41st International Symposium on Mathematical Foundations of Computer Science

MFCS 2016, August 22–26, 2016, Kraków, Poland

Edited by

Piotr Faliszewski
Anca Muscholl
Rolf Niedermeier

LIPICS

LIPIcs – Vol. 58 – MFCS'16 www.dagstuhl.de/lipics

Editors

Piotr Faliszewski	Anca Muscholl	Rolf Niedermeier
AGH University	Université Bordeaux	Technische Universität Berlin
Kraków, Poland	Talence, France	Berlin, Germany
`faliszew@agh.edu.pl`	`anca@labri.fr`	`rolf.niedermeier@tu-berlin.de`

ACM Classification 1998
F. Theory of Computation

ISBN 978-3-95977-016-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-016-3.

Publication date
August, 2016

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.MFCS.2016.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016).
Editors: Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Foreword

International Symposium on Mathematical Foundations of Computer Science (MFCS conference series) is a well-established venue for presenting research papers in theoretical computer science. The broad scope of the conference encourages interactions between researchers who might not meet at more specialized venues.

The first MFCS conference was organized in 1972 in Jabłonna (near Warsaw, Poland). Since then, the conference traditionally moved between the Czech Republic, Slovakia, and Poland. A few years ago, the conference started traveling around Europe (in 2013 it was held in Austria, then in 2014 in Hungary, and most recently, in 2015, in Italy), yet this year it visited Poland once again.

As compared to the previous editions, this year the conference featured several changes. The most prominent one regarded switching to publishing the proceedings in the Leibniz International Proceedings in Informatics (LIPIcs) series. In effect, there were more relaxed publishing requirements (in particular, the papers were limited to twelve pages, but excluding the references), registration fee was slightly lower, and – foremost – the authors kept the copyright for their papers (the proceedings are published under the Creative Commons CC-BY license; CC-BY 3.0 DE). A less significant change regarded partitioning the submission process. The authors first registered their papers' abstracts (by the 21st of April, 2016) and only then their content (by the 25th of April, 2016). This division has helped with the assignment of the papers to the PC members.

Over 220 abstracts were submitted, of which 195 materialized as papers, of which 84 were finally accepted. The authors of the submitted papers represent nearly 40 countries. Each paper was assigned to three PC members, who reviewed and discussed them thoroughly over a period of nearly seven weeks. As the co-chairs of the program committee, we would like to express our deep gratitude to all the committee members for their hard, dedicated work. The quality of the submitted papers was very high and many good papers had to be rejected.

The conference featured five invited talks, by Shai Ben-David (University of Waterloo, Canada), Mikołaj Bojańczyk (University of Warsaw, Poland), Patricia Bouyer-Decitre(LSV, CNRS & ENS de Cachan, France), Tobias Friedrich (Hasso Plattner Institute, Potsdam, Germany), and Virginia Vassilevska Williams (Stanford University, USA). We would like to thank them deeply for their contributions and their time.

These are the first MFCS proceedings published in the Dagstuhl/LIPIcs series. Thus we would like to particularly thank Marc Herbstritt and the LIPIcs team for all the help and support. We believe that the cooperation between MFCS and Dagstuhl/LIPIcs in the future will be as seamless and fruitful as ours.

Piotr Faliszewski
Anca Muscholl
Rolf Niedermeier

Conference Organization

Program Committee

Luca Aceto	Reykjavik University, Iceland
Eric Allender	Rutgers University, USA
Christer Bäckström	Linköping University, Sweden
Arnold Beckmann	Swansea University, UK
Philip Bille	Technical University of Denmark, Denmark
Tomas Brazdil	Masaryk University, Czech Republic
Laurent Bulteau	University Lyon 1, France
Edith Cohen	Google, USA
Veronique Cortier	CNRS, Loria, France
Mark De Berg	Technische Universiteit Eindhoven, Netherlands
Gabriele Di Stefano	University of L'Aquila, Italy
Piotr Faliszewski	AGH University, Poland (co-chair)
Alain Finkel	LSV, ENS Cachan & CNRS, France
Vojtech Forejt	Oxford University, UK
Laurent Gourves	Lamsade, France
Jarosław Grytczuk	Jagiellonian University, Poland
Martin Hoefer	Max-Planck-Institut für Informatik, Germany
Artur Jeż	University of Wrocław, Poland
Dietrich Kuske	Technische Universität Ilmenau, Germany
Jérôme Lang	Lamsade, France
Sophie Laplante	Université Paris Diderot Paris 7, France
Sławomir Lasota	University of Warsaw, Poland
Helger Lipmaa	University of Tartu, Estonia
Markus Lohrey	University of Siegen, Germany
Veli Mäkinen	University of Helsinki, Finland
Wim Martens	University of Bayreuth, Germany
Anca Muscholl	Université Bordeaux, France (co-chair)
Rolf Niedermeier	Technische Universität Berlin, Germany (co-chair)
Joel Ouaknine	Oxford University, UK
Katarzyna Paluch	University of Wrocław, Poland
Doron Peled	Bar Ilan University, Israel
Maria Polukarov	University of Southampton, UK
Simona Ronchi Della Rocca	Universita' di Torino, Italy
Pierluigi San Pietro	Politecnico di Milano, Italy
Sven Schewe	University of Liverpool, UK
Henning Schnoor	University of Kiel, Germany
Maria Serna	Universitat Politecnica de Catalunya, Spain
Martin Skutella	Technische Universität Berlin, Germany
Daniel Stefankovic	University of Rochester, USA
Frank Stephan	National University of Singapore, Singapore
Christino Tamon	Clarkson University, USA
Mirek Truszczynski	University of Kentucky, USA
Emilio Tuosto	University of Leicester, UK

External Reviewers

Faried Abu Zaid	C. Aiswarya	Alessandro Aloisio
Carme Alvarez	Kazuyuki Amano	Antonios Antoniadis
Timos Antonopoulos	Itai Arad	Alessandro Artale
David Auger	Martin Aumüller	Giovanni Bacci
Giorgio Bacci	Max Bannach	Nikhil Bansal
Nicolas Basset	Djamal Belazzougui	Marco Bernardo
Marcello M. Bersani	Dietmar Berwanger	René Van Bevern
Olaf Beyersdorff	Marcin Bienkowski	Stefano Bistarelli
Timothy Black	Maria J. Blesa	Sebastian Böcker
Benedikt Bollig	Edouard Bonnet	Julian Bradfield
Vasco Brattka	Paul Breiding	Romain Brenguier
Jop Briet	James Brotherston	Dan Browne
Nader Bshouty	Antonio Bucciarelli	Peter Buergisser
Jaroslaw Byrka	Cristian S. Calude	Arnaud Carayol
Clement Carbonnel	Katarina Cechlarova	Andrea Cerone
Ada Chan	Yanping Chen	Alessandra Cherubini
Yann Chevaleyre	Dmitry Chistikov	Tobias Christiani
Vincenzo Ciancia	Serafino Cicerone	Anne Condon
Patrick Hagge Cording	Miguel Couceiro	Basile Couëtoux
Roy Crole	Ágnes Cseh	Fabio Cunial
Wojciech Czerwiński	Pedro R. D'Argenio	Mattia D'Emidio
Stefan Dantchev	Bernardo M. David	Holger Dell
Josep Diaz	Nico Döttling	Laurent Doyen
Manfred Droste	Szymon Dudycz	Christoph Dürr
Hicham El-Zein	Michael Elberfeld	Lior Eldar
David Frutos Escrig	Marco Faella	John Fearnley
Guillaume Fertin	Nathanaël Fijalkow	Till Fluschnik
Lila Fontes	Fredrik Nordvall Forsberg	Marie Fortin
Dimitris Fotakis	Eli Fox-Epstein	Dominik D. Freydenberger
Anna Frid	Ophir Friedler	Vincent Froese
Travis Gagie	Didier Galmiche	Moses Ganardi
Álvaro García-Pérez	Paul Gastin	Rati Gelashvili
George Giakkoupis	Aristides Gionis	Stefan Göller
Robert Granger	Martin Grohe	Luciano Gualà
Rohit Gurjar	Stefan Haar	Christoph Haase
Peter Habermehl	Matthew Hague	Arnd Hartmanns
Chaodong He	Mika Hirvensalo	John M. Hitchcock
Peter Høyer	Chien-Chung Huang	Paul Hunter
Anisse Ismaili	Takehiro Ito	Sanjay Jain
Sune K. Jakobsen	Emmanuel Jeandel	Jisu Jeong
Mark Jerrum	Seungbum Jo	Stephen Jordan
Stasys Jukna	Volker Kaibel	Igor Kaitovic
Frank Kammer	Eleni Kanellou	Mamadou Moustapha Kanté
Juha Kärkkäinen	Petteri Kaski	Joost-Pieter Katoen
Bart De Keijzer	Stefan Kiefer	Sandra Kiefer
Eun Jung Kim	Björn Kinscher	David Kirkpatrick

Jyrki Kivinen	Ralf Klasing	Bartek Klin
Sigrid Knust	Sang-Ki Ko	Tomasz Kociumaka
Bojana Kodric	Mikko Koivisto	Pavel Kolev
Christian Komusiewicz	Daniel König	Juha Kontinen
Eryk Kopczynski	Sajin Koroth	Robin Kothari
Andreas Krebs	Jan Kretinsky	Stephan Kreutzer
Antonin Kucera	Manfred Kufleitner	Sebastian Kuhnert
Raghav Kulkarni	Mrinal Kumar	Adam Kunysz
Orna Kupferman	Sebastian Küpper	Marcin Kurdziel
Alexander Kurz	Martin Kutrib	Anthony Labarre
Michael Lampis	Dominique Larchey-Wendling	Sven Laur
Mathieu Lauriere	Francois Le Gall	Stephane Le Roux
Erik Jan van Leeuwen	Stefano Leucci	Anthony Leverrier
Mateusz Lewandowski	Didier Lime	Anthony Widjaja Lin
Luigi Liquori	Enric Cosme Llópez	Shachar Lovett
Etienne Lozes	Giorgio Lucarelli	Martin Lück
Christopher Lynch	Alexis Maciel	Krzysztof Magiera
Adam Malinowski	Florin Manea	Bodo Manthey
Jieming Mao	Nicolas Markey	Bastien Maubert
Richard Mayr	Pierre Mckenzie	Nicole Megow
Saeed Mehrabi	Sebastian Meiser	Piotr Micek
Dimitrios Michail	Matteo Mio	Hendrik Molter
Tobias Mömke	Benjamin Monmege	Kenichi Morita
Angelo Morzenti	Peter Mosses	Wolfgang Mulzer
Daniel Nagaj	Paresh Nakhe	André Nichterlein
Andre Nies	Matthias Niewerth	Bengt J. Nilsson
Petr Novotný	Jan Obdrzalek	Pascal Ochem
Joanna Ochremiak	Alexander Okhotin	Miguel Romero Orth
Yota Otachi	Jan Otop	Denis Pankratov
Francesco Pasquale	Daniel Paulusma	Sylvain Perifel
Cynthia Phillips	Giovanni Pighizzini	Jean-Eric Pin
Joao Sousa Pinto	Marek Piotrów	Solon Pissis
Matej Pivoluska	Jaco van de Pol	Yann Ponty
Amaury Pouly	Thomas Powell	Matteo Pradella
Simon Puglisi	Florian Rabe	Mathieu Raffinot
Narad Rampersad	Steven Ramsay	Jean Jose Razafindrakoto
Alexander Razborov	Stefano Crespi Reghizzi	Vojtech Rehak
Leonid Reyzin	Leonid Reyzin	Fabián Riquelme
Romeo Rizzi	Emanuele Rodaro	Trent Rogers
Matteo Rossi	Noy Rotbart	Jörg Rothe
Michał Różański	Irena Rusu	Chandan Saha
Rahul Santhanam	Sylvain Schmitz	Stefan Schneider
Philippe Schnoebelen	Pascal Schweitzer	Juraj Sebej
Peter Selinger	Jeffrey Shallit	Arseny Shur
Rene Sitters	Piotr Skowron	Michał Skrzypczak
William Slofstra	Manuel Sorge	Krzysztof Sornat
Srikanth Srinivasan	Tatiana Starikovskaya	Michelle Strout
Georg Struth	Grégoire Sutre	Tomoyuki Suzuki

Asahi Takaoka	Hisao Tamaki	Isabelle Tellier
Neil Thapen	Szymon Toruńczyk	Patrick Totzke
Gabriella Trucco	Iddo Tzameret	Henning Urbat
Alexander Ushakov	Daniel Valenzuela	Gabriel Valiente
Vinodchandran Variyam	Walter Vogler	Ilya Volkovich
Vojtěch Vorel	Jan Philipp Wächter	Neil Walkinshaw
Armin Weiss	Mathias Weller	Tim Willemse
Ryan Williams	John Wilmes	Joost Winter
James Worrell	Christian Wulff-Nilsen	Lirong Xia
Fan Yang	Florian Yger	Bruno Zanuttini
Rico Zenklusen	Georg Zetzsche	Martin Ziegler
Jens Zumbrägel		

Steering Committee

Juraj Hromkovič	ETH, Switzerland
Antonín Kučera	Masaryk University, Czech Republic (chair)
Jerzy Marcinkowski	University of Wrocław, Poland
Damian Niwiński	University of Warsaw, Poland
Branislav Rovan	Comenius University, Slovakia
Jiří Sgall	Charles University, Czech Republic

How Far Are We From Having a Satisfactory Theory of Clustering?

Shai Ben-David[1]

1 **Universitys of Waterloo, Waterloo, Ontario, Canada**
shai@cs.uwaterloo.edu

Abstract

This is an overview of the invited talk delivered at the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS-2016).

1998 ACM Subject Classification I.5.3 Clustering

Keywords and phrases clustering, theory, algorithm tuning, computational complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.1

Category Invited Talk

1 Overview of the Talk

Unsupervised learning, utilizing the huge amounts of raw data available, is widely recognized as one of the most important challenges facing machine learning nowadays. For supervised tasks, machine learning theory has been successful in several respects; providing significant understanding of machine learning tasks (in terms of the informational and computational resources they require and in providing algorithmic tools to address them), insights about the pros and cons of alternative machine learning paradigms and their parameter settings, and initiating the development of new algorithmic approaches. However, no such successes had been achieved so far for the unsupervised ML domain.

My talk will focus on clustering, arguably the most fundamental unsupervised data processing task. I will discuss two aspects in which theory could play a significant role in guiding the use of clustering tools. The first is model selection - how should a user pick an appropriate clustering tool for a given clustering problem, and how should the parameters of such an algorithmic tool be tuned? In contrast with other common computational tasks, in clustering, different algorithms often yield drastically different outcomes. Therefore, the choice of a clustering algorithm may play a crucial role in the usefulness of an output clustering solution. Just the same, currently there exist no methodical guidance for clustering tool selection for a given clustering task. I will describe some recent proposals aiming to address this crucial lacuna.

The second aspect of clustering that I will address is the computational complexity of computing a cost minimizing clustering (given some clustering objective function). Once a clustering model (or objective) has been picked, the task becomes an optimization problem. While most of the clustering objective optimization problems are computationally infeasible, they are being carried out routinely in practice. This theory-practice gap has attracted significant research attention recently. I will describe some of the theoretical attempts to address this gap and discuss how close do they bring us to a satisfactory understanding of the computational resources needed for achieving good clustering solutions.

Decidable Extensions of MSO

Mikołaj Bojańczyk[1]

1 University of Warsaw, Warsaw, Poland
bojan@mimuw.edu.pl

Abstract

This is an overview of the invited talk delivered at the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS-2016).

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases monadic second-order logic, extensions, decidability, automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.2

Category Invited Talk

1 Overview of the Talk

Büchi's theorem says that it is decidable if a formula of MSO (monadic second-order logic) can be satisfied in an infinite word. Rabin generalised this to infinite trees. These are among the most powerful decidability results in computer science, and many other decidability results can be obtained as corollaries. In my talk, I will discuss how to go beyond these results and what features can be added to MSO so that it remains decidable. The added feature are going to be extra quantifiers, like the "unboundedness" quantifier or a probabilistic "almost surely" quantifier.

In the proofs of Büchi's and Rabin's theorems, the key role is played by automata. In the extensions from my talk, this will also be the case. The automata are going to be nondeterministic devices with new asymptotic acceptance conditions, which go beyond the classical Büchi or parity acceptance conditions.

Optimal Reachability in Weighted Timed Automata and Games*

Patricia Bouyer-Decitre[1]

1 LSV, CNRS & ENS Cachan, Université Paris-Saclay, France
bouyer@lsv.fr

Abstract

This is an overview of the invited talk delivered at the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS-2016).

1998 ACM Subject Classification F.1.1 Models of Computation – D.2.4 Software/Program Verification

Keywords and phrases timed automata, model-checking, optimization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.3

Category Invited Talk

1 Overview of the Talk

Toward the development of more reliable computerized systems, expressive models are designed, targetting application to automatic verification (model-checking). As part of this effort, timed automata have been proposed in the early nineties [2] as a powerful and suitable model to reason about (the correctness of) real-time computerized systems. Timed automata extend finite-state automata with several clocks, which can be used to enforce timing constraints between various events in the system. They provide a convenient formalism and enjoy reasonably-efficient algorithms (*e.g.* reachability can be decided using polynomial space), which explains the enormous interest that they provoked in the community of formal methods. Timed games [4] extend timed automata with a way of modelling systems interacting with external, uncontrollable components: some transitions of the automaton cannot be forced or prevented to happen. The reachability problem then asks whether there is a strategy (or controller) to reach a given state, whatever the (uncontrollable) environment does. This problem can also be decided, in exponential time.

Timed automata and games are not powerful enough for representing quantities like resources, prices, temperature, etc. The more general model of hybrid automata [14] allows for accurate modelling of such quantities using hybrid variables. The evolution of these variables follow differential equations, depending on the state of the system, and this unfortunately makes the reachability problem undecidable, even in the very restricted case of stopwatches (stopwatches are clocks that can be stopped, and hence, automata with stopwatches only are the simplest hybrid automata one can think of).

Weighted (or priced) timed automata [3, 5] and games [15, 1, 9] have been proposed in the early 2000's as an intermediary model for modelling resource consumption or allocation problems in real-time systems (*e.g.* optimal scheduling [6]). As opposed to (linear) hybrid

* This work was partially supported by ERC project EQualIS (308087).

systems, an execution in a weighted timed model is simply one in the underlying timed model: the extra quantitative information is just an observer of the system, and it does not modify the possible behaviours of the system.

In this talk, we will investigate the models of weighted timed automata and games, and we will mostly focus on the important optimal reachability problem: given a target location, we want to compute the optimal (*i.e.* smallest) cost for reaching a target location, and a corresponding strategy. We will survey the main results that have been obtained on that problem, from the primary results of [3, 5, 16, 13, 8, 17, 7] to the most recent developments [11, 10]. We will also mention our new tool TiAMo, which can be downloaded at `https://git.lsv.fr/colange/tiamo`. We will finally show that weighted timed automata and games have applications beyond that of model-checking [12].

References

1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *LNCS*, pages 49–62. Springer, 2001.

4 Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier Science, 1998.

5 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th International Workshop on Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.

6 Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal scheduling using priced timed automata. *ACM Sigmetrics Performancs Evaluation Review*, 32(4):34–40, 2005.

7 Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem. *Formal Methods in System Design*, 31(2):135–175, 2007.

8 Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

9 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.

10 Patricia Bouyer, Maximilien Colange, and Nicolas Markey. Symbolic optimal reachability in weighted timed automata. In *Proc. 28th International Conference on Computer Aided Verification (CAV'16) – Part I*, LNCS. Springer, 2016. To appear.

11 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In *Proc. 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *LIPIcs*, pages 311–324. Leibniz-Zentrum für Informatik, 2015.

12 Patricia Bouyer, Nicolas Markey, Nicolas Perrin, and Philipp Schlehuber. Timed automata abstraction of switched dynamical systems using control funnels. In *Proc. 13th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'15)*, volume 9268 of *LNCS*, pages 60–75. Springer, 2015.

13 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proc. 3rd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3821 of *LNCS*, pages 49–64. Springer, 2005.

14 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

15 Salvatore La Torre, Supratik Mukhopadhyay, and Aniello Murano. Optimal-reachability and control for acyclic weighted timed automata. In *Proc. 2nd IFIP International Conference on Theoretical Computer Science (TCS 2002)*, volume 223 of *IFIP Conference Proceedings*, pages 485–497. Kluwer, 2002.

16 Kim G. Larsen, Gerd Behrmann, Ed Brinksma, Angskar Fehnker, Thomas Hune, Paul Pettersson, and Judi Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 493–505. Springer, 2001.

17 Jacob I. Rasmussen, Kim G. Larsen, and K. Subramani. On using priced timed automata to achieve optimal scheduling. *Formal Methods in System Design*, 29(1):97–114, 2006.

Scale-Free Networks, Hyperbolic Geometry, and Efficient Algorithms

Tobias Friedrich

Hasso Plattner Institute, Potsdam, Germany

Abstract

The node degrees of large real-world networks often follow a power-law distribution. Such scale-free networks can be social networks, internet topologies, the web graph, power grids, or many other networks from literally hundreds of domains. The talk will introduce several mathematical models of scale-free networks (e.g. preferential attachment graphs, Chung-Lu graphs, hyperbolic random graphs) and analyze some of their properties (e.g. diameter, average distance, clustering). We then present several algorithms and distributed processes on and for these network models (e.g. rumor spreading, load balancing, de-anonymization, embedding) and discuss a number of open problems. The talk assumes no prior knowledge about scale-free networks, distributed computing or hyperbolic geometry.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases power-law graphs, scale-free graphs, random graphs, distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.4

Category Invited Talk

1 Short Review of Network Models

There are numerous models for large complex networks. The talk reviews some popular scale-free random graph models. The most cited network model are preferential attachment graphs by Barabási and Albert [2]. A bit more accessible for a formal analysis is the model of graphs with fixed expected degree sequences by Chung and Lu [10]. Both models follow a power-law degree distribution, but show only an extremely small clustering coefficient. Other models like the small-world model by Watts and Strogatz [24] generate local clustering, but do not converge to a power-law degree distribution.

