Gregory Z. Gutin, Eun Jung Kim, Michael Lampis, and Valia Mitsou. Vertex cover problem parameterized above and below tight bounds. *Theory of Computing Systems*, 48(2):402–410, 2011. doi:10.1007/s00224-010-9262-y.

26 Gregory Z. Gutin and Matthias Mnich. A survey on graph problems parameterized above and below guaranteed values. *CoRR*, abs/2207.12278, 2022. doi:10.48550/arXiv.2207.12278.

27 Gregory Z. Gutin, Leo van Iersel, Matthias Mnich, and Anders Yeo. Every ternary permutation constraint satisfaction problem parameterized above average has a kernel with a quadratic number of variables. *Journal of Computer and System Sciences*, 78(1):151–163, 2012. doi:10.1016/j.jcss.2011.01.004.

28 Petter Holme. Temporal network structures controlling disease spreading. *Physical Review E*, 94.2:022305, 2016. doi:10.1103/PhysRevE.94.022305.

29 Leon Kellerhals, Tomohiro Koana, and Pascal Kunz. Vertex cover and feedback vertex set above and below structural guarantees. 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 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.IPEC.2022.19.

30 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002. doi:10.1006/jcss.2002.1829.

31 William Ogilvy Kermack and Anderson G. McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 115(772):700–721, 192o7. doi:10.1098/rspa.1927.0118.

32 Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. Interference-free walks in time: Temporally disjoint paths. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4090–4096. International Joint Conferences on Artificial Intelligence Organization, 2021. doi:10.24963/ijcai.2021/563.

33 Meena Mahajan and Venkatesh Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *Journal of Algorithms*, 31(2):335–354, 1999. doi:10.1006/jagm.1998.0996.

34 Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. Parameterizing above or below guaranteed values. *Journal of Computer and System Sciences*, 75(2):137–153, 2009. doi:10.1016/j.jcss.2008.08.004.

35 George B. Mertzios, Othon Michail, and Paul G. Spirakis. Temporal network optimization subject to connectivity constraints. *Algorithmica*, 81(4):1416–1449, 2019. doi:10.1007/s00453-018-0478-6.

36 Othon Michail and Paul G. Spirakis. Traveling salesman problems in temporal graphs. *Theoretical Computer Science*, 634:1–23, 2016. doi:10.1016/j.tcs.2016.04.006.

37 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

38 Hendrik Molter, Malte Renken, and Philipp Zschoche. Temporal reachability minimization: Delaying vs. deleting. In *Proceedings of the 46th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 202 of *Leibniz International Proceedings in Informatics*, pages 76:1–76:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.MFCS.2021.76.

39 Mark E J Newman. *Networks*. Oxford University Press, 2018.

40 Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Physical Review E*, 84(1):016105, 2011. doi:10.1103/PhysRevE.84.016105.

41 Suhas Thejaswi, Juho Lauri, and Aristides Gionis. Restless reachability problems in temporal graphs. *CoRR*, abs/2010.08423, 2020.

42 Huanhuan Wu, James Cheng, Yiping Ke, Silu Huang, Yuzhen Huang, and Hejun Wu. Efficient algorithms for temporal path computation. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2927–2942, 2016. doi:10.1109/TKDE.2016.2594065.