There are a number of variations of the aforementioned models to generate graphs with power-law degree distribution *and* local clustering, but most are very artificial and therefore do not give an explanation *why* large networks typically show both properties. In the last couple of years it has been observed that complex scale-free network topologies with high clustering coefficients emerge naturally from hyperbolic metric spaces [23]. There seems to be a close relationship between hyperbolic geometry and complex networks. This can be explained by observing that the nodes of real-world networks can be often organized hierarchically, in an approximate tree-like fashion. Based on this and other observations, hyperbolic random graphs have been suggested in [23] and experimentally studied in [21]. Boguñá, Papadopoulos, and Krioukov [6] describe how hyperbolic mappings can be used to improve internet routing. Hyperbolic networks seem to combine all desired features of real networks in a natural model.

2 Short Review of Algorithmic Results

The model of Chung and Lu [10] has been studied intensively. It has a giant connected component that contains a linear fraction of the nodes [10] and ultra-short average distances of $\mathcal{O}(\log\log n)$ [11, 12]. Algorithmically, these graphs have been examined in various contexts like information dissemination [14], bootstrap percolation [1], de-anonymization [7], and finding cliques [15]. Rumor spreading has also been studied on the preferential attachment model [13, 9]. Graphs can be generated from both models in linear time [22, 3].

For hyperbolic random graphs, much less is known so far. Besides the power-law degree distribution and high clustering [18, 21], the model generates larger cliques [16], polylogarithmic diameter [19, 17] and ultra-short average distances of order $\mathcal{O}(\log\log n)$ [8]. Hyperbolic random graphs can be generated in linear time [8]. In quasilinear runtime it is also possible to assign hyperbolic coordinates to large real-world graphs such that the hyperbolic metric approximates the graph distance [5]. Depending on the exponent β of the power-law degree distribution, the graphs have comparatively small separators and sublinear treewidth [4]. The model also allows fast bootstrap percolation [20].

References

1 H. Amini and N. Fountoulakis. What I tell you three times is true: Bootstrap percolation in small worlds. In *8th WINE*, pp. 462–474, 2012.

2 A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286: 509–512, 1999.

3 V. Batagelj and U. Brandes. Efficient generation of large random networks. *Physical Review E*, 71:036113, 2005.

4 T. Bläsius, T. Friedrich, and A. Krohmer. Hyperbolic random graphs: Separators and treewidth. In *24th ESA*, 2016.

5 T. Bläsius, T. Friedrich, A. Krohmer, and S. Laue. Efficient embedding of scale-free graphs in the hyperbolic plane. In *24th ESA*, 2016.

6 M. Boguñá, F. Papadopoulos, and D. Krioukov. Sustaining the internet with hyperbolic mapping. *Nature Communications*, 1, 2010.

7 K. Bringmann, T. Friedrich, and A. Krohmer. De-anonymization of heterogeneous random graphs in quasilinear time. In *22nd ESA*, pp. 197–208, 2014.

8 K. Bringmann, R. Keusch, and J. Lengler. Geometric inhomogeneous random graphs. *arXiv preprint arXiv:1511.00576*, 2015.

9 F. Chierichetti, S. Lattanzi, and A. Panconesi. Rumor spreading in social networks. *Theoretical Computer Science*, 412:2602–2610, 2011.

10 F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6:125–145, 2002.

11 F. Chung and L. Lu. The average distance in a random graph with given expected degrees. *Internet Mathematics*, 1:91–113, 2004.

12 S. Dereich, C. Mönch, and P. Mörters. Typical distances in ultrasmall random networks. *Advances in Applied Probability*, 44:583–601, 2012.

13 B. Doerr, M. Fouz, and T. Friedrich. Social networks spread rumors in sublogarithmic time. In *43rd STOC*, pp. 21–30, 2011.

14 N. Fountoulakis, K. Panagiotou, and T. Sauerwald. Ultra-fast rumor spreading in social networks. In *23rd SODA*, pp. 1642–1660, 2012.

15 T. Friedrich and A. Krohmer. Parameterized clique on scale-free networks. In *23rd ISAAC*, pp. 659–668, 2012.

16 T. Friedrich and A. Krohmer. Cliques in hyperbolic random graphs. In *34th INFOCOM*, pp. 1544–1552, 2015.

17 T. Friedrich and A. Krohmer. On the diameter of hyperbolic random graphs. In *42nd ICALP*, pp. 614–625, 2015.

18 L. Gugelmann, K. Panagiotou, and U. Peter. Random hyperbolic graphs: degree sequence and clustering. In *39th ICALP*, pp. 573–585, 2012.

19 M. Kiwi and D. Mitsche. A bound for the diameter of random hyperbolic graphs. In *12th ANALCO*, pp. 26–39, 2015.

20 C. Koch and J. Lengler. Bootstrap percolation on geometric inhomogeneous random graphs. In *43rd ICALP*, 2016.

21 D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá. Hyperbolic geometry of complex networks. *Phys. Rev. E*, 82:036106, 2010.

22 J. C. Miller and A. Hagberg. Efficient generation of networks with given expected degrees. In *8th WAW*, pp. 115–126, 2011.

23 F. Papadopoulos, D. V. Krioukov, M. Boguñá, and A. Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *29th INFOCOM*, pp. 2973–2981, 2010.

24 D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

RNA-Folding - From Hardness to Algorithms

Virginia Vassilevska Williams

Stanford University, USA

Abstract

This is an overview of the invited talk delivered at the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS-2016).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases RNA folding, matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.5

Category Invited Talk

1 Overview of the Talk

A fundamental problem in computational biology is predicting the base-pairing of an RNA secondary structure. Most algorithms for this rely on an algorithm for a simplified version of this problem, RNA-folding, defined as follows: given a sequence S of letters over the alphabet $\{A, U, C, G\}$ where A can only be paired with U and C can only be paired with G, determine the best "folding" of S, i.e. a maximum size *nested* pairing of the symbols of S. For instance, in the sequence $ACUG$ the best pairing is either matching A with U, or matching C with G, but not both as that pairing wouldn't be nested.

A dynamic programming algorithm from 1980 by Nussinov and Jacobson [1] solves the RNA-folding problem on an n letter sequence in $O(n^3)$ time. Despite many efforts, until recently, the best algorithms for RNA-folding only shaved small logarithmic factors over this cubic running time. In this talk I will discuss our recent research on RNA-folding and related problems.

Our first result attempts to explain why it has been so difficult to obtain faster algorithms. We show that if one can solve RNA-folding on n length strings faster than one can currently multiply n by n matrices, then the Clique problem would have surprisingly fast algorithms. The current fastest algorithm to multiply n by n matrices runs in $O(n^{2.373})$ time and the fastest known Clique algorithms use this result. Obtaining an $O(n^{2.36})$ time algorithm for RNA-folding would thus be potentially difficult as it would imply a breakthrough for Clique algorithms and potentially also for matrix multiplication.

While this hardness result is appealing, it does not explain the seeming n^3 barrier. No better hardness seemed possible to us, and thus it became increasingly more plausible that RNA-folding should have a faster algorithm and in fact one using fast matrix multiplication. Indeed, this turned out to be true: we were recently successful in obtaining the first truly subcubic time algorithm for the problem. My talk will strive to give some insights into the hardness result and the new algorithm.

References

1 Ruth Nussinov and Ann B.Jacobson. Fast algorithm for predicting the secondary structure of single-stranded RNA. *Proceedings of the National Academy of Sciences of the United States of America*, 77(11):6309–6313, 1980.

Integer Factoring Using Small Algebraic Dependencies*

Manindra Agrawal[1], Nitin Saxena[2], and Shubham Sahai Srivastava[3]

1 Indian Institute of Technology, Kanpur, INDIA
manindra@cse.iitk.ac.in
2 Indian Institute of Technology, Kanpur, INDIA
nitin@cse.iitk.ac.in
3 Indian Institute of Technology, Kanpur, INDIA
ssahai@cse.iitk.ac.in

Abstract

Integer factoring is a curious number theory problem with wide applications in complexity and cryptography. The best known algorithm to factor a number n takes time, roughly, $\exp(2\log^{1/3} n \cdot \log^{2/3}\log n)$ (*number field sieve*, 1989). One basic idea used is to find two squares, possibly in a number field, that are congruent modulo n. Several variants of this idea have been utilized to get other factoring algorithms in the last century. In this work we intend to explore new ideas towards integer factoring. In particular, we adapt the AKS primality test (2004) ideas for integer factoring.

In the motivating case of semiprimes $n = pq$, i.e. $p < q$ are primes, we exploit the difference in the two Frobenius morphisms (one over $\mathbb{F}_p$ and the other over $\mathbb{F}_q$) to factor n in special cases. Specifically, our algorithm is polynomial time (on number theoretic conjectures) if we know a *small algebraic dependence* between p, q. We discuss families of n where our algorithm is significantly faster than the algorithms based on known techniques.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems, I.1.2 Algorithms

Keywords and phrases integer, factorization, factoring integers, algebraic dependence, dependencies

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.6

1 Introduction

Factoring a positive integer n is the process of finding a positive integer m $(1 < m < n)$ that divides n. Integer factorization has been fascinating mathematicians for centuries [8]. There has been continuous attempts to expand our abilities to factor larger and larger integers (see [13], [2]).

In general, factoring a composite number is widely believed to be a "hard" problem, with no efficient general purpose algorithms known. There are several *special* purpose factoring algorithms which can factor composites efficiently, provided some specific property is satisfied. Some of the algorithms being: Trial division (or Eratosthenes sieve, see [11]), Fermat's factorization [14], Euler's factorization [18][20], Pollard's rho algorithm [22], Pollard's $p-1$ algorithm [21], Williams' $p+1$ algorithm [28], Lenstra's elliptic curve factorization [17], quadratic sieve [9], and the number field sieve [5]. Sieve ideas have been the most successful ones in factoring, see an excellent survey in [24].

* N.S. is supported by DST/SJF/MSA-01/2013-14.

The “hardness” of integer factorization has no known proof, but, the belief hinges only on our inability to factor a general composite efficiently. However this belief is so strong, that the most widely used public key cryptosystems (eg. RSA [4]) are based on this “inherent” difficulty to factorize integers. Such applications in cryptography make integer factorization problem even more interesting. Giving a polynomial time algorithm to factorize any given integer, might result in breaking most widely used cryptosystems. On the other hand, proving (or giving evidence) that no efficient algorithm exists for factoring a general composite would further strengthen the trust on these cryptosystems.

This does not mean that no progress was made in the direction, to come up with a general purpose algorithm. Although there is no algorithm that can factor (even heuristically) all integers in “polynomial time" (i.e. polynomial in the bit-size of the input number), yet there are several algorithms that run in *subexponential* time (i.e. $\exp(O(\log^{\epsilon} n))$ time for $\epsilon < 1$). These are faster than the simple “high school” method (i.e. trial division algorithm, having exponential running time). The fastest general purpose algorithm for factoring a number n, is the general number field sieve (see [15]), with heuristic running time $\exp\left(\left(\sqrt[3]{64/9}+o(1)\right)(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right)$. The other widely used algorithm in practice is the quadratic sieve algorithm [23], having running time $\exp\left((1+o(1))\sqrt{\log n \log\log n}\right)$, which is a modification of Dixon’s algorithm [7], that had a (rigorously provable) running time of $\exp\left((2\sqrt{2}+o(1))\sqrt{\log n \log\log n}\right)$.

In 1997, Peter Shor discovered the first polynomial time algorithm for factoring integers on a *quantum computer* [26]. To factor an integer n, it takes $O((\log n)^2 \log\log n \log\log\log n)$ time[1] on quantum computer and $O(\log n)$ post-processing time on classical computer for converting the output of quantum computer to factors of n. If one day quantum computation becomes feasible for large inputs, then this will have serious implications in cryptography [3].

One common thread in these, increasingly complex, algorithms is the trick of finding two squares in some number field, such that the *difference of the squares*, say $a^2 - b^2$, is a multiple of n. Then we can hope that the factors $(a-b), (a+b)$ would also lead us to the factors of n. The origins of this trick dates back atleast to Fermat, and was also exploited by Gauss, Seelhoff and Kraitchik (see the early history of factoring in [29]). One wonders whether other natural tricks or ideas could be discovered for factoring integers.

In this work we propose a method for factoring semiprimes $n = pq$ (i.e. $p < q$ are primes) using the difference in the Frobenius morphisms over the finite fields $\mathbb{F}_p$ and $\mathbb{F}_q$. We do this by working in a *cyclotomic* ring extension $(\mathbb{Z}/n\mathbb{Z})[\zeta] := \mathbb{Z}[X]/\left(n, \frac{X^r-1}{X-1}\right)$. We pick a random element $u(\zeta) \in (\mathbb{Z}/n\mathbb{Z})[\zeta]$ and compute the exponentiation u^e, for a carefully chosen positive integer e. For example, when $e = n$ we can invoke the Frobenius morphisms to deduce $u(\zeta)^n = u(\zeta^p)^q \pmod p$ and $u(\zeta)^n = u(\zeta^q)^p \pmod q$. A similar line of thought has been explored in [6], where they viewed the problem from the perspective of AKS [1] polynomial. Although no family of n was identified in that work to be particularly good. The asymptotic complexity of the algorithm was also not analyzed, but some supporting experimental data was included.

We identify certain families of n where this idea gives a fast factoring algorithm. Especially, in our main result we pick e corresponding to a *known* algebraic dependency of p and q. In this case, we show that the ring computations in $(\mathbb{Z}/n\mathbb{Z})[\zeta]$ are expected to factorize n. We believe that such computations in the cyclotomic ring have a good chance in further improving the state of the art in factoring. Similar techniques were utilized in [1] to give the

[1] We can shorten this using the soft-Oh notation as $\tilde{O}(\log^2 n)$.

first deterministic poly-time primality test. Moreover, for integer factoring even "heuristic" algorithms that are expected to run in poly-time (in the worst-case) would be of great interest.

Our notion of "small" algebraic dependence and the proof of its existence is captured in the following proposition. We say that a bivariate polynomial $f(X,Y)$ is *nondegenerate* if there appears, with a nonzero coefficient, a monomial X^iY^j in f such that $i \neq j$.

▶ Proposition 1.1 (Small dependency exists). For numbers $d, a < b \in \mathbb{N}$, there exists a degree $\leq d$ nondegenerate integral polynomial $f(X,Y)$ of sparsity 2γ and coefficients c_i's of magnitude at most $b^{d/(\gamma-1)}$ such that: $f(a,b) = \sum_{i=1}^{2\gamma} c_i\, a^{\alpha_i}\, b^{\beta_i} = 0$. (Note that $2\gamma \leq \binom{d+2}{2}$ as $0 \leq \alpha_i + \beta_i \leq d$.)

It is proven in Section 3. Recall that Fermat's factoring algorithm works fast when the primes p, q are really close[2]; formally, when there is an $f(x,y) = y - x - \alpha$, for a small α, such that $f(p,q) = 0$. We generalize the condition of Fermat's factoring algorithm to higher degree dependencies (and with more general coefficients). The above proposition gives the parameters for such f to exist. Our algorithms will require the *knowledge* of such an f (unfortunately, in general, it may be hard to find f given only n).

One such interesting dependence is addressed in Section 4.2. For $n = pq$, $p < q$, we represent q in base p as $q = a_0 + a_1.p + a_2.p^2 + a_3.p^3 + \cdots$. We define the *p^{th} norm of q* as $|q|_p := \prod_i (a_i + 1)$. Given a small bound B on $|q|_p$, our algorithm factors n in time $O(B^2 \log^2 n)$. This immediately gives us a family of n which can be factored efficiently (under certain number theory conjectures) using our algorithm. This family is a natural generalization of the family of numbers ($n = ab$, where $b - a$ is small) that can be factored efficiently using Fermat's factoring algorithm.

Our general approach works in polynomial time, assuming that a suitable dependency is provided (and that certain number theory conjectures hold). The algorithm presented in this paper runs in $\tilde{O}(\gamma^3 d \log^2 n)$ time, where d is the degree bound of the dependency, γ is its sparsity, and n is the number to be factored. Observe that once such a bivariate nondegenerate dependency $f(X,Y)$, of degree d is given, we can easily transform it to get a univariate polynomial $X^d f(X, n/X)$ which has p as a root. Notice, that it is important here that the dependency is nondegenerate. For degenerate dependency of the form $f(X,Y) = \sum_{i\leq d} a_i X^i Y^i$ the substitution $f(X, n/X)$ will give us a number instead of a univariate polynomial, and we could not proceed further.

Now, once we get a univariate polynomial $f' := X^d f(X, n/X)$ which has p as a root, we could simply try to find its integral roots by factoring it using Schönhage's algorithm [25] having time complexity $\tilde{O}(d^4 \cdot (d^2 + \log^2 |f'|))$, where $|f'|$ upper bounds the coefficients in f'. On the other hand, our new approach is sensitive to sparsity γ and tolerates bigger coefficients. So, for dependencies, having 'small' γ and 'large' d, our algorithm will outperform Schönhage's algorithm by several orders. For example, given dependence $f(x,y) = y + c_1 x^d + c_0$, where $|c_1| = |c_0| = n^{O(d)}$, our algorithm will factor it in time $\tilde{O}(d \log^2 n)$, whereas Schönhage's algorithm will take time $\tilde{O}(d^4 \cdot (d^2 + d^2 \log^2 n)) = \tilde{O}(d^6 \log^2 n)$.

The main result established is presented in Section 5. The section presents the algorithm to factor n when a small dependency is provided. The result is summarized by the following theorem.

▶ Theorem 1.2 (Main Result). For an integer $n = p{\cdot}q$ ($p < q$ are primes), given a nondegenerate integral (p,q) dependency of the form $f(X,Y) = \sum_{i=1}^{\gamma} c_i\, X^{\alpha_i}\, Y^{\beta_i}$, where $\forall i$, $0 \leq \alpha_i + \beta_i \leq d$,

[2] Essentially, one tries to find $q - p$ by brute-force.

$|c_i| = n^{O(d)} := A$ we can factor n in $\tilde{O}(\gamma^3 d \log^2 n)$ time. (Assuming Artin's conjecture & 3.)

We also present an alternate analysis of this algorithm in Section 6. This section also generalizes the result for integers of the form $n = p \cdot n'$, where p is a prime smaller than the largest prime factor of n. The following theorem presents the result of that section.

▶ Theorem 1.3. For an integer $n = p \cdot n'$ (where p is a prime smaller than the largest prime factor of n), given a nondegenerate integral (p, n') dependency of the form $f(X,Y) = \sum_{i=1}^{\gamma} c_i\, X^{\alpha_i}\, Y^{\beta_i}$, where $\forall i$, $0 \le \alpha_i + \beta_i \le d$, $|c_i| < n^d$, we can factor n in $\tilde{O}(\mu^3 \cdot \gamma d^4 \log^2 n)$ time. Here $\mu := \sum_i e_i$ for the prime factorization $n = \prod_i p_i^{e_i}$. (Assuming Artin's conjecture & 4.)

Here as well, for $\mu, \gamma = O(1)$ the time complexity is better than that of simply factoring $X^d f(X, n/X)$ by Schönhage's algorithm. Also, the algorithm seems simpler than the sophisticated lattice computations that underlie Schönhage's polynomial factoring algorithm (see [12]).

The paper is organized into following sections: Section 2 talks about the notations and results used in the paper. Section 3 proves the existence of small dependence. In Section 4, we discuss two simple dependencies as motivating examples, and explore the idea of exponentiation in the cyclotomic ring to factor n. Section 5 presents the main result of the paper. An alternate analysis of the algorithm is presented in Section 6.

2 Notation and Preliminaries

This section states the notations and number theory results that we will use later.

Polynomial notation. The form of polynomials that we compute in this work is exponentiation; motivated by the *AKS polynomial* (see [1]) used for primality testing:

$$\mathcal{P} = a(x)^e \pmod{n, x^r - 1}, \quad \text{where } a(x) \text{ is a polynomial.} \tag{1}$$

For technical reasons we will actually work modulo the r-th cyclotomic polynomial $\varphi_r(x)$. Then, we represent exponentiation by the following shorthand notation

$$\mathcal{P} = a(\zeta_r)^e \pmod{n}, \text{ and might drop } r \text{ when clear from the context.} \tag{2}$$

Formally, this arithmetic happens in the ring $(\mathbb{Z}/n\mathbb{Z})[\zeta_r] := \mathbb{Z}[X]/(n, \varphi_r(X))$, where every element can be written as a $(\mathbb{Z}/n\mathbb{Z})$-linear-combination of the monomials $\{X^i \mid 0 \le i \le \varphi(r) - 1\}$, where $\varphi(r)$ is the Euler totient function (also, the degree of the cyclotomic polynomial). This will be our standard representation.

In this paper we assume r to be a prime, mainly, to simplify the analysis since $\varphi_r(x) = (x^r - 1)/(x - 1)$. Also for composite r's the cyclotomic extension is quite well structured. For the basic properties of the cyclotomics see [27, Chap.2].

Artin's conjecture. Emil Artin (1927, see [19]) conjectured: For any non-square $a \in \mathbb{Q}\backslash\{-1\}$ there exist infinitely many primes p such that a is a *primitive root* modulo p, i.e. the multiplicative order $\mathrm{ord}_p(a) = p - 1$.

There has been impressive positive progress towards this conjecture [10]. Moreover, a quantitative version of this conjecture is also believed to be true.

▶ Conjecture (Artin's conjecture, see [19]). For any non-square $a \in \mathbb{Q} \setminus \{-1\}$, the number of primes $p \le x$ with $\mathrm{ord}_p(a) = p - 1$ is asymptotically at least $\mathcal{C}_{Artin} \cdot \pi(x)$. ($\pi(x)$ is the number of primes in the interval $[1, x]$ and $\mathcal{C}_{Artin} = 0.3739558136192\cdots$.)

Frobenius morphism. For a prime p consider the polynomial ring $R := \mathbb{F}_p[X]$ over the finite field $\mathbb{F}_p$. Consider the map $\phi : R \to R$ given by the exponentiation $a(X) \mapsto a(X)^p$. It is easy to see that ϕ is actually a (ring) endomorphism of R, and the trivial[3] automorphism of $\mathbb{F}_p$. In other words, we have the useful identity: $\forall a(X) \in R$, $a(X)^p = a(X^p)$.

Other notations. We use $[n]$ to denote the set $\{1, 2 \cdots, n\}$. The notation $\log_{q,r} p$ is used to denote, the *discrete log*, $\log_q p$ in the field $\mathbb{F}_r$, i.e. it is the exponent $i \in \{0, \ldots, r-2\}$ such that $p = q^i \pmod r$. Here, we assumed that r is a prime, and q is a primitive root modulo r. (We hope to get such an r corresponding to a q as the density of r's is high as per Artin's conjecture.) Bold faced symbols (e.g. $\boldsymbol{\alpha}$) represent vectors. $\mathbb{F}_q[\zeta]$ represents some ring $\mathbb{F}_q[X]/(\varphi_r(X))$.

We recall a useful standard property of cyclotomic polynomials. This is the main reason why Artin's conjecture appears in this work.

▶ Lemma 2.1. Let $q \neq r$ be primes. The integral polynomial $\varphi_r(x) = (x^r - 1)/(x - 1)$ is irreducible over $\mathbb{F}_q$ iff q is a primitive root modulo r.

Proof. Let q generate $\mathbb{F}_r^*$. Wlog assume $r > 2$, as $\varphi_r(x)$ is linear for $r = 2$. Suppose $\varphi_r(x)$ is reducible and has a degree d factor $g(x)$, where $d \in [r-2]$. Let α be a root of $g(x)$ in the (splitting) field $\mathbb{F}_q[x]/(g(x))$. As this is the field $\mathbb{F}_{q^d}$, the multiplicative order $\mathrm{ord}(\alpha)$ will divide $q^d - 1$. Since α is a root of $x^r - 1$, we also have $\mathrm{ord}(\alpha)|r$. Thus, $\mathrm{ord}(\alpha)$ is 1 or r. It cannot be 1 as $q \neq r$. So,

$$\mathrm{ord}(\alpha) = r.$$

$$\begin{aligned} \text{Consequently, } & r \,|\, q^d - 1 \\ q^d &= 1 \pmod r \\ (r-1) \,|\, d & \quad [\because q \text{ generates } \mathbb{F}_r^*] \,. \end{aligned}$$

This contradicts $d \in [r-2]$. Hence, $\varphi_r(x)$ is irreducible modulo q.

For the converse note that $\varphi_r(x)$ being irreducible modulo q, means that it divides $x^{q^{r-1}} - x$, and no other $x^{q^i} - x$ for a smaller i. Equivalently, $r \,|\, q^{r-1} - 1$ and no other $q^i - 1$ for a smaller i. Thus, q generates $\mathbb{F}_r^*$. ◀

3 Existence of small dependencies

Our basic idea is based on the following elementary property of numbers.

Proof for Proposition 1.1. Clearly, $2\gamma \leq \binom{d+2}{2} =: \gamma_0$ which is the upper bound for the number of exponents (α_i, β_i) in f.

Let $A := 2b^{d/(\gamma-1)}$. Consider a set $\mathcal{S}$ of nondegenerate combinations (i.e. $i_1 \neq i_2$ for at least one monomial in each sum),

$$\mathcal{S} := \left\{ \sum_{0 \leq i_1 + i_2 \leq d} \alpha_{i_1,i_2} \cdot a^{i_1} b^{i_2} \;\middle|\; \alpha_{i_1,i_2} \in \mathbb{Z}, |\alpha_{i_1,i_2}| \leq \frac{A}{2}, \text{ at most } \gamma \; \alpha_{i_1,i_2}\text{'s are nonzero} \right\}.$$

[3] Fermat's little theorem (1640).

Then, we have

$$\forall \beta \in \mathcal{S},\ |\beta| \leq \gamma \cdot \frac{A}{2} \cdot b^d .$$

Consider the set $\mathcal{V}$ comprising the coefficient-vectors $\boldsymbol{\alpha}$ corresponding to every element of $\mathcal{S}$. Then the cardinality of $\mathcal{V}$ can be lower bounded (by doing a sum over the possible supports of $\boldsymbol{\alpha}$) as,

$$\begin{aligned} |\mathcal{V}| &\geq \binom{\gamma_0}{\gamma} \cdot A^{\gamma} + \binom{\gamma_0}{\gamma-1} \cdot A^{\gamma-1} + \cdots + \binom{\gamma_0}{1} \cdot A + 1 \\ &> \left(\frac{\gamma_0}{\gamma}\right)^{\gamma} \cdot A^{\gamma} \qquad \text{[Simple binomial estimate]} \end{aligned}$$

Clearly, if $|\mathcal{V}| = |\mathcal{S}|$ is greater than $\max\{|\beta| \mid \beta \in \mathcal{S}\}$, then by the pigeon-hole principle there will be atleast two distinct vectors $\boldsymbol{\alpha}, \boldsymbol{\alpha'} \in \mathcal{V}$ that correspond to the same number $\beta \in \mathcal{S}$. This gives us the desired dependency,

$$0 = \sum_{0 \leq i_1 + i_2 \leq d} (\alpha_{i_1,i_2} - \alpha'_{i_1,i_2}) \cdot a^{i_1} b^{i_2} .$$

Hence, for the desired small dependency it suffices to ensure that,

$$\begin{aligned} |\mathcal{V}| &> \max\{|\beta|\} \\ \text{or } \left(\frac{\gamma_0 A}{\gamma}\right)^{\gamma} &\geq \gamma \cdot \frac{A}{2} \cdot b^d \\ \text{or } A^{\gamma-1} &\geq \left(\frac{\gamma}{\gamma_0}\right)^{\gamma} \cdot \frac{\gamma b^d}{2} \\ \text{or } A &\geq \left(\frac{\gamma}{\gamma_0}\right)^{\gamma/(\gamma-1)} \cdot \left(\frac{\gamma}{2}\right)^{1/(\gamma-1)} \cdot b^{d/(\gamma-1)} \\ \text{or } A &\geq 2 \cdot b^{d/(\gamma-1)} \end{aligned} \tag{3}$$

Clearly, for our A, Equation 3 is satisfied. Hence, the required dependency exists. ◀

Hence, there is a trade-off between the sparsity (γ) and the magnitude (c_i) of the dependency polynomial.

▶ Remark. This bound is not optimal. Eg. if we allow f to have sparsity γ_0 then a slightly better bound of $A = 2b^{d/(\gamma_0-1)}$ can be shown; which for $d = 1$ seems optimal.

For a nonconstant γ, or a superpolynomial coefficient-bound A, it would be quite expensive to search for such a dependency f in general. So, our algorithms would be interesting only for those $n = pq$ where it is relatively easy to find an f such that $f(p, q) = 0$.

4 Motivating Dependencies

In the previous section we have shown that a “small" dependency will always exist (Proposition 1.1). Although, in general this dependency could be hard to find, but in special cases there are several natural dependencies. Some of them have already been witnessed and worked upon. An example of one such naturally occurring dependency is, when the two factors are very close to each other. In other words, for $n = pq$, $q - p = \alpha$, where α is some *small*[4]

[4] The term “small" is used vaguely here. The running time of the algorithm is proportional to α. Hence, we could work with α according to the running time we aim for. For polynomial time algorithm, we want $\alpha = \text{poly} \log n$.

constant. Consequently, in such cases both p and q will be close to $\sqrt{n}$. Hence, to factor n, we can simply use the trial division algorithm, starting from $\sqrt{n}$, which would work efficiently as α is small. A more sophisticated and faster way to factor a number having such a dependency ($q - p = \alpha$) would be to use Fermat's factorization method. We propose a new method to factor numbers having such a dependency.

4.1 Factoring numbers having dependency of the form $q - p = \alpha$

Assuming that we have n and a bound B such that $q - p = \alpha \leq B$ the idea is to pick an element $(x + a) \in (\mathbb{Z}/n\mathbb{Z})[x]/(n, x^r - 1)$ and compute $\mathcal{P} := (x + a)^n$, for an r slightly bigger than B. The hope is that the two (underlying) polynomials, $\mathcal{P}_q = (x^q + a)^p \pmod{q, x^r - 1}$ and $\mathcal{P}_p = (x^p + a)^q \pmod{p, x^r - 1}$ would have different *supports* (i.e. there is a monomial x^i, $i \in [0, 1, \cdots, r - 1]$, that appears with zero coefficient in exactly one of the polynomials[5]). We can clearly see, that $r \leq q$ is the trivial upper bound. But we can likely improve this upper bound further.

For $r < p < q$ it seems likely that for most a's, $(x + a)^p \pmod{q, x^r - 1}$ will have each of the r monomials (i.e. x^i, $i \in [0, 1, \cdots, r - 1]$) appearing with nonzero coefficient[6]. We pose this formally.

▶ Conjecture 1. For primes $p < q$, $1 \leq r < p$ and a random $a \in \mathbb{Z}/q\mathbb{Z}$, $(x+a)^p \pmod{q, x^r - 1}$ is full support with high (i.e. constant) probability.

The rationale for this conjecture is that we expect $(x + a)^p$ to be a "random" element in the cyclotomic ring. So, it will be rare that there is a zero coefficient in its standard representation.

On the other hand $(x + a)^q \pmod{p, x^r - 1}$, for $r \geq 2B + 3$, has proper support as we now show.

▶ Theorem 4.1. For primes $p < q$ and $r \geq 2(q - p) + 3$, $(x + a)^q \pmod{x^r - 1, p}$ is proper support.

Proof. Consider the polynomial,

$$\begin{aligned}
\mathcal{P}_p &= (x + a)^q \pmod{p} \\
&= (x + a)^p (x + a)^{q-p} \pmod{p} \\
&= (x^p + a)(x + a)^{q-p} \pmod{p} \\
&= \underbrace{(x^p)(x + a)^{q-p}}_{\text{Sparsity} \leq q-p+1} \quad + \quad \underbrace{a \cdot (x + a)^{q-p}}_{\text{Sparsity} \leq q-p+1} \pmod{p}\,.
\end{aligned}$$

Hence, Sparsity$(\mathcal{P}_p) \leq 2(q - p + 1)$. So, taking $r \geq 2(q - p + 1) + 1$ will ensure that atleast one monomial in $(x + a)^q \pmod{x^r - 1, p}$ has the zero coefficient. ◀

These observations motivate the following algorithm.

[5] It is easy to see that this implies that one of the coefficients in $\mathcal{P}$ will share a nontrivial gcd with n.
[6] Such a polynomial we call *full* support, and its opposite is *proper* support.

Algorithm 1 Factoring Integer : $FAC_1(n, B)$

```
Require: Odd $n = pq$ ($p < q$ are primes) and a parameter $B \geq (q - p)$.
1: $r \leftarrow 2$
2: while $r \leq 2B + 3$ and $n$ is not factored do
3:     Choose a random number $a < n$
4:     Compute $\mathcal{P} = (x + a)^n \pmod{x^r - 1, n}$
5:     Take gcd of $n$ with $ra$, and with the coefficients of $\mathcal{P}$.
6:     if $n$ is factored then
7:         return $factor$
8: return 0
```

Time complexity. The polynomial computation in step 4, takes time $\tilde{O}(r \log^2 n)$ using fast arithmetic. Taking GCD in step 5, takes similar time. Hence, the overall time complexity of the algorithm is $\tilde{O}(B^2 \log^2 n)$. Note that it is a probabilistic algorithm based on Conjecture 1. It can be seen as an alternative (albeit slower) to Fermat's factoring algorithm.

4.2 Bound based on p^{th} norm of q

This subsection discusses a more general, yet natural, dependency and presents the algorithm to factor n in such cases.

Let us represent q in base p (so that a_i's are in $[0, \cdots, p-1]$),

$$q = a_0 + a_1.p + a_2.p^2 + a_3.p^3 + \cdots .$$

We define the p^{th} *norm of* q as $|q|_p := \prod_i (a_i + 1)$. It is defined as a 'measure' for the size of the coefficients in base p representation of q.

Can we factor $n = pq$ (primes $p < q$) if we have an upper bound B on $|q|_p$? We can generalize the methods of the last section.

By Conjecture 1 we expect $(x + a)^p \pmod{x^r - 1, q}$ to be full support, for random a. The other modulus is covered by the following simple observation.

▶ Theorem 4.2. For primes $p < q$ and $r > |q|_p$, $(x + a)^q \pmod{x^r - 1, p}$ is proper support.

Proof. By using the base-p representation of q we have,

$$\begin{aligned}
(x + a)^q &= (x + a)^{a_0 + a_1.p + a_2.p^2 + a_3.p^3 + \cdots} \\
&= \prod_i (x + a)^{a_i.p^i} \\
&= \prod_i (x^{p^i} + a)^{a_i} \pmod{p} \\
\therefore \text{Sparsity}((x + a)^q \mod p) &\leq \prod_i (a_i + 1) \\
&= |q|_p .
\end{aligned}$$

Hence, for $r > |q|_p$, $(x + a)^q \pmod{x^r - 1, p}$ is proper support. ◀

Remark. For dependency of the form $q - p = \alpha$, where $0 < \alpha < p$, the p^{th} norm of q is $2(\alpha + 1)$. Hence, we get a natural generalization of numbers n that are good for Fermat factorization.

These observations again motivate the following algorithm.

Algorithm 2 Factoring Integer : $FAC_2(n, B)$

```
Require: Odd $n = pq$ ($p < q$ are primes) and a parameter $B > |q|_p$.
1: $r \leftarrow 2$
2: while $r \leq B$ and $n$ is not factored do
3:     Choose a random number $a < n$
4:     Compute $\mathcal{P} = (x + a)^n \pmod{x^r - 1, n}$
5:     Take gcd of $n$ with $ra$, and with the coefficients of $\mathcal{P}$.
6:     if $n$ is factored then
7:         return $factor$
8: return 0
```

Time complexity. The overall time complexity of the algorithm is $\tilde{O}(B^2 \log^2 n)$, as in the previous subsection. Note that it is a probabilistic algorithm based on Conjecture 1. It can be seen as a natural generalization (albeit slower) of Fermat's factoring algorithm.

4.3 Relaxing conjecture 1

In the previous subsections the proofs of factoring depend on Conjecture 1. In this section we relax the conjecture; which might make it easier to prove.

The point is that we just need to prove, that for a random $a(x)$, with high probability there is a difference in the supports of the two polynomials:

$$\begin{aligned} &a(x^p)^q \pmod{x^r - 1, p} \quad \text{and,} \\ &a(x^q)^p \pmod{x^r - 1, q} \end{aligned} \tag{4}$$

in the case when $r > |q|_p$. The rationale is again that the first polynomial is proper support, while the second polynomial is likely to have a support *different* from the first.

▶ Conjecture 2. For primes $p < q$, $r > |q|_p$ and a random $a(x) \in (\mathbb{Z}/n\mathbb{Z})[x]/(x^r - 1, n)$, the two polynomials in Equation 4 have, with high probability, different support.

It can be seen that based on this conjecture, an algorithm similar to Algorithm 2 can be designed to factor n (in probabilistic time $\tilde{O}(B^2 \log^2 n)$).

5 General dependencies

The previous section addressed dependencies of specific forms. In this section, we move to the case of more general dependencies between the two factors. For $n = pq$, primes $p < q$, we consider a nondegenerate dependency $f(x, y)$ of degree bound d with at most γ nonzero coefficients. So, $0 = f(p, q) = \sum_{i=1}^{\gamma} c_i\, p^{\alpha_i}\, q^{\beta_i} = 0$, where $\forall i, 0 \leq \alpha_i + \beta_i \leq d$, $|c_i| = n^{O(d)}$. Proposition 1.1 gives the almost optimal parameters for its existence in general.

When we are given n and f, our idea is to compute AKS exponentiation (Eqn.2) in a cyclotomic ring extension over $\mathbb{Z}/n\mathbb{Z}$ and try distinguishing the two Frobenius morphisms. We give the details in the form of an algorithm and then the proof. The key step will be the computation of an expression $\prod_{i=1}^{\gamma} a(\zeta^{p^{\alpha_i - \beta_i}})^{c_i n^{\beta_i}}$, for a random element $a(\zeta)$. Note that modulo p it is the same as exponentiation by $\sum_{i=1}^{\gamma} c_i p^{\alpha_i} q^{\beta_i} = 0$. Also, note that $p^{\alpha_i - \beta_i}$ exists modulo r, when r and p are coprime.

Algorithm 3 Factoring Integer : $FAC_3(n, f)$

```
Require: Odd $n = pq$ ($p < q$ are primes), and a nondegenerate dependency $f = \sum_{i=1}^{\gamma} c_i x^{\alpha_i} y^{\beta_i}$, where $\forall i, 0 \le \alpha_i + \beta_i \le d$, $|c_i| = n^{O(d)}$.
1: Choose a random prime $r \le 10\gamma \log \gamma$ and verify that $\gcd(r, n) = 1$.
2: for $t \in [r-1]$ do
3:   $count \leftarrow 0$
4:   while $count < 5 \log\log n^r$ do
5:     Choose a random element $a(\zeta) := a(x) \in \mathbb{Z}[x]/(\varphi_r(x), n)$.
6:     Compute $\mathcal{P} := \prod_{i=1}^{\gamma} a(\zeta^{t^{\alpha_i - \beta_i}})^{c_i n^{\beta_i}}$.
7:     Take gcd of $n$ with the coefficients of $\mathcal{P}$.
8:     if $n$ is factored then
9:       return $factor$
10: return 0
```

To study this algorithm we would need a qualitative conjecture about the distribution of discrete logarithms.

▶ Conjecture 3. For a fixed p, q, f as before and $R > 10\gamma$, the function $\log_{q,r} p$ takes almost random values e as we vary $r \in [R]$ such that, with a constant probability,

$$\sum_{i=1}^{\gamma} c_i\, p^{\beta_i}\, q^{e\alpha_i + (1-e)\beta_i} \neq 0 \left(\bmod \frac{q^{r-1}-1}{q-1}\right).$$

The rationale for this conjecture is that as we vary r in a range bigger than $[\gamma]$ we expect e to be "random" enough so that the two γ-dimensional vectors $\left(c_i p^{\beta_i} \mid i \in [\gamma]\right)$ and $\left(q^{e\alpha_i+(1-e)\beta_i} \mid i \in [\gamma]\right)$ are not orthogonal $\pmod{(q^{r-1}-1)/(q-1)}$. One necessary condition for this is: $\{e\alpha_i + (1-e)\beta_i \mid i \in [\gamma]\}$ be a set of distinct functions in e, with at least one of them being nontrivially dependent on e. The *distinctness* holds because $e\alpha_i + (1-e)\beta_i = e\alpha_j + (1-e)\beta_j$ iff $(\alpha_i, \beta_i) = (\alpha_j, \beta_j)$ iff $i = j$. (*Note :* We use that, for some i, $\alpha_i \neq \beta_i$ as f is nondegenerate.)

Now we are ready to prove the correctness of the algorithm.

Proof for Theorem 1.2. By Artin's conjecture we can deduce that we will get a prime r, with constant probability, such that: q generates the unit group of $\mathbb{F}_r$. In this case Lemma 2.1 asserts that $\varphi_r(x)$ is irreducible over $\mathbb{F}_q$. Hence, $\mathbb{F}_q[\zeta] := \mathbb{F}_q[x]/(\varphi_r(x))$ is a field.

We are interested in the iteration when the variable t equals $p \pmod r$. Then we can write,

$$\mathcal{P} = \prod_{i=1}^{\gamma} a(\zeta^{p^{\alpha_i-\beta_i}})^{c_i n^{\beta_i}}. \tag{5}$$

Going modulo p, and using the "first" Frobenius morphism, we get:

$$\begin{aligned}\mathcal{P} &= \prod_{i=1}^{\gamma} a(\zeta)^{c_i p^{\alpha_i} q^{\beta_i}} \pmod p\\ &= a(\zeta)^{\sum_{i=1}^{\gamma} c_i p^{\alpha_i} q^{\beta_i}} \pmod p \;= a(\zeta)^0 \pmod p\\ &= 1 \pmod p.\end{aligned} \tag{6}$$

Now let $e := \log_{q,r} p$. So, we can replace p with q^e in Equation 5, to get

$$\mathcal{P} = \prod_{i=1}^{\gamma} a(\zeta^{q^{e(\alpha_i-\beta_i)}})^{c_i n^{\beta_i}}.$$

Going modulo q, and using the "second" Frobenius morphism, we get:

$$\begin{aligned}\mathcal{P} &= \prod_{i=1}^{\gamma} a(\zeta)^{c_i n^{\beta_i} q^{e(\alpha_i-\beta_i)}} \pmod q \\ &= \prod_{i=1}^{\gamma} a(\zeta)^{c_i p^{\beta_i} q^{e\alpha_i+(1-e)\beta_i}} \pmod q \\ &= a(\zeta)^{\sum_{i=1}^{\gamma} c_i p^{\beta_i} q^{e\alpha_i+(1-e)\beta_i}} \pmod q \qquad (7)\end{aligned}$$

Let us call the exponent $m := \sum_{i=1}^{\gamma} c_i\, p^{\beta_i}\, q^{e\alpha_i+(1-e)\beta_i}$.

If we can show that $a(\zeta)^m \notin \mathbb{F}_q$ then, by Equation 6, we get different supports in the polynomials $\mathcal{P} \pmod p$ and $\mathcal{P} \pmod q$. This means that step 7 would factor n. So, it suffices to ensure that $a(\zeta)^{m(q-1)} \neq 1 \pmod q$, in other words, the multiplicative order of $a(\zeta)$ in the field $\mathbb{F}_q[\zeta]$, denoted $\mathrm{ord}(a(\zeta), \mathbb{F}_q[\zeta])$ satisfies:

$$\mathrm{ord}(a(\zeta), \mathbb{F}_q[\zeta]) \not| \, m(q-1)\,. \qquad (8)$$

From step 5 (of the algorithm) we can treat $a(\zeta)$ as a random element in $\mathbb{F}_q[\zeta]$. From the initial discussion we have that $\mathbb{F}_q[\zeta]$ is the field $\mathbb{F}_{q^{r-1}}$. From this we can estimate the probability of $a(\zeta)$ having the largest multiplicative order.

▶ Claim 5.1. $\mathrm{ord}(a(\zeta), \mathbb{F}_q[\zeta]) = (q^{r-1}-1)$, with probability at least $\frac{1}{3\log\log(q^{r-1}-1)}$, when $r > 7$.

Proof. See full version of the paper. ◀

Thus, the repetitions in step 4 ensure that with a high probability we will pick a generator $a(\zeta)$ of $\mathbb{F}_q[\zeta]$. Now Equation 8 can be rewritten as:

$$\sum_{i=1}^{\gamma} c_i\, p^{\beta_i}\, q^{e\alpha_i+(1-e)\beta_i} \neq 0 \left(\mathrm{mod}\ \frac{q^{r-1}-1}{q-1}\right).$$

Conjecture 3 ensures this with high probability (for a random r). Hence, step 7 will factor with high (i.e. constant) probability.

Time Complexity. The 'for' loop of Step 2-9, runs for $r-1 = \tilde{O}(\gamma)$ times. The 'while' loop in Step 4-9, runs $O(\log\log(n^r)) = \tilde{O}(1)$ times.

The polynomial computation in step 6 is the expensive part. We would use repeated squaring and fast ring arithmetic. It multiplies γ many factors. The exponent of each factor can be bounded by An^d, so, by repeated squaring it takes $O(d\log n + \log A) = O(d\log n)$ time ($\because A := n^{O(d)}$). Also, in each step of repeated squaring there will be two polynomials multiplied in the cyclotomic ring; we can compute the product in $\tilde{O}(r\log n)$ time. Hence, step 6 takes $\tilde{O}(\gamma \cdot d\log n \cdot r\log n) = \tilde{O}(\gamma^2 d\log^2 n)$ time.

So, the overall time complexity of the Algorithm 3 is $\tilde{O}(\gamma^3 d\log^2 n)$.

◀

Clearly, the running time of the algorithm depends on the sparsity. For sparse dependency f, i.e. $\gamma = \tilde{O}(1)$, the running time becomes $\tilde{O}(d\log^2 n)$ which is only linear in d. If the given dependency has sparsity $\gamma = O(d^{1.6})$ then the running time is a much slower $\tilde{O}(d^{5.8}\log^2 n)$, but it is a simple algorithm and still faster than the known methods.

6 Alternate Analysis

In this section we present an alternate analysis and a corresponding algorithm to factor n. The algorithm presented is just a slightly modified version of Algorithm 3, and it will not need n/p to be a prime. The conjecture that our analysis relies on will be different from Conjecture 3.

Algorithm 4 Factoring Integer : $FAC_4(n, f, \mu)$

```
Require: Odd $n = pn'$ (prime $p$ is not the largest prime factor $q$ of $n$), and a nondegenerate
    $(p, n')$ dependency $f = \sum_{i=1}^{\gamma} c_i x^{\alpha_i} y^{\beta_i}$ where $\forall i, 0 \le \alpha_i + \beta_i \le d$, $|c_i| \le n^d$. Let $\mu := \sum_i e_i$
    for the prime factorization $n = \prod_i p_i^{e_i}$.
 1: $r \leftarrow 7\mu d$.
 2: while $r \le 10\mu d \log(d+1)$ do
 3:     Choose the next prime $r$, and verify that $\gcd(r, n) = 1$.
 4:     for $t$ in range $[r-1]$ do
 5:         Choose a random element $a(\zeta) := a(x) \in \mathbb{Z}[x]/(\varphi_r(x), n)$.
 6:         Compute $\mathcal{P} := \prod_{i=1}^{\gamma} a(\zeta^{t^{\alpha_i - \beta_i}})^{c_i n^{\beta_i}}$.
 7:         Take gcd of $n$ with the coefficients of $\mathcal{P}$.
 8:         if $n$ is factored then
 9:             return $factor$
10: return 0
```

To study this algorithm we will need a conjecture about discrete logarithm.

▶ Conjecture 4. For f, p, q, d, μ as before, there exists a prime $r \in [7\mu d, 10\mu d \log(d+1)]$ such that: $\mathrm{ord}_r(q) = r - 1$, $e := \log_{q,r} p < \frac{r}{2d}$ and $f(q^e, n/q^e) \neq 0$.

The rationale behind this conjecture is Artin's conjecture together with the feeling that the function $\log_{q,r} p$ should take "random" values in $\{0, \ldots, r-1\}$, in particular, values as small as $\frac{r}{2d}$. Also, since the interval is large enough we expect to get several such (r, e); one of these q^e is expected to not be a root of $f(X, n/X)$.

We now state our theorem.

Proof for Theorem 1.3. See full version of the paper. ◀

Given a sparse dependence of degree d, (small or constant γ and μ) our algorithm's performance is better than Schönhage's univariate polynomial factoring algorithm.

7 Conclusion

We initiate a new factoring idea using the AKS-type cyclotomic computation [1]. It uses the two Frobenius morphisms and we have been able to analyze it for specific families of n (based on some "reasonable" conjectures). It is a simple algorithm and, in special cases, it performs better than applying the previously known techniques. The outstanding question is what do we do when there is no dependency $f(x, y)$ readily available for n?

In this (general) case we could compute several (say, poly($\log n$)-many) AKS-type polynomials

$$S := \{a(\zeta_r)^e \mid r, a, e \text{ carefully chosen given } n\}$$

and try to apply easy algebraic operations on S. For example, view S as a lattice generator and apply the famous LLL basis reduction algorithm on it [16]. Or, compute other linear algebra operations on S. Do these operations lead us to a factor of n ?

References

1 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Math*, 160(2):781–793, 2004.

2 Shi Bai, Pierrick Gaudry, Alexander Kruppa, Emmanuel Thome, and Paul Zimmermann. Factorization of RSA-220 with CADO-NFS. 2016.

3 Daniel Julius Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

4 Dan Boneh et al. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

5 Joe Peter Buhler, Hendrik Willem Lenstra Jr, and Carl Pomerance. Factoring integers with the number field sieve. In *The development of the number field sieve*, pages 50–94. Springer, 1993.

6 Yingpu Deng and Yanbin Pan. An algorithm for factoring integers. Cryptology ePrint Archive, Report 2012/097, 2012.

7 John D Dixon. Asymptotically fast factorization of integers. *Mathematics of computation*, 36(153):255–260, 1981.

8 Carl Friedrich Gauss. *Disquisitiones Arithmeticae.* 1801. Article 329.

9 Joseph Gerver. Factoring large numbers with a quadratic sieve. *Mathematics of Computation*, 41(163):287–294, 1983.

10 Rajiv Gupta and Maruti Ram Murty. A remark on artin's conjecture. *Inventiones mathematicae*, 78(1):127–130, 1984.

11 F.R.S. Horsley, Rev. Samuel. The sieve of eratosthenes. being an account of his method of finding all the prime numbers. *Philosophical Transactions (1683-1775)*, 62:327–347, 1772.

12 Ravi Kannan. Algorithmic geometry of numbers. *Annual review of computer science*, 2(1):231–267, 1987.

13 Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Klaas Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter Lawrence Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology–CRYPTO'10*, pages 333–350. 2010.

14 R Sherman Lehman. Factoring large integers. *Mathematics of Computation*, 28(126):637–646, 1974.

15 Arjen Klaas Lenstra, Hendrik Willem Lenstra Jr., Mark Steven Manasse, and John M. Pollard. The number field sieve. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, pages 564–572, 1990.

16 Arjen Klaas Lenstra, Hendrik Willem Lenstra, and Lászlo Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

17 Hendrik Willem Lenstra Jr. Factoring integers with elliptic curves. *Annals of mathematics*, pages 649–673, 1987.

18 James McKee. Turning euler's factoring method into a factoring algorithm. *Bulletin of the London Mathematical Society*, 28(133):351–355, 1996.

19 Pieter Moree. Artin's primitive root conjecture—a survey. *INTEGERS*, 10(6):1305–1416, 2012.

20 Oystein Ore. *Number theory and its history.* Courier Corporation, 2012.

21 John M Pollard. Theorems on factorization and primality testing. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 76 of *Cambridge Univ Press*, pages 521–528, 1974.

22 John M Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

23 Carl Pomerance. The quadratic sieve factoring algorithm. In *Advances in cryptology*, pages 169–182, 1985.

24 Carl Pomerance. A tale of two sieves. *Biscuits of Number Theory*, 85, 2008.

25 Arnold Schönhage. Factorization of univariate integer polynomials by diophantine approximation and improved basis reduction algorithm. *ICALP*, 172:436–447, 1984.

26 Peter Williston Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

27 Lawrence Clinton Washington. *Introduction to cyclotomic fields*, volume 83. Springer, 2012.

28 Hugh Cowie Williams. A $p + 1$ method of factoring. *Mathematics of Computation*, 39(159):225–234, 1982.

29 Hugh Cowie Williams and Jeffrey Outlaw Shallit. Factoring integers before computers. *Mathematics of computation*, 48:481–531, 1994. (1943-1993, Fifty Years of Computational Mathematics (W. Gautschi, ed.), Proc. Sympos. Appl. Math.).

Routing with Congestion in Acyclic Digraphs

Saeed Akhoondian Amiri[1], Stephan Kreutzer[*2], Dániel Marx[†3], and Roman Rabinovich[4]

1 **Technical University Berlin, Berlin, Germany**
`saeed.amiri@tu-berlin.de`
2 **Technical University Berlin, Berlin, Germany**
`stephan.kreutzer@tu-berlin.de`
3 **Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary**
`dmarx@cs.bme.hu`
4 **Technical University Berlin, Berlin, Germany**
`roman.rabinovich@tu-berlin.de`

Abstract

We study the version of the k-disjoint paths problem where k demand pairs $(s_1, t_1), \ldots, (s_k, t_k)$ are specified in the input and the paths in the solution are allowed to intersect, but such that no vertex is on more than c paths. We show that on directed acyclic graphs the problem is solvable in time $n^{O(d)}$ if we allow congestion $k - d$ for k paths. Furthermore, we show that, under a suitable complexity theoretic assumption, the problem cannot be solved in time $f(k)n^{o(d/\log d)}$ for any computable function f.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Algorithms, Disjoint Paths, Congestion, Acyclic Digraphs, XP, W[1]-hard

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.7

1 Introduction

The k-disjoint paths problem and related routing problems are among the central problems in combinatorial optimisation. In the most basic variant of the k-disjoint paths problem, a graph G is given with k pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices and the task is to find k pairwise vertex-disjoint paths linking each s_i to its corresponding target t_i.

The problem is well known to be NP-complete [14]. On undirected graphs with a fixed number k of source/terminal pairs, Robertson and Seymour proved in their monumental graph minor series [21] that the problem is polynomial-time solvable. In fact, they showed that it is fixed-parameter tractable with parameter k: it can be solved in cubic time for every fixed value of k.

For directed graphs, the problem is computationally much harder. Fortune et al. [15] proved that it is already NP-complete for only $k = 2$ source/terminal pairs. In particular,

[*] The research of Saeed Amiri, Stephan Kreutzer and Roman Rabinovich has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No 648527).

[†] The research of Dániel Marx was supported by ERC Starting Grant PARAMTIGHT (No. 280152) and OTKA grant NK105645.

this also implies that it is not fixed-parameter tractable on directed graphs. Following this result a lot of work has gone into establishing more efficient algorithms on restricted classes of digraphs.

Fortune et al. [15] showed that the problem can be solved in time $n^{O(k)}$ on acyclic digraphs, that is, it is polynomial-time for every fixed k. However, as proved by Slivkins [22], the problem is $W[1]$-hard on acyclic digraphs, and therefore unlikely to be fixed-parameter tractable. On the other hand, Cygan et al. [11] proved that the problem is fixed-parameter tractable with parmeter k when restricted to planar digraphs. Related to this, Amiri et al. [1] proved that the problem remains NP-complete even in upward planar graphs, but admits a single exponential fixed-parameter algorithm.

Disjoint paths problems have also been studied intensively in the area of approximation algorithms, both on directed and undirected graphs (see, e.g., [9, 18, 2, 5, 8, 4, 6, 10, 7]). The goal is, given an input graph G and demands $(s_1, t_1), \dots, (s_k, t_k)$ to *route* as many pairs as possible in polynomial time. There are many variations what it means for a pair to be routable. In particular, a problem studied intensively in the approximation literature is a relaxed version of disjoint paths where the paths are no longer required to be fully disjoint. Instead, they may intersect but every vertex of the graph is allowed to be contained in at most c paths, for some fixed constant c. This is called *congestion c routing*. In particular, the well-linked decomposition framework developed in [10] for undirected graphs and later generalised to digraphs in [7] has proved to be very valuable for obtaining good approximation algorithms for disjoint paths problems on planar graphs and digraphs.

In this paper, we are interested in exact solutions for high congestion routing on acyclic digraphs. More precisely, we study the following problem.

▶ **Definition 1.** 1. Let G be a digraph and let $I := \{(s_1, t_1), \dots, (s_k, t_k)\}$ be a set of pairs of vertices. Let $c \geq 1$. A *c-routing* of I is a set $\{P_1, \dots, P_k\}$ of paths such that, for all $1 \leq i \leq k$, path P_i links s_i to t_i and no vertex $v \in V(G)$ appears in more than c paths from $\{P_1, \dots, P_k\}$.
2. Let $k, c \geq 1$. In the (k, c)-Congestion Routing problem, a digraph G is given in the input together with a set $I := \{(s_1, t_1), \dots, (s_k, t_k)\}$ of k pairs of vertices (the demands); the task is to decide whether there is a c-routing of I in G.

We consider (k, c)-Congestion Routing on acyclic digraphs. First, it is not very difficult to show that, for every fixed $c \geq 1$, we can generalise the $n^{O(1)}$ time algorithm of Fortune et al. [15] to (k, c)-Congestion Routing. By revisiting the W[1]-hardness proof of Slivkins [22] and making appropriate modifications, we can establish that the problem remains W[1]-hard for every fixed congestion $c \geq 1$. Moreover, by doing the proof in a more modern way (reducing from general subgraph isomorphism instead of maximum clique and invoking a lower bound of Marx [20]), we can show that the $n^{O(k)}$ time algorithm is essentially best possible with respect to the exponent of n. This lower bound is under the Exponential-Time Hypothesis (ETH), which can be informally stated as n-variable 3Sat cannot be solved in time $2^{o(n)}$ (see [16, 19, 12] for more background).

▶ **Theorem 2.** *For any fixed integer $c \geq 1$, (k, c)-*Congestion Routing *is* W[1]*-hard parameterised by k and, assuming ETH, cannot be solved in time $f(k)n^{o(k/\log k)}$ for any computable function f.*

Intuitively, one can expect the problem to become simpler if c is almost as large as k: after all, the problem is trivial if $c \geq k$. Therefore, we study the complexity of the problem in settings close to this extreme case. The main algorithmic result of this paper is to show that for any fixed value of $d \geq 1$, the $(k, k-d)$-Congestion Routing problem can be solved in time $n^{O(d)}$. That is, the exponent of the polynomial bounding the running time of the algorithm only depends on d but not on the number k.

▶ **Theorem 3.** *For every fixed $d \geq 1$ and for all $k \geq 1$ the $(k, k-d)$-*Congestion Routing *problem on acyclic digraphs can be solved in time $n^{O(d)}$.*

A simple corollary of Theorem 2 shows that $(k, k-d)$-Congestion Routing is unlikely to be fixed-parameter tractable and the running time of Theorem 3 essentially cannot be improved (assuming ETH). Observe that if we set $d := k-1$, then $(k, k-d)$-Congestion Routing is simply the standard k-disjoint path problem, thus any algorithmic result for $(k, k-d)$-Congestion Routing parameterised by d would imply the essentially same algorithmic result for the fully disjoint version parameterised by k.

▶ **Corollary 4.** $(k, k-d)$-Congestion Routing *is* W[1]*-hard parameterised by d (if k is part of the input) and, assuming ETH, cannot be solved in time $f(k)n^{o(d/\log d)}$ for any computable function f.*

Organisation. The paper is organised as follows. In Section 3 we fix some notation and prove our main algorithmic result. The corresponding lower bound is then proved in Section 4.

2 Preliminaries

We review basic notation and concepts of graph theory needed in the paper. We refer to [13, 3] for background.

Let G be a digraph. We write $V(G)$ and $E(G)$ for its set of vertices and edges, respectively. We assume that there is no edge with the same head and tail, i.e. there are no loops in the digraphs we consider in this paper. If $(u, v) \in E(G)$ is an edge, then u is its *tail* and v is its *head*. G is *simple* if there are no two distinct edges which have the same tail and the same head. Otherwise we call G a *multi digraph*.

A *path* P in a digraph G is determined by a sequence $(v_1, \dots, v_\ell)$ of vertices such that $v_i \neq v_j$ for all $1 \leq i < j \leq \ell$ and $(v_i, v_{i+1}) \in E(G)$ for all $1 \leq i < \ell$. We write $E(P)$ for the set $\{(v_i, v_{i+1}) : 1 \leq i \leq \ell - 1\}$ of edges appearing in P and $V(P)$ for the set $\{v_1, \dots, v_\ell\}$ of vertices. We say that P *links* v_1 to v_ℓ.

Two paths P_1 and P_2 are *edge disjoint* if $E(P_1) \cap E(P_2) = \emptyset$.

3 A polynomial-time algorithm on acyclic digraphs

In this section we prove the first main result of this paper, Theorem 3, which we repeat here for convenience.

Theorem 3. *For every fixed $d \geq 1$, the $(k, k-d)$-*Congestion Routing *problem on acyclic digraphs can be solved in time $n^{O(d)}$.*

We first need some additional notation and prove some auxiliary lemmas.

▶ **Definition 5.** Let G be a digraph and let $\mathcal{L}$ be a set of paths in G. For every $v \in V(G)$ we define the *congestion* of v with respect to $\mathcal{L}$ as the number of paths in $\mathcal{L}$ containing v.

The following lemma provides a simple extension of the algorithm from [15] for disjoint paths in acyclic digraphs.

▶ **Lemma 6.** *On acyclic digraphs G the (k,c)-*Congestion Routing *probem can be solved in time $n^{O(k)}$, where $n := |G|$.*

Proof. In [15], Fortune et al. proved that the k-disjoint paths problem can be solved in time $n^{O(k)}$ on any n-vertex acyclic digraph G.

Let G, $(s_1,t_1),\dots,(s_k,t_k)$ and c be given. We construct a new digraph H with $V(H) := V(G) \times \{1,\dots,c\}$ and $E(H) := \{\big((u,i),(v,j)\big) : (u,v) \in E(G), 1 \leq i,j \leq c\}$.

Then H contains k pairwise vertex disjoint paths $P_1,\dots,P_k$ such that P_i links $(s_i,1)$ to $(t_i,1)$ if, and only if, there is a positive solution to the (k,c)-Congestion Routing Problem on G. By the algorithm in [15] we can decide whether the paths $P_1,\dots,P_k$ exist in H in time $|V(H)|^{O(k)}$ and hence in time $(c \cdot n)^{O(k)} = n^{O(k)}$ as $c \leq n$. ◀

We will use this lemma in the form given in the next corollary.

▶ **Corollary 7.** *For $c,k \geq 0$ such that $k \in O(c)$, the (k,c)-*Congestion Routing *problem can be solved on any acyclic n-vertex digraph G in time $n^{O(c)}$.*

The next lemma provides the main reduction argument for proving Theorem 3.

▶ **Lemma 8.** *Let G be an acyclic directed graph and let $d \geq 1$ and $k > 3d$. Let $I := \{(s_1,t_1),\dots,(s_k,t_k)\} \subseteq V(G) \times V(G)$ be a set of source/terminal pairs. There exists a $(k-d)$-routing of I if, and only if, for every pair $(s,t) \in I$ there is a path in G from s to t and there is a subset $I' \subsetneq I$ of order $|I'| = k-1$ such that there is a $(k-d-1)$-routing of I'.*

Proof. The if direction is easy to see. Let $\mathcal{S}' := \{P_1,\dots,P_{k-1}\}$ be a $(k-d-1)$-routing of a set $I' \subseteq I$ of order $k-1$. Let s,t be such that $I = I' \cup \{(s,t)\}$. By assumption there is a simple path P from s to t in G. Then $\mathcal{S} := \mathcal{S}' \cup \{P\}$ is a $(k-d)$-routing of I.

For the reverse direction let $I := \{(s_1,t_1),\dots,(s_k,t_k)\}$ and let $\hat{\mathcal{S}} := \{\hat{P}_1,\dots,\hat{P}_k\}$ be a $(k-d)$-routing of I such that $\hat{P}_i$ links s_i to t_i, for all $1 \leq i \leq k$. We define a multi digraph G' on the same vertex set $V(G)$ as G as follows. For every pair $u,v \in V(G')$ such that $e = (u,v) \in E(G)$ and every $1 \leq i \leq k$, if e occurs on the path $\hat{P}_i \in \mathcal{S}$, then we add a new edge $e^i = (u,v)$ to G'. Hence, if any edge $e \in E(G)$ is used by ℓ different paths in $\hat{\mathcal{S}}$, then G' contains ℓ parallel edges between the endpoints of e. In the rest of the proof we will work on the multi digraph G'. We can now take a set $\mathcal{S} := \{P_1,\dots,P_k\}$ of pairwise edge disjoint paths, where P_i is the path from s_i to t_i induced by the edge set $\{e^i : e \in E(\hat{P}_i)\}$. That is, by using the parallel edges, we can turn the routing $\hat{\mathcal{S}}$ into a $(k-d)$-routing $\mathcal{S}$ of I where the paths are mutually edge disjoint.

In the remainder of the proof we will construct a subset $I' \subsetneq I$ of order $k-1$ and a $(k-d-1)$-routing of I' in G' which is pairwise edge disjoint. This naturally induces a $(k-d-1)$-routing of I' in G. Note that in G', if $\mathcal{L}$ is any set of pairwise edge disjoint paths, then the congestion of any vertex with respect to $\mathcal{L}$ is at most the congestion of the vertex with respect to $\mathcal{S}$ (and thus $\hat{\mathcal{S}}$) in G' (and G, respectively). Indeed, every edge in $\mathcal{L}$ has a corresponding path in $\mathcal{S}$, so no vertex can be contained in more paths from $\mathcal{L}$ than in $\mathcal{S}$.

Let $\sqsubseteq$ be a topological ordering of G' and let $A := \{a_1,\dots,a_\ell\}$ be the set of vertices of congestion $k-d$ with respect to $\mathcal{S}$ such that $a_i \sqsubseteq a_j$ whenever $i < j$. As $k > 3d$, for all $1 \leq i < \ell$ there is a path in G from a_i to a_{i+1}.

For $1 \leq i \leq k$, an *atomic subpath* of P_i (with respect to $\mathcal{S}$) is a subpath of P_i that starts and ends in a vertex of $A \cup \{s_i,t_i\}$ and is internally vertex disjoint from A. Hence, every path $P_i \in \mathcal{S}$ consists of the concatenation $P_1^1 \cdot \dots \cdot P_i^{\ell_i}$ of its atomic subpaths where we identify the last vertex of P_i^j with the first vertex of P_i^{j+1} for all $1 \leq j < \ell_i$. Note that any two atomic subpaths of paths P_i, P_j in $\mathcal{S}$ are pairwise edge disjoint.

Let $I' \subset I$ be a subset of order $k-1$. A routing $\mathcal{S}' := \{P'_1, \dots, P'_{k-1}\}$ of I' is *conservative with respect to* $\mathcal{S}$ if it consists of pairwise edge disjoint paths and every path in $\mathcal{S}'$ consists of a concatenation of atomic subpaths of paths in $\mathcal{S}$. In the sequel, whenever we speak of a conservative I'-routing we implicitly mean that it is conservative with respect to $\mathcal{S}$.

If $\mathcal{S}'$ is a conservative I'-routing with respect to $\mathcal{S}$, then it consists of pairwise edge disjoint paths and hence for every $v \in V(G)$ the congestion of v with respect to $\mathcal{S}'$ is at most the congestion of v with respect to $\mathcal{S}$.

Let $1 \leq i_1 < i_2 \leq \ell$ and let $1 \leq j \leq k$. Let $\mathcal{S}'$ be a conservative I'-routing. An (i_1, i_2)-*jump of colour* j is a subpath P' of P_j from a_{i_1} to a_{i_2} such that for all i with $i_1 < i < i_2$ the vertex a_i is not on P_j. Note that any jump is an atomic subpath. We call the jump P' *free with respect to* $\mathcal{S}'$ if P' is not used by any path in $\mathcal{S}'$.

We are now ready to complete the proof of the lemma. Note first that, as $k > 3d$, for any three vertices $b_1, b_2, b_3 \in A$ there is a path $P \in \mathcal{S}$ that contains b_1, b_2, b_3. Hence, we can choose an $h \in \{1, \dots, k\}$ such that $a_1, a_\ell \in V(P_h)$ and there is a vertex a_r with $1 < r < \ell$ such that $a_r \in V(P_h)$. Let $I' := I \setminus \{(s_h, t_h)\}$. If $A \subseteq V(P_h)$, then $\mathcal{S} \setminus \{P_h\}$ is a $(k-d-1)$-routing of I' and we are done. Otherwise, for every vertex $a_r \in A$ which has congestion $k-d$ with respect to $\mathcal{S} \setminus \{P_h\}$ there are i, j with $i < r < j$ and an (i,j)-jump P of colour h. This follows as $a_1, a_\ell \in V(P_h)$. Note also that a_1 and a_ℓ have congestion $k-d-1$ in $\mathcal{S} \setminus \{P_h\}$. Note that this jump P is free with respect to $\mathcal{S} \setminus \{P_h\}$.

Thus, it is easily seen that $\mathcal{S} \setminus \{P_h\}$ satisfies the following two properties:

1. For every vertex a_r of congestion $k-d$ with respect to $\mathcal{S} \setminus \{P_h\}$ there are indices $i < r < j$ such that there is a free (i,j)-jump P with respect to $\mathcal{S} \setminus \{P_h\}$.
2. For any three vertices b_1, b_2, b_3 of congestion $k-d$ with respect to $\mathcal{S} \setminus \{P_h\}$ there is a path $Q \in \mathcal{S} \setminus \{P_h\}$ with $\{b_1, b_2, b_3\} \subseteq V(Q)$.

Now let $\mathcal{S}'$ be a routing of I' which satisfies Condition 1 and 2 (with respect to $\mathcal{S}'$ instead of $\mathcal{S} \setminus \{P_h\}$) and, subject to this, the number of vertices of congestion $k-d$ with respect to $\mathcal{S}'$ is minimal.

We claim that $\mathcal{S}'$ is a $(k-d-1)$-routing of I'. Let $\mathcal{S}' := \{Q_1, \dots, Q_{k-1}\}$. Towards a contradiction, suppose there is a vertex a_r of congestion $k-d$ with respect to $\mathcal{S}'$. As $\mathcal{S}'$ is conservative, we have $a_r \in A$. Hence, by assumption, there are $i < r < j$ and a free (i,j)-jump P with respect to $\mathcal{S}'$.

Let Q_h be a path in $\mathcal{S}'$ that contains a_i, a_r and a_j, which exists by Condition 2. Let $Q_h := Q_h^1 \cup Q_h^2 \cup Q_h^3$ where

- Q_h^1 is the initial subpath of Q_h from its first vertex to a_i,
- Q_h^2 is the subpath starting at a_i and ending in a_j and
- Q_h^3 is the subpath starting in a_j and ending at the end of Q_h.

We define $Q'_h := Q_h^1 \cup P \cup Q_h^3$, i.e. Q'_h is the path obtained from Q_h by replacing the inner subpath Q_h^2 by the (i,j)-jump P. Let $\mathcal{L} := (\mathcal{S}' \setminus \{Q_h\}) \cup \{Q'_h\}$. Then $\mathcal{L}$ is a routing of I'. It is also conservative as we have only rerouted a single path along a free jump.

We need to show that for all b_1, b_2, b_3 of congestion $k-d$ with respect to $\mathcal{L}$ there is a path $Q \in \mathcal{L}$ containing b_1, b_2, b_3. By assumption, such a path Q' exists in $\mathcal{S}'$. If $Q' \neq Q_h$, then we are done. So suppose $Q_h = Q'$. But then this implies that $b_s \notin \{a_{i+1}, \dots, a_{j-1}\}$ for all $1 \leq s \leq 3$ as otherwise the congestion of b_s would have dropped to $k-d-1$ in $\mathcal{L}$. But then $b_1, b_2, b_3 \in V(Q'_h)$.

It remains to show that for every vertex a_s of congestion $k-d$ with respect to $\mathcal{L}$ there is a free (i,j)-jump for some $i < s < j$. As before, by assumption, there are $s_1 < s < s_2$ and a free (s_1, s_2)-jump with respect to $\mathcal{S}'$. If this jump is not P, then it still exists with respect to

$\mathcal{L}$ and we are done. So suppose this jump is P, which implies that $i < s < j$. Furthermore, $a_s \notin Q_h$ as otherwise the congestion of a_s in $\mathcal{L}$ would be $k - d - 1$. But then, there must be indices i_1, i_2 with $i \leq i_1 < s < i_2 \leq j$ such that $a_{i_1}, a_{i_2} \in V(Q_h)$ and $a_{s'} \notin V(Q_h)$ for all $i_1 < s' < i_2$. Hence, the atomic subpath Q'' of Q_h from a_{i_1} to a_{i_2} is an (i_1, i_2)-jump as required. As $Q'' \subseteq Q_h^2$, this jump is now free.

Finally, the vertex a_r now has congestion $k-d-1$ with respect to $\mathcal{L}$ as a_r is not contained in Q'_h. Hence, $\mathcal{L}$ has fewer vertices of congestion $k - d$ than $\mathcal{S}'$, contradicting the choice of $\mathcal{S}'$. Thus, $\mathcal{S}'$ must have been a $(k - d - 1)$-routing of I' as required. This completes the proof of the lemma. ◀

By repeatedly applying Lemma 8 we obtain the following corollary, which essentially implies Theorem 3.

▶ **Corollary 9.** *Let G be an acyclic digraph, $d \geq 0$, $k \geq 3d$ and let $I := \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be a set of pairs of vertices such that for all $1 \leq i \leq k$ there is a path in G linking s_i to t_i. Then G contains a $(k - d)$-routing of I if, and only if, there is a subset $I' \subseteq I$ with $|I'| = 3d$ such that G contains a $2d$-routing of I'.*

We are now ready to prove Theorem 3.

Proof of Theorem 3. Let G, k, d and $I := \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be given. Let $n := |G|$. If for some $1 \leq i \leq k$ there is no path in G from s_i to t_i, then the answer is no and we are done. If $k \leq 3d$, then we can apply Corollary 7 to compute the answer in time $n^{O(d)}$ as required.

Otherwise, by Corollary 9, there is a $(k - d)$-routing for I in G if, and only if, there is a subset $I' \subsetneq I$ of order $3d$ such that I' has a $2d$-routing. There are $\binom{k}{3d} \leq k^{3d} \leq n^{3d}$ subsets I' of order $3d$. By Corollary 7, we can decide for any such I' of order $3d$ in time $n^{O(d)}$ whether a $2d$-routing of I' exists. Hence, by iterating through all possible subsets I', we can decide in time $n^{O(d)}$ whether there is a $(k - d)$-routing of I in G. ◀

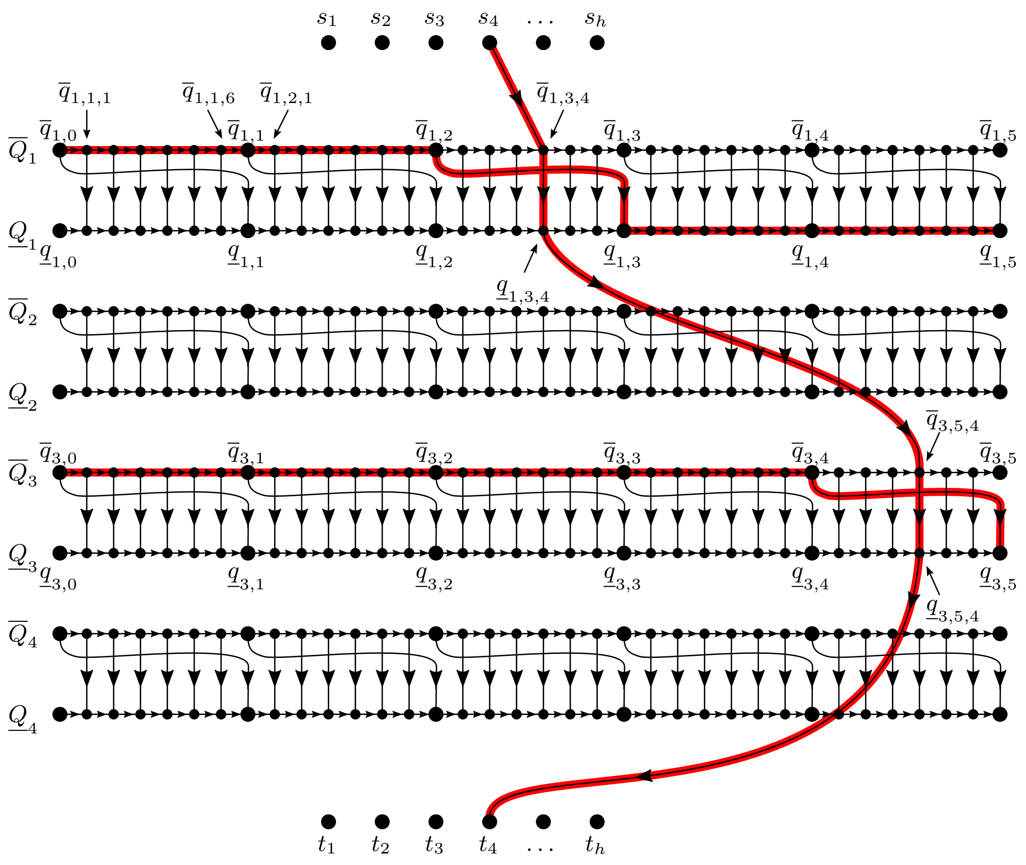
4 Lower Bounds

In this section, we prove Theorem 2 by a reduction from Partitioned Subgraph Isomorphism. The input of the Partitioned Subgraph Isomorphism problem consists of a graph H with vertex set $\{u_1, \ldots, u_k\}$ and a graph G whose vertex set is partitioned into k classes $V_1, \ldots, V_k$. The task is to find a mapping $\mu : V(H) \to V(G)$ such that $\mu(u_i) \in V_i$ for every $1 \leq i \leq k$ and μ is a subgraph embedding, that is, if u_i and u_j are adjacent in H, then $\mu(u_i)$ and $\mu(u_j)$ are adjacent in G.

▶ **Theorem 10** ([20]). *Assuming ETH,* Partitioned Subgraph Isomorphism *cannot be solved in time $f(k)n^{o(k/\log k)}$ (where $k = |V(H)|$) for any computable function f, even when H is assumed to be 3-regular and bipartite.*

To prove Theorem 2, we need a reduction from Partitioned Subgraph Isomorphism (for 3-regular bipartite graphs) to (k, c)-Congestion Routing, where the number k of demands is linear in the number of vertices of H.

Proof (of Theorem 2). We prove the theorem by a reduction from Partitioned Subgraph Isomorphism. Let H and G be two graphs, let $V(H) = \{u_1, \ldots, u_k\}$, and let $(V_1, \ldots, V_k)$ be a partition of $V(G)$. By copying vertices if necessary, we may assume that every V_i has the same size n; let us denote by $\{v_{i,1}, \ldots, v_{i,n}\}$ the vertices in V_i. By Theorem 10, we may assume that H is 3-regular and bipartite. This means that H has exactly $h = 3k/2$ edges



s_1 s_2 s_3 s_4 … s_h
$\overline{q}_{1,1,1}$
$\overline{q}_{1,1,6}$
$\overline{q}_{1,2,1}$
$\overline{q}_{1,3,4}$
$\overline{q}_{1,0}$ $\overline{q}_{1,1}$ $\overline{q}_{1,2}$ $\overline{q}_{1,3}$ $\overline{q}_{1,4}$ $\overline{q}_{1,5}$
$\overline{Q}_1$
$\underline{Q}_1$
$\underline{q}_{1,0}$ $\underline{q}_{1,1}$ $\underline{q}_{1,2}$ $\underline{q}_{1,3}$ $\underline{q}_{1,4}$ $\underline{q}_{1,5}$
$\underline{q}_{1,3,4}$
$\overline{Q}_2$
$\underline{Q}_2$
$\overline{q}_{3,5,4}$
$\overline{q}_{3,0}$ $\overline{q}_{3,1}$ $\overline{q}_{3,2}$ $\overline{q}_{3,3}$ $\overline{q}_{3,4}$ $\overline{q}_{3,5}$
$\overline{Q}_3$
$\underline{Q}_3$
$\underline{q}_{3,0}$ $\underline{q}_{3,1}$ $\underline{q}_{3,2}$ $\underline{q}_{3,3}$ $\underline{q}_{3,4}$ $\underline{q}_{3,5}$
$\underline{q}_{3,5,4}$
$\overline{Q}_4$
$\underline{Q}_4$
t_1 t_2 t_3 t_4 … t_h

Figure 1 Part of the directed graph D constructed in the proof of Theorem 2 with $k = 4$, $h = 6$, and $n = 5$. For clarity, we consider only one edge e_4 of H, which connects u_1 and u_3, and assume that the only edge between V_1 and V_3 is between $v_{1,3}$ and $v_{3,5}$. The highlighted red paths show the paths P_1^v, P_3^v, and P_4^e of the solution.

and both partite classes contain $k/2$ vertices. Without loss of generality, we can assume that $U_1 = \{u_1, \ldots, u_{k/2}\}$ and $U_2 = \{u_{k/2+1}, \ldots, u_k\}$ are the two partite classes. Let us fix an arbitrary ordering e_1, …, e_h of the edges of H.

Construction. We construct an instance of (k, c)-Congestion Routing in the following way. We construct a directed graph D that contains, for every $1 \leq i \leq k$, two directed paths $\overline{Q}_i$ and $\underline{Q}_i$ (see Figure 1). Path $\overline{Q}_i$ has $n(h + 1) + 1$ vertices: it contains the vertices $\overline{q}_{i,0}$, …, $\overline{q}_{i,n}$ in this order and additionally, for every $1 \leq j \leq n$, the vertices $\overline{q}_{i,j,1}$, …, $\overline{q}_{i,j,h}$ are inserted between $\overline{q}_{i,j-1}$ and $\overline{q}_{i,j}$. The path $\underline{Q}_i$ is defined the same way, with vertices $\underline{q}$ instead of $\overline{q}$. For every $1 \leq \ell \leq h$, we introduce two vertices s_ℓ and t_ℓ. Then we complete the construction of the graph D by introducing further edges as follows.

- For every $1 \leq i \leq k$ and $1 \leq j \leq n$, we introduce the edge $(\overline{q}_{i,j-1}, \underline{q}_{i,j})$ (the curved bypass edges in Figure 1).
- For every $1 \leq i \leq k$, $1 \leq j \leq n$, and $1 \leq s \leq h$, we introduce the edge $(\overline{q}_{i,j,s}, \underline{q}_{i,j,s})$ (the vertical edges in Figure 1).
- For every $1 \leq \ell \leq h$, we do the following. Suppose that edge e_ℓ of H connects u_{i_a} and u_{i_b} for some $1 \leq i_a \leq k/2$ and $k/2 + 1 \leq i_b \leq k$. Then for every pair of vertices $v_{i_a,j_a} \in V_{i_a}$ and $v_{i_b,j_b} \in V_{i_b}$ that are adjacent in G, we introduce the following three edges into D:

$(s_\ell, \overline{q}_{i_a,j_a,\ell})$, $(\underline{q}_{i_a,j_a,\ell}, \overline{q}_{i_b,j_b,\ell})$, and $(\underline{q}_{i_b,j_b,\ell}, t_\ell)$.

To complete the construction of the (k,c)-Congestion Routing instance, we define the following set of $k + 2k(c-1) + h$ demands:

- For every $1 \leq i \leq k$, we introduce the demand $(\overline{q}_{i,0}, \underline{q}_{i,n})$ (vertex demands).
- For every $1 \leq i \leq k$, we introduce $c-1$ copies of the demand $(\overline{q}_{i,0}, \overline{q}_{i,n})$ (blocking demands).
- For every $1 \leq i \leq k$, we introduce $c-1$ copies of the demand $(\underline{q}_{i,0}, \underline{q}_{i,n})$ (blocking demands).
- For every $1 \leq \ell \leq h$, we introduce the demand (s_ℓ, t_ℓ) (edge demands).

Note that, for every fixed $c \geq 1$, the number of demands is $O(k)$. In the rest of the proof, we show that a routing with congestion c exists if and only if the Partitioned Subgraph Isomorphism instance has a solution. Then the W[1]-hardness and lower bound stated in Theorem 10 implies the same hardness results for the routing problem.

Subgraph embedding $\Rightarrow$ routing. Suppose first that vertices $v_{1,z_1} \in V_1$, …, $v_{k,z_k} \in V_k$ form a solution to the Partitioned Subgraph Isomorphism instance. We construct a routing that contains the following paths, satisfying the demands defined above:

- For every $1 \leq i \leq k$, the vertex demand $(\overline{q}_{i,0}, \underline{q}_{i,n})$ is satisfied by a path P_i^v that goes from $\overline{q}_{i,0}$ to $\overline{q}_{i,z_i-1}$ on $\overline{Q}_i$, uses the edge $(\overline{q}_{i,z_i-1}, \underline{q}_{i,z_i})$, and then goes from $\underline{q}_{i,z_i}$ to $\underline{q}_{i,n}$ on $\underline{Q}_i$.
- For every $1 \leq i \leq k$, each of the $c-1$ copies of the blocking demand $(\overline{q}_{i,0}, \overline{q}_{i,n})$ is satisfied by a path going on $\overline{Q}_i$.
- For every $1 \leq i \leq k$, each of the $c-1$ copies of the blocking demand $(\underline{q}_{i,0}, \underline{q}_{i,n})$ is satisfied by a path going on $\underline{Q}_i$.
- For every $1 \leq \ell \leq h$, the edge demand (s_ℓ, t_ℓ) is satisfied by a 5-edge path $P_\ell^e = (s_\ell, \overline{q}_{i_a,z_{i_a},\ell}, \underline{q}_{i_a,z_{i_b},\ell}, \overline{q}_{i_b,z_{i_b},\ell}, \underline{q}_{i_b,z_{i_b},\ell}, t_\ell)$.

It is easy to verify that these are indeed paths: all the required edges exist. We claim that each vertex of D is used by at most c of these paths. It is easy to see that two paths $P_{i'}^v$ and $P_{i''}^v$ with $i \neq i''$ satisfying vertex demands do not intersect, and this is also true for any two paths $P_{\ell'}^e$ and $P_{\ell''}^e$ with $\ell' \neq \ell''$ satisfying edge demands (note that each vertex of the path P_ℓ^e has ℓ in its index). The crucial observation is that the path P_i^v does not intersect the path P_ℓ^e for any ℓ. The only way this could possibly happen is if edge e_ℓ of H connects u_{i_a} with u_{i_b}, and i is equal to i_a or i_b. But the path P_ℓ^e uses only vertex $\overline{q}_{i_a,z_{i_a},\ell}$ from $\overline{Q}_{i_a}$ and vertex $\underline{q}_{i_a,z_{i_b},\ell}$ from $\underline{Q}_{i_b}$, while the path P_i^v does not use these vertices, as it jumps from $\overline{q}_{i,z_i-1}$ to $\underline{q}_{i,z_i}$. Thus each vertex is used by at most $c-1$ paths satisfying a blocking demand and at most one additional path satisfying a vertex or edge demand. We can conclude that each vertex is used by at most c of the paths, what we had to show.

Routing $\Rightarrow$ subgraph embedding. Next we show that given a routing with congestion c, it is possible to construct the required subgraph embedding from H to G. It is clear that the path satisfying the blocking demand $(\overline{q}_{i,0}, \overline{q}_{i,n})$ is exactly $\overline{Q}_i$: after leaving $\overline{Q}_i$, there is no way to return back to it. Similarly, the solution must use path $\underline{Q}_i$ to satisfying the blocking demand $(\underline{q}_{i,0}, \underline{q}_{i,n})$. It is also clear that the path P_i^v satisfying the vertex demand $(\overline{q}_{i,0}, \underline{q}_{i,n})$ has to be contained in the union of $\overline{Q}_i$ and $\underline{Q}_i$. Let $1 \leq z_i \leq n$ be the smallest value such that $\underline{q}_{i,z_i}$ is on path P_i^v (note that this value is positive, as vertex $\underline{q}_{i,0}$ cannot be reached from $\overline{q}_{i,0}$). Observe that path P_i^v uses every vertex of $\underline{Q}_i$ from $\underline{q}_{i,z_i}$ to $\underline{q}_{i,n}$ (as it cannot leave $\underline{Q}_i$). Moreover, since P_i^v does not use the part of $\underline{Q}_i$ from $\underline{q}_{i,0}$ to $\underline{q}_{i,z_i-1}$ by definition, it has to use the part of $\overline{Q}_i$ from $\overline{q}_{i,0}$ to $\overline{q}_{i,z_i-1}$.

We claim that mapping vertex u_i of H to vertex v_{i,z_i} of G is a correct subgraph embedding of H into G. To show this, suppose that edge e_i of H connects u_{i_a} and u_{i_b} with $1 \leq i_a \leq k/2$ and $k/2+1 \leq i_b \leq k$; we need to show that $v_{i_a,z_{i_a}} \in V_{i_a}$ and $v_{i_b,z_{i_b}} \in V_{i_b}$ are adjacent. Consider the path P^e_ℓ satisfying edge demand (s_ℓ, t_ℓ). By construction, the vertex of P^e_ℓ after s_ℓ has to be on the path $\overline{Q}_{i_a}$ and the vertex of P^e_ℓ before t_ℓ has to be on $\underline{Q}_{i_b}$. The only way to go from $\overline{Q}_{i_a}$ to $\underline{Q}_{i_b}$ is to use an edge of the form $(\underline{q}_{i_a,j_a,\ell}, \overline{q}_{i_b,j_b,\ell})$: the only way we can leave the union of $\overline{Q}_{i_a}$ and $\underline{Q}_{i_a}$ is to enter some $\overline{Q}_i$ with $k/2+1 \leq i \leq k$, and there is no edge connecting $\overline{Q}_{i_b}$ or $\underline{Q}_{i_b}$ with any $\overline{Q}_i$ with $k/2+1 \leq i \leq k$ and $i \neq i_b$ (this is the part of the proof where we use that H is bipartite). We claim that $j_a = z_{i_a}$. If $j > z_{i_a}$, then $\underline{q}_{i_a,j_a,\ell}$ is also used by the $c-1$ paths satisfying the blocking demand $(\underline{q}_{i_a,0}, \underline{q}_{i_a,n})$ and (as we have seen) the path $P^v_{i_a}$, contradicting the assumption that the routing has congestion c. If $j < z_{i_a}$, then there is no way for the path P^e_ℓ to reach $\underline{q}_{i_a,j_a,\ell}$ from s_ℓ: each vertex of the path $\overline{Q}_{i_a}$ from $\overline{q}_{i_a,0}$ to $\overline{q}_{i_a,j_a}$ is used by $c-1$ paths satisfying the blocking demand $(\underline{q}_{i_a,0}, \underline{q}_{i_a,n})$ and (as shown above) by the path $P^v_{i_a}$. This shows $j_z = z_{i_a}$ and a similar argument shows $j_b = z_{i_b}$. Now the existence of the edge $(\underline{q}_{i_a,z_a,\ell}, \overline{q}_{i_b,z_b,\ell})$ means, by construction, that G contains an edge between $v_{i_a,z_a} \in V_{i_a}$ and $v_{i_b,z_b} \in V_{i_b}$, what we had to show. ◀

5 Conclusion

In this paper we have studied the (k,c)-Congestion Routing problem on acyclic digraphs. It is easy to see that the $n^{O(k)}$ algorithm in [15] for solving the disjoint paths problem on acyclic digraphs can be extended to an $n^{O(k)}$ algorithm for (k,c)-Congestion Routing. As we proved in Theorem 2, the $n^{O(k)}$ time algorithm is essentially best possible with respect to the exponent of n, under the Exponential-Time Hypothesis (ETH). We therefore studied the extreme cases of relatively high congestion $k-d$ for some fixed value of d. In Theorem 3 we showed that in this case we can obtain an $n^{O(d)}$ algorithm on acyclic digraphs, i.e. the algorithm only depends on the offset d in $(k,k-d)$-Congestion Routing but not on the number k of demand pairs. The proof relied on a reduction argument that shows that as long as k is big enough compared to d, then a demand pair can be eliminated without changing the answer.

It will be interesting to see whether our result can be extended to larger classes of digraphs. In particular classes of digraphs of bounded directed tree width would be a natural target. On such classes, the k-disjoint paths problem can be solved in time $n^{O(k+w)}$, where w is the directed tree width of the input digraph (see [17]). It is conceivable that our results extend to bounded directed tree width classes and we leave this for future research.

References

1 Saeed Akhoondian Amiri, Ali Golshani, Stephan Kreutzer, and Sebastian Siebertz. Vertex disjoint paths in upward planar graphs. In Edward A. Hirsch, Sergei O. Kuznetsov, Jean-Éric Pin, and Nikolay K. Vereshchagin, editors, *Computer Science - Theory and Applications: 9th International Computer Science Symposium in Russia, CSR 2014, Moscow, Russia, June 7-11, 2014. Proceedings*, pages 52–64. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-06686-8_5.

2 Matthew Andrews, Julia Chuzhoy, Venkatesan Guruswami, Sanjeev Khanna, Kunal Talwar, and Lisa Zhang. Inapproximability of edge-disjoint paths and low congestion routing on undirected graphs. *Combinatorica*, 30(5):485–520, 2011. doi:10.1007/s00493-010-2455-9.

3 Jorgen Bang-Jensen and Gregory Z. Gutin. *Digraphs - Theory, Algorithms and Applications.* Springer, 2nd edition, 2010.

4 Parinya Chalermsook, Julia Chuzhoy, Alina Ene, and Shi Li. Approximation algorithms and hardness of integral concurrent flow. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 689–708, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214040.

5 C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-disjoint paths in planar graphs. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 71–80, Oct 2004. doi:10.1109/FOCS.2004.27.

6 Chandra Chekuri and Alina Ene. Poly-logarithmic approximation for maximum node disjoint paths with constant congestion. 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 326–341. SIAM, 2013. doi:10.1137/1.9781611973105.24.

7 Chandra Chekuri and Alina Ene. The all-or-nothing flow problem in directed graphs with symmetric demand pairs. In Jon Lee and Jens Vygen, editors, *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings*, volume 8494 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2014. doi:10.1007/978-3-319-07557-0_19.

8 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. Multicommodity flow, well-linked terminals, and routing problems. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 183–192, New York, NY, USA, 2005. ACM. doi:10.1145/1060590.1060618.

9 Chandra Chekuri, Sanjeev Khanna, and F. Bruce Shepherd. An $o(\sqrt{n})$ approximation and integrality gap for disjoint paths and unsplittable flow. *Theory of Computing*, 2(7):137–146, 2006. doi:10.4086/toc.2006.v002a007.

10 J. Chuzhoy and S. Li. A polylogarithmic approximation algorithm for edge-disjoint paths with congestion 2. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 233–242, Oct 2012. doi:10.1109/FOCS.2012.54.

11 M. Cygan, D. Marx, M. Pilipczuk, and M. Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 197–206, Oct 2013. doi:10.1109/FOCS.2013.29.

12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015. doi:10.1007/978-3-319-21275-3.

13 Reinhard Diestel. *Graph Theory.* Springer-Verlag, 4th edition, 2010.

14 S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

15 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111 – 121, 1980. URL: http://www.sciencedirect.com/science/article/pii/0304397580900092, doi:http://dx.doi.org/10.1016/0304-3975(80)90009-2.

16 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

17 Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *J. Comb. Theory, Ser. B*, 82(1):138–154, 2001.

18 G. Stavros Kolliopoulos and Clifford Stein. Approximating disjoint-path problems using packing integer programs. *Mathematical Programming*, 99(1):63–87, 2003. doi:10.1007/s10107-002-0370-6.

19 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: `http://albcom.lsi.upc.edu/ojs/index.php/beatcs/article/view/96`.

20 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. `arXiv:toc:v006/a005`, `doi:10.4086/toc.2010.v006a005`.

21 N. Robertson and P. D. Seymour. Graph minors I – XXIII, 1982 – 2010. *Appearing in Journal of Combinatorial Theory, Series B.*, from 1982-2010.

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

Stochastic Timed Games Revisited

S. Akshay*[1], Patricia Bouyer†[2], Shankara Narayanan Krishna ‡[3], Lakshmi Manasa[4], and Ashutosh Trivedi[5]

1 **Department of Computer Science & Engineering, IIT Bombay, Bombay, India**
`akshayss@cse.iitb.ac.in`
2 **LSV, CNRS & ENS Cachan, Université Paris-Saclay, Paris, France**
`bouyer@lsv.fr`
3 **Department of Computer Science & Engineering, IIT Bombay, Bombay, India**
`krishnas@cse.iitb.ac.in`
4 **Department of Computer Science & Engineering, IIT Bombay, Bombay, India**
`manasa@cse.iitb.ac.in`
5 **University of Colorado, Boulder, USA**
`ashutosh.trivedi@colorado.edu`

Abstract

Stochastic timed games (STGs), introduced by Bouyer and Forejt, naturally generalize both continuous-time Markov chains and timed automata by providing a partition of the locations between those controlled by two players (Player Box and Player Diamond) with competing objectives and those governed by stochastic laws. Depending on the number of players – 2, 1, or 0 – subclasses of stochastic timed games are often classified as $2\frac{1}{2}$-player, $1\frac{1}{2}$-player, and $\frac{1}{2}$-player games where the $\frac{1}{2}$ symbolizes the presence of the stochastic "nature" player. For STGs with reachability objectives it is known that $1\frac{1}{2}$-player one-clock STGs are decidable for qualitative objectives, and that $2\frac{1}{2}$-player three-clock STGs are undecidable for quantitative reachability objectives. This paper further refines the gap in this decidability spectrum. We show that quantitative reachability objectives are already undecidable for $1\frac{1}{2}$ player four-clock STGs, and even under the time-bounded restriction for $2\frac{1}{2}$-player five-clock STGs. We also obtain a class of $1\frac{1}{2}$, $2\frac{1}{2}$ player STGs for which the quantitative reachability problem is decidable.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Timed automata, stochastic games, two-counter machines

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.8

1 Introduction

Two-player zero-sum games over finite state-transition graphs are a natural framework for controller synthesis for discrete event systems. In this setting two players – say Player Box and Player Diamond (after necessity and possibility operators) – represent the controller and the environment, and control-program synthesis corresponds to finding a winning (or optimal) strategy of the controller for some given performance objective. Finite graphs, however, often do not satisfactorily model real-time safety-critical systems as they disregard not only the continuous dynamics of the physical environment but also the presence of stochastic behavior. Stochastic behavior in such systems stems from many different sources, e.g., faulty

* Partly supported by DST-INSPIRE Faculty Grant [IFA12-MA-17].
† Partly supported by ERC project EQualIS (308087).
‡ Partly supported by CEFIPRA project AVeRTS.

or unreliable sensors or actuators, uncertainty in timing delays, the random coin flips of distributed communication and security protocols.

Timed automata [2] were introduced as a formalism to model asynchronous real-time systems interacting with a continuous physical environment. Timed automata and their two-player counterparts [3] provide an intuitive and semantically unambiguous way to model non-stochastic real-time systems, and a number of case-studies [23] demonstrate their application in the design and analysis of real-time systems. On the other hand, classical formalisms (discrete-time and continuous-time) Markov decision processes (MDPs) and stochastic games [22, 15] naturally model analysis and synthesis problems for stochastic systems, and have been applied in control theory, operations research, and economics.

For the formal analysis of stochastic real-time systems, a number of recent works considered a combination of stochastic features with timed automata, e.g. probabilistic timed automata [18], continuous probabilistic timed automata [17] and stochastic timed automata [9]. Probabilistic timed automata, respectively continuous probabilistic and stochastic timed automata can be considered as generalizations of timed automata with the features of discrete-time Markov decision processes, respectively continuous-time Markov chains [5] (or even generalized semi-Markov processes [13]). Stochastic timed games [12] form the most general formalism for studying controller-synthesis for stochastic real-time systems. These games can be considered as interactions between three players – Player Box, Player Diamond and the stochastic player (Nature) – such that Player Box and Player Diamond are adversarial and choose their delay and action so as to maximize and minimize probability to reach a given set of target states, while the stochastic player plays according to a given probability distribution. A key verification problem in this setting is that of games with reachability objectives, where the goal of Player Diamond is to reach a set of target states, while the goal of the Player Box is to avoid it.

Related Work. Probabilistic timed automata [18] and games [16] can be considered as subclasses of stochastic timed games where all of the locations controlled by stochastic players are *urgent* (no time delay allowed), while the decision-stochastic timed automata of [10] can be seen as a subclass of $1\frac{1}{2}$-player STGs where the locations of the rational players are urgent. The quantitative reachability problem for probabilistic timed automata is known to be decidable [18] with any number of clocks, while the best known decidability result for the quantitative reachability problem for $1\frac{1}{2}$-player STGs is using a single clock. $\frac{1}{2}$-player STGs, also called stochastic timed automata (STA) [9], have also received considerable attention: an abstraction based on the region abstraction has been proposed, which allows to solve the qualitative reachability problem under a *fairness* assumption on the STA (several subclasses of STAs have been proven to be fair). For quantitative reachability, the only decidability result is for a subclass of single-clock STA [8], but a recent approximability result has been shown in [7] for the class of *fair STA*.

Other variants of stochastic timed automata have been studied in the past. The model in [17] uses "countdown clocks" (which decrease from a set value) unlike the more timed-automata style of clock variables used in our model. The model in [11] (which is also called stochastic timed automata; we shall refer to them here as Modest-STA) is very general and encompasses most models with time and probabilities (and in particular the STA of [9]). However, Modest-STA is more aimed at capturing general languages (and providing a tool-set to simulate their runs) and less with decidability issues, and hence is orthogonal to our approach.

Table 1 Results in bold are contributions from this paper. "Conj" are conjectures.

Model		Qualitative Results	Quantitative Results
$\frac{1}{2}$ player	1 clock	Dec. [4]	Dec. (some restrictions) [8]
	n clocks	Open in general Dec. (fair) [9]	Open in general Approx. (fair) [7]
$1\frac{1}{2}$ player	1 clock	Dec. [12]	**Dec. (Initialized, Theorem 8)**
	n clocks	Open	**Undec. (Theorem 3)** Conj: **Undec.** (Time bounded)
$2\frac{1}{2}$ player	1 clock	Conj: **Dec.**	**Dec. (Initialized, Corollary 9)**
	n clocks	Open	Undec [12] **Undec. (Time bounded, Theorem 6)**

Contributions. The scope of this paper is to investigate decidability of the reachability problem in STGs as defined in [12], for which the decidability picture is far from complete. In [12], the authors showed the decidability of qualitative reachability problem on 1-clock $1\frac{1}{2}$-player STGs, and the undecidability of quantitative reachability problem on STGs (with $2\frac{1}{2}$-players). This leaves a wide gap in the decidability horizon of STGs. In this paper, we study $1\frac{1}{2}$, $2\frac{1}{2}$-player games and contribute to a better understanding of the decidability status of STGs with quantitative reachability objectives.

Table 1 summarizes the results presented in this paper. We show that the quantitative reachability problem is already undecidable for $1\frac{1}{2}$-player games for systems with 4 or more clocks and for $2\frac{1}{2}$-player games the quantitative reachability problem remains undecidable even under the time-bounded restriction with 5 or more clocks. Another key contribution of this paper is the characterization of a previously unexplored subclass of stochastic timed games for which we recover decidability of quantitative reachability game for $1\frac{1}{2}$ (and even $2\frac{1}{2}$)-player stochastic timed games. We call a 1-clock stochastic timed game *initialized* if (i) all the transitions from non-stochastic states to stochastic states reset the clock, and (ii) in every bounded cycle, the clock is reset. The definition can be generalized to multiple clocks using the notion of strong reset where one resets all the clocks together. For some of the gaps in this spectrum, we provide our best conjectures as justified in the Discussion section:–the undecidability of time-bounded quantitative reachability for $1\frac{1}{2}$-player STG, and the decidability of qualitative reachability of 1-clock $2\frac{1}{2}$-player STG. Due to lack of space, details of some proofs can be found in [1].

2 Stochastic Timed Games

We use standard notations for the set of reals ($\mathbb{R}$), rationals ($\mathbb{Q}$), and integers ($\mathbb{Z}$), and add subscripts to indicate additional constraints (for instance $\mathbb{R}_{\geq 0}$ is for the set of non-negative reals). Let $\mathcal{C}$ be a finite set of real-valued variables called *clocks*. A *valuation* on $\mathcal{C}$ is a function $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$. We assume an arbitrary but fixed ordering on the clocks and write x_i for the clock with order i. This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$. Abusing notations slightly, we use a valuation on $\mathcal{C}$ and a point in $\mathbb{R}_{\geq 0}^{|\mathcal{C}|}$ interchangeably. For a subset of clocks $X \subseteq \mathcal{C}$ and valuation $\nu \in \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$, we write $\nu[X{:=}0]$ for the valuation where $\nu[X{:=}0](x) = 0$ if $x \in X$, and $\nu[X{:=}0](x) = \nu(x)$ otherwise. For $t \in \mathbb{R}_{\geq 0}$, write $\nu + t$ for the valuation defined by $\nu(x) + t$ for all $x \in X$. The valuation $\mathbf{0} \in \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{C}$. A clock constraint over $\mathcal{C}$ is a subset of $\mathbb{R}_{\geq 0}^{|\mathcal{C}|}$ defined by a (finite) conjunction of constraints of the form $x \bowtie k$, where $k \in \mathbb{Z}_{\geq 0}$, $x \in \mathcal{C}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. We write $\varphi(\mathcal{C})$ for the set of clock constraints. For a constraint $g \in \varphi(\mathcal{C})$, and a valuation ν, we write $\nu \models g$ to represent the fact that valuation ν satisfies constraint g (defined in a natural way). A timed automaton (TA) [2] is a tuple $\mathcal{A} = (L, \mathcal{C}, E, \mathcal{I})$ such that (i) L is a finite set of locations, (ii) $\mathcal{C}$ is a finite set of clocks, (iii) $E \subseteq L \times \varphi(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ is a finite set of edges, (iv) $\mathcal{I} : L \to \varphi(\mathcal{C})$ assigns an invariant to each location. A state s of a timed automaton is a pair $s = (\ell, \nu) \in L \times \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$ such that $\nu \models \mathcal{I}(\ell)$ (the clock valuation should satisfy the invariant of the location). If $s = (\ell, \nu)$, and $t \in \mathbb{R}_{\geq 0}$, we write $s + t$ for the state $(\ell, \nu + t)$. A transition (t, e) from a state $s = (\ell, \nu)$ to a state $s' = (\ell', \nu')$ is written as $s \xrightarrow{t,e} s'$ if $e = (\ell, g, C, \ell') \in E$, such that $\nu + t \models g$, and for every $0 \leq t' \leq t$ we have $\nu + t' \models \mathcal{I}(\ell)$ and $\nu' = \nu + t[C{:=}0](x)$. A run is a finite or infinite sequence of transitions $\rho = s_0 \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \ldots$ of states and transitions. An edge e is enabled from s whenever there is a state s' such that $s \xrightarrow{0,e} s'$. Given a state s of $\mathcal{A}$ and an edge e, we define $I(s, e) = \{t \in \mathbb{R}_{\geq 0} \mid s \xrightarrow{t,e} s'\}$ for some s' and $I(s) = \bigcup_{e \in E} I(s, e)$. We say that $\mathcal{A}$ is non-blocking iff for all states s, $I(s) \neq \emptyset$. Now we are ready to introduce stochastic timed games.

▶ **Definition 1** (Stochastic Timed Games [12]). A *stochastic timed game (STG)* is a tuple $\mathcal{G} = (\mathcal{A}, (L_\Box, L_\Diamond, L_\bigcirc), \omega, \mu)$ where

- $\mathcal{A}{=}(L, \mathcal{C}, E, \mathcal{I})$ is a timed automaton;
- $L_\Box, L_\Diamond$, and $L_\bigcirc$ form a partition of L characterizing the set of locations controlled by players $\Box$ and $\Diamond$ and the stochastic player, respectively;
- $\omega : E(L_\bigcirc) \to \mathbb{Z}_{>0}$ assigns some positive weight to each edge originating from $L_\bigcirc$ (notation $E(L_\bigcirc)$);
- μ is a function assigning a measure over $I(s)$ to all states $s \in L_\bigcirc \times \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$ satisfying the properties that $\mu(s)(I(s)) = 1$ and for Lebesgue measure λ, if $\lambda(I(s)) > 0$ then for each measurable set $B \subseteq I(s)$ we have $\lambda(B) = 0$ if and only if $\mu(s)(B) = 0$.

The timed automaton $\mathcal{A}$ is said equipped with uniform distributions over delays if for every state s, $I(s)$ is bounded, and $\mu(s)$ is the uniform distribution over $I(s)$. The timed automaton $\mathcal{A}$ is said equipped with exponential distributions over delays whenever, for every state s, either $I(s)$ has Lebesgue measure zero, or $I(s){=}\mathbb{R}_{\geq 0}$ and for every location l, there is a positive rational α_l such that $\mu(s)(I(s)){=}\int_{t \in I} \alpha_l e^{-\alpha_l t} dt$. For $s \in L_\bigcirc \times \mathbb{R}_{\geq 0}^{|\mathcal{C}|}$, both delays and discrete moves will be chosen probabilistically: from s, a delay t is chosen following the probability distribution over delays $\mu(s)$. Then, from state $s + t$, an enabled edge is selected following a discrete probability distribution that is given in a usual way with the weight function w: in state $s + t$, the probability of edge e (if enabled), denoted $p(s + t)(e)$ is $w(e) / \sum_{e'} \{w(e') \mid e'$ is enabled in $s + t\}$. This way of probabilizing behaviours in timed automata has been presented in [9].

If $L_\Box{=}\emptyset$ then the STGs are called $1\frac{1}{2}$ STGs or $1\frac{1}{2}$-player STGs while STGs with $L_\Box = L_\Diamond{=}\emptyset$ are called $\frac{1}{2}$ STGs or $\frac{1}{2}$-player STGs or STAs. We often refer to $l{\in}L_\bigcirc$ as stochastic nodes, $l \in L_\Box$ as box (or $\Box$) nodes and $l \in L_\Diamond$ as diamond (or $\Diamond$) nodes.

Fix a STG $\mathcal{G} = (\mathcal{A}, (L_\Box, L_\Diamond, L_\bigcirc), \omega, \mu)$ with $\mathcal{A} = (L, \mathcal{C}, E, \mathcal{I})$ for the rest of this section.

Strategies, Profiles, and Runs. A strategy for Player $\Box$ (resp. $\Diamond$) is a function that maps a finite run $\rho = s_0 \xrightarrow{t_0,e_0} s_1 \xrightarrow{t_1,e_1} \ldots s_n$ to a pair (t, e) such that $s_n \xrightarrow{t,e} s'$ for some state s', whenever $s_n = (\ell_n, \nu_n)$ and $\ell_n \in L_\Box$ (resp. $\ell_n \in L_\Diamond$). In this work we focus on deterministic strategies, though randomized strategies could also make sense; nevertheless understanding the case of deterministic strategies is already challenging. A strategy profile is a pair $\Lambda = (\lambda_\Diamond, \lambda_\Box)$ where $\lambda_\Diamond, \lambda_\Box$ respectively are strategies of players $\Diamond$ and $\Box$. In order to measure probabilities of certain sets of runs, the following measurability condition is imposed on strategy profiles $\Lambda = (\lambda_\Diamond, \lambda_\Box)$: for every finite sequence of edges $e_1, \ldots, e_n$ and every state s, the function $\kappa_s : (t_1, \ldots, t_n) \to (t, e)$ defined by $\kappa_s(t_1, \ldots, t_n) = (t, e)$ iff $\Lambda(s \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \ldots \xrightarrow{t_n,e_n} s_n) = (t, e)$, should be measurable.

Given a finite run ρ ending in state s_0, and a strategy profile Λ, define $\mathit{Runs}(\mathcal{G}, \rho, \Lambda)$ (resp. $\mathit{Runs}^\omega(\mathcal{G}, \rho, \Lambda)$) to be the set of all finite (resp. infinite) runs generated by Λ after prefix ρ; that is, the set of all runs of the automaton satisfying the following condition: If $s_i = (\ell_i, \nu_i)$ and $\ell_i \in L_\Diamond$ (resp. $\ell_i \in L_\Box$), then $\lambda_\Diamond$ (resp. $\lambda_\Box$) returns (t_{i+1}, e_{i+1}) when applied to $\rho \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} \ldots \xrightarrow{t_i,e_i} s_i$. Given a finite sequence $e_1, \ldots, e_n$ of edges, a symbolic path $\pi_\Lambda(\rho, e_1 \ldots e_n)$ is defined as

$$\pi_\Lambda(\rho, e_1 \ldots e_n) = \{\rho' \in \mathit{Runs}(\mathcal{G}, \rho, \Lambda) \mid \rho' = \rho \xrightarrow{t_1,e_1} s_1 \xrightarrow{t_2,e_2} s_2 \ldots \xrightarrow{t_n,e_n} s_n, \text{ with } t_i \in \mathbb{R}_{\geq 0}\}.$$

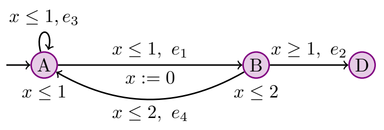
When Λ is clear, we simply write $\pi(\rho, e_1 \ldots e_n)$.

Probability Measure of a Strategy Profile. Given a strategy profile $\Lambda = (\lambda_\Diamond, \lambda_\Box)$, and a finite run ρ ending in $s = (\ell, \nu)$, a measure $\mathcal{P}_\Lambda$ can be defined on the set $\mathit{Run}(\mathcal{G}, \rho, \Lambda)$, following [12]: First, for the empty sequence ϵ, $\mathcal{P}_\Lambda(\pi(\rho, \epsilon)) = 1$, and

- If $\ell \in L_\Diamond$ (resp. $\ell \in L_\Box$), and $\lambda_\Diamond(\rho) = (t, e)$ (resp. $\lambda_\Box(\rho) = (t, e)$), then $\mathcal{P}_\Lambda(\pi(\rho, e_1 \ldots e_n))$ equals 0 if $e_1 \neq e$ and equals $\mathcal{P}_\Lambda(\pi(\rho \xrightarrow{t,e} s', e_2 \ldots e_n))$, otherwise.
- If $\ell \in L_\bigcirc$, $\mathcal{P}_\Lambda(\pi(\rho, e_1 \ldots e_n)) = \int_{t \in I(s,e_1)} p(s+t)(e_1) \cdot \mathcal{P}_\Lambda(\pi(\rho \xrightarrow{t,e_1} s', e_2 \ldots e_n))\, d\mu(s)(t)$ where $s \xrightarrow{t,e_1} s'$ for every $t \in I(s, e_1)$.

The cylinder generated by a symbolic path is defined as follows: an infinite run ρ'' is in the cylinder generated by $\pi_\Lambda(\rho, e_1, \ldots, e_n)$ denoted $\mathsf{Cyl}(\pi_\Lambda(\rho, e_1, \ldots, e_n))$ if ρ'' is in $\mathit{Runs}^\omega(\mathcal{G}, \rho, \Lambda)$ and there is a finite prefix ρ' of ρ'' such that $\rho' \in \pi_\Lambda(\rho, e_1, \ldots, e_n)$. It is routine to extend the above measure $\mathcal{P}_\Lambda$ to cylinders, and thereafter to the generated σ-algebra; extending [9], one can show this is a probability measure over $\mathit{Runs}^\omega(\mathcal{G}, \rho, \Lambda)$.

Example. An example of a STG is shown in the adjoining figure. In this example all the locations belong to stochastic player (this is an $\frac{1}{2}$ STG) and there is only one clock named x.



We explain here the method for computing probabilities. We assume uniform distribution over delays at all states, and initial state $s_0 = (A, 0)$. Let $d\mu_{(A,0)}$ be the uniform distribution over $[0, 1]$ and $d\mu_{(B,0)}$ uniform distribution over $[0, 2]$. Then $\mathcal{P}(\pi((A, 0), e_1e_2))$ equals $\frac{1}{8}$:

$$\int_0^1 \frac{\mathcal{P}(\pi((B,0), e_2))}{2} d\mu_{(A,0)}(t) = \int_0^1 \frac{1}{2}(\int_1^2 \frac{1}{2} d\mu_{(B,0)}(u))\, d\mu_{(A,0)}(t) = \frac{1}{2}\int_0^1 (\int_1^2 \frac{1}{2}\frac{1}{2} du))\, dt)$$

Reachability Problem. We study the reachability problem for STGs, stated as follows. Given a STG $\mathcal{G}$ with a set T of target locations, an initial state s_0 and a threshold $\bowtie p$ with $p \in [0, 1] \cap \mathbb{Q}$, decide whether there is a strategy $\lambda_\Diamond$ for Player $\Diamond$ such that for every strategy $\lambda_\Box$ for Player $\Box$, $\mathcal{P}_\Lambda(\{\rho \in \mathit{Run}(\mathcal{G}, s_0, \Lambda) \mid \rho \text{ visits } T\}) \bowtie p$, with $\Lambda = (\lambda_\Diamond, \lambda_\Box)$. There are two categories of reachability questions:

1. **Quantitative reachability**: The constraint on probability involves $0 < p < 1$.
2. **Qualitative reachability**: The constraint on probability involves $p \in \{0, 1\}$.

The key results of the paper are the following:

▶ **Theorem 2.** *The quantitative reachability problem is*
1. *Undecidable for $1\frac{1}{2}$ STGs with 4 or more clocks;*
2. *Undecidable for $2\frac{1}{2}$ STGs with 5 or more clocks even under the time-bounded semantics;*
3. *Decidable for $1\frac{1}{2}$ and $2\frac{1}{2}$* initialized *STGs with one clock.*

Mentioned restrictions (time-bounded semantics and initialized) will be introduced when needed. In Section 3, we deal with the quantitative reachability problem, where we show strengthened undecidability results. In Section 4, we explore a new model of STGs with a single clock and an initialized restriction to recover decidability for the quantitative reachability problem. In Section 5, we discuss the intrinsic difficulties and challenges ahead, summarize our key contributions and conjectures.

3 Undecidability Results for Quantitative Reachability

In this section, we focus on the quantitative reachability problem for STGs. We strengthen the existing undecidability result, which holds for $2\frac{1}{2}$ STGs [12], in two distinct directions. First, we show the undecidability of the quantitative reachability problem in $1\frac{1}{2}$ STGs, improving from $2\frac{1}{2}$. Second, we show the undecidability of the quantitative reachability problem for $2\frac{1}{2}$ STGs even in the time-bounded setting.

For both results, given a two-counter machine, we construct respectively, $1\frac{1}{2}$ and $2\frac{1}{2}$ STGs whose building blocks are the modules for the instructions in the two-counter machine. The objective of player $\Diamond$ is linked to a faithful simulation of various increment, decrement and zero-test instructions of the two-counter machine by choosing appropriate delays to adjust the clocks to reflect changes in counter values. However, the two proofs differ in how this verification is done and even in the problem from which the reduction is done, i.e., halting/non-halting for two-counter machines. This results in two quite different and non-trivial reductions as described in Subsection 3.1 and Subsection 3.2 respectively.

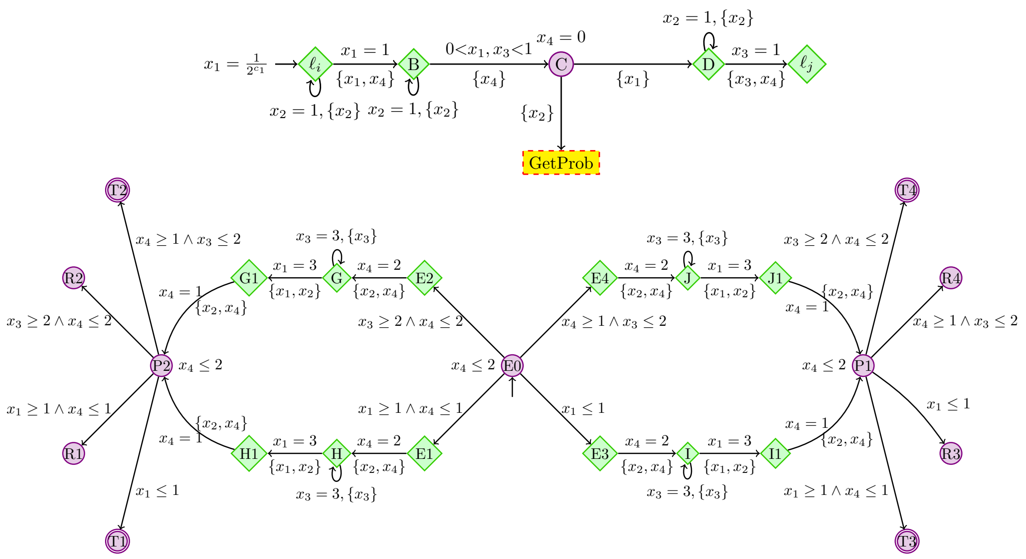
3.1 Quantitative reachability for $1\frac{1}{2}$ STGs

As mentioned above, in the case of $1\frac{1}{2}$ STGs we improve the corresponding result of [12] for $2\frac{1}{2}$ STGs. But unlike in [12], we reduce from the *non-halting problem* for two-counter machines to the existence of a winning strategy for Player $\Diamond$ with the desired objective. This crucial difference makes it possible for the probabilistic player to verify the simulation performed by player $\Diamond$.

▶ **Theorem 3.** *The quantitative reachability problem is undecidable for $1\frac{1}{2}$ STGs with ≥ 4 clocks.*

Let $\mathcal{M}$ be a two-counter machine. Our reduction uses a $1\frac{1}{2}$ player STG $\mathcal{G}$ with four clocks and uniform distributions over delays, and a set of target locations T such that player $\Diamond$ has a strategy to reach T with probability $\frac{1}{2}$ iff $\mathcal{M}$ does not halt. Each instruction (increment, decrement and test for zero value) is specified using a module. The main invariant in our reduction is that upon entry into a module, we have that $x_1 = \frac{1}{2^{c_1}}, x_2 = \frac{1}{2^{c_2}}$, $x_3 = x_4 = 0$, where c_1 (resp. c_2) is the value of counter C_1 (resp. C_2) in $\mathcal{M}$.

We outline the simulation of an increment instruction « ℓ_i : increment counter C_1, goto ℓ_j » in Figure 1 (top). The module is entered with values $x_1 = \frac{1}{2^{c_1}}, x_2 = \frac{1}{2^{c_2}}$, $x_3 = x_4 = 0$. A time $1 - \frac{1}{2^{c_1}}$ is spent at location ℓ_i, so that at location B we have $x_1 = 0$, $x_2 = \frac{1}{2^{c_2}} + 1 - \frac{1}{2^{c_1}}$ (or $\frac{1}{2^{c_2}} - \frac{1}{2^{c_1}}$, if $c_2 > c_1$ – we write in all cases $\frac{1}{2^{c_2}} + 1 - \frac{1}{2^{c_1}} \mod 1$), $x_3 = 1 - \frac{1}{2^{c_1}}$, $x_4 = 0$.



■ **Figure 1** The Increment c_1 module on the top and the GetProb gadget below

An amount of time $t \in (0, \frac{1}{2^{c_1}})$ is spent at B, which is decided by Player $\Diamond$. We rewrite this as $t = \frac{1}{2^{c_1+1}} \pm \epsilon$ for $-\frac{1}{2^{c_1+1}} < \epsilon < \frac{1}{2^{c_1+1}}$. This is because, ideally we want t to be $\frac{1}{2^{c_1+1}}$ and want to consider any deviation as an error.

Now at C, we have $x_1 = t$, $x_2 = \frac{1}{2^{c_2}} + 1 - \frac{1}{2^{c_1}} + t \bmod 1$, $x_3 = 1 - \frac{1}{2^{c_1}} + t$, $x_4 = 0$. The computation proceeds to D with probability $\frac{1}{2}$, and the location ℓ_j corresponding to the next instruction ℓ_j is reached with $x_1 = \frac{1}{2^{c_1}} - t$, $x_2 = \frac{1}{2^{c_2}}$, $x_3 = x_4 = 0$. On the other hand, with probability $\frac{1}{2}$, the gadget *GetProb* is reached. The gadget *GetProb* has 4 target locations $T1, T2, T3, T4$, which we will show are reached with probability $\frac{1}{2}$ from the start location $E0$ of *GetProb* iff $t = \frac{1}{2^{c_1+1}}$. Thus, in this case when $t = \frac{1}{2^{c_1+1}}$, we reach ℓ_j with the values $x_1 = \frac{1}{2^{c_1+1}}$, $x_2 = \frac{1}{2^{c_2}}$, $x_3 = x_4 = 0$ which implies that c_1 has been incremented correctly according to our encoding. We now look at the gadget *GetProb*.

▶ **Lemma 4.** *For any value $\epsilon \in (-\frac{1}{2^{c_1+1}}, \frac{1}{2^{c_1+1}})$, the probability to reach a target location in GetProb from $E0$ is $\frac{1}{2}(1-4\epsilon^2)$ $(\leq \frac{1}{2})$. Further this probability is equal to $\frac{1}{2}$ iff $\epsilon = 0$.*

Proof. Note that when the start location $E0$ of *GetProb* is reached, we have $x_1 = \frac{1}{2^{c_1+1}} + \epsilon$, $x_2 = 0$, $x_3 = 1 - \frac{1}{2^{c_1+1}} + \epsilon$, $x_4 = 0$. A total of 2 time units can be spent at $E0$. It can be seen that transitions to $E3$ and $E4$ are respectively enabled with the time intervals $[0, 1 - \frac{1}{2^{c_1+1}} - \epsilon]$ and $[1, 1 + \frac{1}{2^{c_1+1}} - \epsilon]$. Similarly, reaching $E1$ and $E2$ are enabled by the time intervals $[1 - \frac{1}{2^{c_1+1}} - \epsilon, 1]$ and $[1 + \frac{1}{2^{c_1+1}} - \epsilon, 2]$. The sum of probabilities of reaching either $E3$ or $E4$ is thus $\frac{1}{2}(1-2\epsilon)$. Similarly, the sum of probabilities for reaching $E1$ or $E2$ is $\frac{1}{2}(1+2\epsilon)$. The locations $P1, P2$ are then reached with the values $x_1 = \frac{1}{2^{c_1+1}} + \epsilon$, $x_2 = 0$, $x_3 = 1 - \frac{1}{2^{c_1+1}} + \epsilon$, $x_4 = 0$. The probability of reaching the target locations $T3$ or $T4$ (i.e., through $P1$) from $E0$ is hence $\frac{1}{2}(1+2\epsilon)\frac{1}{2}(1-2\epsilon) = \frac{1}{4}(1-4\epsilon^2)$, while the probability of reaching a target location $T1$ or $T2$ (i.e., through $P2$) from $E0$ is $\frac{1}{2}(1+2\epsilon)\frac{1}{2}(1-2\epsilon) = \frac{1}{4}(1-4\epsilon^2)$. Thus, the probability of reaching a target location (one of $T1, T2, T3, T4$) in *GetProb* is, $\frac{1}{2}(1-4\epsilon^2)$, which is always $\leq \frac{1}{2}$. This completes the first statement of the lemma. Further, from the expression, we immediately have that the probability to reach a target location in *GetProb* from $E0$ is $\frac{1}{2}$ iff $\epsilon = 0$. ◀

The decrement c_1, increment c_2 as well as decrement c_2 modules are similar and these as well as the zero test modules can be found in [1].

▶ **Lemma 5.** *Player $\diamond$ has a strategy to reach the (set of) target locations in $\mathcal{G}$ with probability $\frac{1}{2}$ iff the two-counter machine does not halt.*

Proof. Suppose the two-counter machine halts (say in k steps). Then there are two cases: (a) the simulations of all instructions are correct in $\mathcal{G}$. In this case, the target location can be reached in either of the first k steps. By Lemma 4, the probability of reaching a target location in the first k steps is the summation $\frac{1}{2}.\frac{1}{2} + (\frac{1}{2})^2.\frac{1}{2} + (\frac{1}{2})^3.\frac{1}{2} + \cdots + (\frac{1}{2})^k.\frac{1}{2} < \frac{1}{2}$. (b) Player $\diamond$ made an error in the computation in the first k steps. But then again by Lemma 4, the finite sum obtained is $< \frac{1}{2}$ (since in the error step(s), the probability to reach target locations is $\frac{1}{2} - 4\epsilon^2 < \frac{1}{2}$). Thus, if the two-counter machine halts, under any strategy of $\diamond$ player, the probability to reach the target locations is $< \frac{1}{2}$.

On the other hand, suppose the two-counter machine does not halt. Then, if Player $\diamond$ chooses the strategy which faithfully simulates all instructions of the two-counter machine, the probability to reach the (set of) target locations is given by the infinite sum $\sum_{i=0}^{\infty}(\frac{1}{2})^i\frac{1}{2} = \frac{1}{2}$. Any other strategy of Player $\diamond$ corresponds to performing at least one error in the simulation. In this case, the infinite sum obtained has at least one term of the form $(\frac{1}{2})^k(\frac{1}{2} - 4\epsilon^2)$, for $\epsilon^2 > 0$. Clearly, such an infinite sum does not sum to $\frac{1}{2}$. This concludes the proof. ◀

The previous proof can be changed for other thresholds and to use unbounded intervals and exponential distributions.

3.2 Time-bounded quantitative reachability for $2\frac{1}{2}$ STGs

In this section, we tackle the *time-bounded* version of the quantitative reachability problem. This strengthens the definition of reachability by considering a given time bound Δ, and requiring that $\mathcal{P}_\sigma(\{\rho \in Run(\mathcal{G}, s_0, \sigma) \mid \rho \text{ visits } T \text{ within } \Delta \text{ time units}) \bowtie p$.

In this new framework, we show the undecidability of the quantitative reachability problem for $2\frac{1}{2}$ STGs. We reduce from the *halting* problem for two-counter machines (unlike in the previous section, where our reduction was from the *non-halting* problem), using Player $\square$ to verify the correctness of the simulation. The complication here is that the total time spent should be bounded and hence we cannot allow arbitrary time elapses. We will in fact show a global time bound of $\Delta = 5$ for this reduction.

▶ **Theorem 6.** *The time-bounded quantitative reachability problem is undecidable for $2\frac{1}{2}$ STGs with ≥ 5 clocks.*

Proof. Let $\mathcal{M}$ be a two-counter machine. We construct an STG with 5 clocks such that the two-counter machine $\mathcal{M}$ halts iff Player $\diamond$ has a strategy to reach some desired locations with probability $\frac{1}{2}$, whatever Player $\square$ does, and such that the total time spent is bounded by $\Delta = 5$ units.

The main idea behind the proof is that the total time spent in the simulation of the k^{th} instruction will be $\frac{1}{2^k}$. We thus get a decreasing sequence of times $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8} \dots$ for simulating the instructions $1, 2 \dots$ and so on. In total, we will use five clocks x_1, x_2, z, a and b. The clocks x_1 and x_2 are used encode the counter values (along with the current instruction number) such that at the end of the k^{th} instruction, if k is even the values are encoded in x_1 and if k is odd they are encoded in x_2 as follows:

$(\boldsymbol{enc_{x_1}})$ k is even and $x_1 = \frac{1}{2^{k+c_1}3^{k+c_2}}$, $x_2 = 0$, $z = 1 - \frac{1}{2^k}$, $a = b = 0$;

$(\boldsymbol{enc_{x_2}})$ k is odd and $x_2 = \frac{1}{2^{k+c_1}3^{k+c_2}}$, $x_1 = 0$, $z = 1 - \frac{1}{2^k}$, $a = b = 0$;

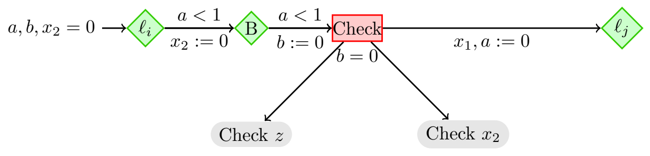


Figure 2 Module for incrementing C_1 (after an even number of steps)

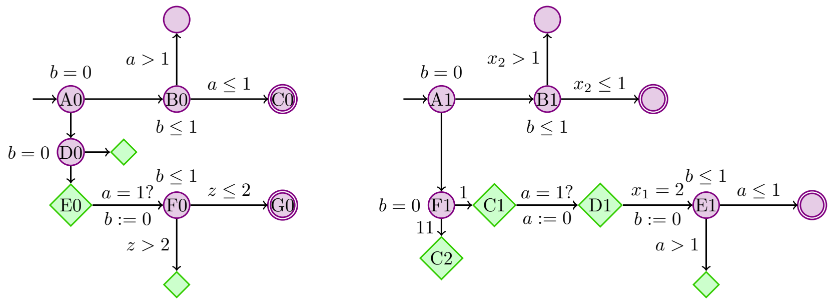


Figure 3 Widgets 'Check z' (left) and 'Check x_2' (right).

We start the simulation with $x_1 = 1, x_2 = z = 0 = a = b$ corresponding to the initial instruction ($k = 0$) and the fact that the values of C_1, C_2 are 0. Moreover, if $x_1 = \frac{1}{2^{k+c_1}3^{k+c_2}}$ at the end of the kth instruction, and if the $(k+1)$th instruction is an increment C_1 instruction, then at the end of the $(k+1)$th instruction, $x_2 = \frac{1}{2^{k+c_1+2}3^{k+c_2+1}}$. Clock z keeps a separate track of the number of instructions simulated so far, by having a value $1 - \frac{1}{2^k}$ after completing the simulation of k instructions. Clocks a and b are auxiliary clocks that we need for the simulation. We assume uniform distribution over delays in probabilistic locations. If no weight is written on an edge, it is assumed to be 1. We outline the simulation of a increment instruction « ℓ_i : increment counter C_1, goto ℓ_j » in Figure 2, assuming this is the $(k+1)$th instruction, where k is even. Thus, at the end of the k first instructions, we have $x_1 = \frac{1}{2^{k+c_1}3^{k+c_2}}$, $z = 1 - \frac{1}{2^k}$ and $a = b = x_2 = 0$ (the other case of odd k, i.e., (enc_{x_2}) encoding is symmetric). At the end of this $(k+1)$th instruction's simulation, the value of clock z should be $z = 1 - \frac{1}{2^{k+1}}$ to mark the end of the $(k+1)^{th}$ instruction. Also, we must obtain $x_2 = \frac{x_1}{2^2 \cdot 3} = \frac{x_1}{12}$, marking the successful increment of C_1.

Player $\diamond$ elapses times t_1, t_2 in locations ℓ_i, B. When the player $\Box$ location *Check* is reached, we have $a = t_1 + t_2 = t$ and $x_2 = t_2$, $z = 1 - \frac{1}{2^k} + t_1 + t_2$. Player $\Box$ has three possibilities : (1) to continue the simulation going to ℓ_{k+2}, (2) verify that $t_2 = \frac{1}{2^{k+c_1+2}3^{k+c_2+1}}$ by going to the widget 'Check x_2' or (3) verify that $t_1 + t_2 = \frac{1}{2^{k+1}}$ by going to the widget 'Check z'. These widgets are given in Figure 3. The probability of reaching a target location in widget 'Check z' is $\frac{1}{2}(1-t) + \frac{1}{4}\frac{1}{2^k} = \frac{1}{2}$ iff $t = \frac{1}{2^{k+1}}$. In widget 'Check x_2', the transitions from $F1$ to $C1$ and $F1$ to $C2$ are taken with probability $\frac{1}{12}$ and $\frac{11}{12}$, respectively since the weights of edges connecting F1,C1 and F1,C2 are respectively 1 and 11. With this, for $n = \frac{1}{2^{k+c_1}3^{k+c_2}}$, the probability of reaching a target location in 'Check x_2' is $\frac{1}{2}(1-t_2) + \frac{n}{24} = \frac{1}{2}$ iff $t_2 = \frac{n}{12}$.

Time elapse for Increment. If player $\Box$ goes ahead with the simulation, the time elapse for the $(k+1)$th instruction is $t_1 + t_2 = \frac{1}{2^{k+1}}$. Consider the case when player $\Box$ goes in to 'Check z'. The time elapse till now is $\frac{1}{2} + \cdots + \frac{1}{2^{k+1}}$. The time spent in the 'Check z' widget is as follows: one unit is spent at location $B0$, one unit at location $F0$, and $1 - t$ units at location

$E0$. Thus, ≤ 3 units are spent at the 'Check z' widget. Similarly, the time spent in the 'Check x_2' widget is one unit at $B1$, $1-t$ units at $C1$, $1-n$ units at $D1$ and one unit at $E1$. Thus a time ≤ 4 is spent in 'Check x_2'. Thus, the time spent till the $(k+1)$th instruction is $\leq \frac{1}{2} + \dots \frac{1}{2^{k+1}} + 4$ if player $\square$ goes in for a check, and otherwise it is $\frac{1}{2} + \dots + \frac{1}{2^{k+1}}$.

Other increment, decrement, zero-check Instructions. The main module corresponding to *increment* C_2 and decrement C_1, C_2 is the same as in Figure 2. The only change needed is in the 'Check x_2' widget. While incrementing c_2, we need $x_2 = \frac{x_1}{2 \cdot 3^2} = \frac{x_1}{18}$. This is done by changing the weights on the outgoing edges from $F1$ to $C1$ and $C2$ to 1 and 17 respectively. Similarly, while *decrementing* C_1, we need $x_2 = \frac{x_1}{3}$. This is done by changing the weights on the outgoing edges of $F1$ to $1, 2$ respectively. Lastly, to *decrement* C_2, we need $x_2 = \frac{x_1}{2}$, and in this case the weights are 1 each.

The zero check module is a bit more complicated. The broad idea is that we use a diamond node to guess whether the current clock (say C_1) value is zero and branch into two sides (zero and non-zero). Then we use a box node on each branch to verify that the guess was correct. If correct, we proceed with the next instruction, if not, we check this by going to a special widget. In this widget, we can reach a target node with probability $\frac{1}{2}$ iff the guess is correct. The details of this widget and the proof that all these simulations can be done in time bounded by $\Delta \leq 5$ units is given in [1]. ◀

4 Decidability results for quantitative reachability

We have seen in the previous section that the quantitative reachability problem is undecidable in $1\frac{1}{2}$ STGs with ≥ 4 clocks. In this section we study the *quantitative reachability* problem in the setting of $1\frac{1}{2}$ STGs *with a single clock*. In [8], the quantitative reachability problem in $\frac{1}{2}$ STGs with a single clock, under certain restrictions, was shown to be decidable by reducing it to the quantitative reachability problem for finite Markov chains. In our case, we lift this to $1\frac{1}{2}$ STGs with a single clock, under similar restrictions, by reducing to the quantitative reachability problem in finite Markov decision processes (MDPs in short).

For the rest of this section, we consider a $1\frac{1}{2}$ STG $\mathcal{G} = (\mathcal{A}, (L_\diamond, L_\bigcirc), \omega, \mu)$ with a single clock denoted x. We write $c_{\max}$ for the maximal constant appearing in a guard of $\mathcal{G}$. We assume w.l.o.g. that target locations belong to player $\diamond$ (a slight modification of the construction can be done if this is not the case). In the following, when we talk about regions, we mean the clock regions from the classical region construction for timed automata [2, 19]: since $\mathcal{G}$ has a single clock, regions in this case are simply either singletons $\{c\}$ with $c \in \mathbb{Z}_{\geq 0} \cap [0; c_{\max}]$, or open intervals $(c, c+1)$ with $c \in \mathbb{Z}_{\geq 0} \cap [0; c_{\max} - 1]$, or the unbounded interval $(c_{\max}; +\infty)$. While region automata are standardly finite automata, we build here from $\mathcal{G}$ a *region STG* $\mathcal{G}_\mathcal{R}$, which has only clock constraints defined by regions (that is, either $x = c$ or $c < x < c+1$ or $x > c_{\max}$), and such that each location of $\mathcal{G}_\mathcal{R}$ is indeed a pair (ℓ, R) where ℓ is a location of $\mathcal{G}$ and R a region (region R is for the region which is hit when entering the location). While it is not completely standard, this kind of construction has been already used in [9, 8, 12], and questions asked on $\mathcal{G}$ can be equivalently asked (and answered) on $\mathcal{G}_\mathcal{R}$. Now, we make the following restrictions on $\mathcal{G}_\mathcal{R}$ (which yields restrictions to $\mathcal{G}$), which we denote $(\star)$:

1. The TA $\mathcal{A}_\mathcal{R}$ is assumed to be structurally non-Zeno: any bounded cycle of $\mathcal{A}_\mathcal{R}$ (a cycle in which all edges have a non-trivial upper-bound) contains at least one location whose associated region is the zero region (i.e., edge leading to it, resets the clock).

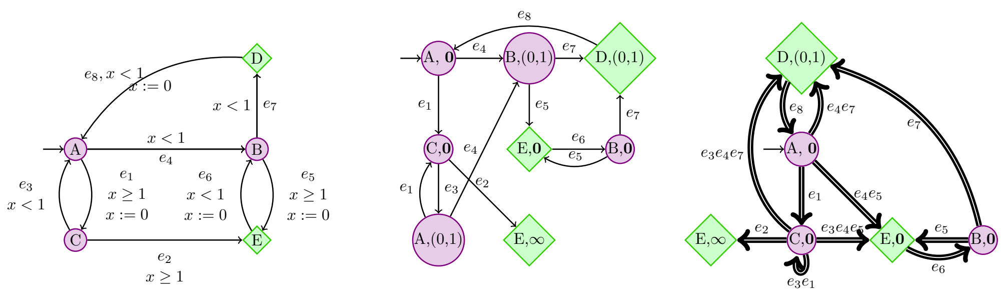


Figure 4 An initialized $1\frac{1}{2}$ player STG $\mathcal{G}$, its region game graph $\mathcal{G}_\mathcal{R}$ and the MDP abstraction $M_\mathcal{G}$.

2. For every state $s = ((\ell, r), \nu)$ of $\mathcal{G}_\mathcal{R}$ such that $\ell \in L_\bigcirc$, $I(s) = \mathbb{R}_{\geq 0}$, and μ_s is an exponential distribution; Furthermore the rate of μ_s only depends on location ℓ.
3. $\mathcal{G}_\mathcal{R}$ is *initialized*, that is, any edge from a non-stochastic location to a stochastic location resets the clock x.

While the first two assumptions are already made in [8], even in the $\frac{1}{2}$ player case, the third condition is new. In the following we denote $\mathbf{0}$ for the region $\{0\}$ and ∞ for the unbounded region $(c_{\max}; +\infty)$. We now show how to obtain an MDP from the STG $\mathcal{G}_\mathcal{R}$. The construction is illustrated on Figure 4. A node (ℓ, R) of $\mathcal{G}_\mathcal{R}$ with $\ell \in L_\bigcirc$ is *deletable* if R is neither the region 0 nor the region ∞. In Figure 4, $(B, (0,1))$ and $(A, (0,1))$ in $\mathcal{G}_\mathcal{R}$ are what we call deletable nodes. Then, we recursively remove all deletable nodes $\mathcal{G}_\mathcal{R}$ while labelling remaining paths with (finite) sequences of edges; each surviving edge is labelled by the probability of the (provably) finitely many sequences of edges appearing in the label. One can prove that this object is actually an MDP, which we denote $M_\mathcal{G}$. Target states in $M_\mathcal{G}$ are defined as the pairs (ℓ, R) where ℓ is a target location in $\mathcal{G}$. We can prove (see [1]) that:

▶ **Lemma 7.** *If $\mathcal{G}$ is an $1\frac{1}{2}$ player STG with one clock satisfying the hypotheses $(\star)$, then $M_\mathcal{G}$ is an MDP such that: (a) for every strategy $\lambda_\diamond$ of player $\diamond$ in $\mathcal{G}$, we can construct a strategy $\sigma_\diamond$ of player $\diamond$ in $M_\mathcal{G}$ such that the probability of reaching a target location in $\mathcal{G}$ is the same as the probability of reaching a target state in $M_\mathcal{G}$; and (b) for every strategy $\sigma_\diamond$ of player $\diamond$ in $M_\mathcal{G}$, we can construct a strategy $\lambda_\diamond$ of player $\diamond$ in $\mathcal{G}$ such that the probability of reaching a target location in $M_\mathcal{G}$ is the same as the probability of reaching a target state in $\mathcal{G}$.*

This lemma allows to reduce the quantitative reachability problem from the $1\frac{1}{2}$ STG $\mathcal{G}$ to the MDP $M_\mathcal{G}$.

As an example, in Figure 4, we show a $1\frac{1}{2}$ player STG $\mathcal{G}$, its region game graph $\mathcal{G}_\mathcal{R}$ (guards omitted for readability) and the MDP abstraction $M_\mathcal{G}$. Note that all $\diamond$ nodes remain, while only those stochastic nodes with regions $\mathbf{0}$ and ∞ are retained in $M_\mathcal{G}$. The stochastic nodes $(B, (0,1))$ as well as $(C, (0,1))$ are deleted in $M_\mathcal{G}$. On deleting nodes from the region graph, the probability on the edges of $M_\mathcal{G}$ is the probability of the respective paths from the region graph. For example, the edge from $(A, 0)$ to $(D, (0,1))$ is labelled with e_4e_7 by deleting $(B, (0,1))$.

Thus, the remaining thing that has to be addressed now is how to compute the probabilties and compare them with a rational threshold. The first thing to note is that the edges of the MDP are all labelled with polynomials over exponentials obtained using the delays from the underlying game with rational coefficients. For example, in Figure 4, in the MDP in the rightmost picture, we obtain: $\mathcal{P}(e_1)$=$\mathcal{P}(e_2)$=$\mathcal{P}(e_5)$=e^{-1}, $\mathcal{P}(e_6, e_7, e_8)$=$1-e^{-1}$,

$\mathcal{P}(e_4e_5)=e^{-1}-e^{-2}$, $\mathcal{P}(e_4e_7)=1-2e^{-1}$, $\mathcal{P}(e_3e_4e_7)=2-5e^{-1}+e^{-2}$, $\mathcal{P}(e_3e_4e_5)=1-e^{-1}+e^{-2}$, $\mathcal{P}(e_3e_1)=\frac{1}{2}(1-e^{-2})$. It can be seen that we can write each of these probabilities as a polynomial in e^{-1}. More generally, for any MDP with differing rates (of the exponential distribution) in each state, we get a set of rational functions in $e^{-\frac{1}{q}}$ for some $q \in \mathbb{Z}_{>0}$, where q is obtained as a function of the rates in each state. Thus, using standard algorithms for MDPs [6], and as done for Markov chains in [8], we get that we can compute expressions for the probability of reaching the targets, and decide the threshold problem.

▶ **Theorem 8.** *Quantitative reachability for 1-clock $1\frac{1}{2}$-player STGs satisfying ($\star$) is decidable.*

We can lift this construction to include $\Box$ player nodes, keeping the same initialized restriction with $\Box$ nodes as well. Then the region game graph $\mathcal{G}_{\mathcal{R}}$ includes $\Box$ nodes in the obvious way, and we consider strategy profiles of $\Box$ and $\Diamond$. The question then is to check if $\Diamond$ has a strategy to reach a target with probability $\sim c$ against all possible strategies of $\Box$ in $M_{\mathcal{G}}$. Hence we have that

▶ **Corollary 9.** *Quantitative reachability for 1-clock $2\frac{1}{2}$ player STGs satisfying ($\star$) is decidable.*

5 Discussion

In this paper, we have refined the decidability boundaries for STGs as summarized in the table in Introduction. The significance of our undecidability results for quantitative reachability (via different two-counter machine reductions) lies in the fact that they introduce ideas which could potentially help in settling other open problems. We highlight these below:

- for $1\frac{1}{2}$ player games, the crux is to cleverly encode the error ϵ made by player $\Diamond$ in such a way that it reflects as $\frac{1}{2} - \epsilon^2$ in the resulting probability. This ensures that the $\Diamond$ player can never cheat and the probability will be $< \frac{1}{2}$ as soon as there is an error (even when simulating a non-halting run of the two-counter machine). Indeed, this is why the reduction is from the non-recursively enumerable non-halting problem.
- for $2\frac{1}{2}$ player games in the *time-bounded setting*, we obtain undecidability by showing a reduction from halting problem for two-counter machines. This is surprising, as time-boundedness restores decidability in several classical undecidable problems like the inclusion problem in timed automata [20, 21]. In the case of priced timed games [14], time-boundedness gives undecidability; however, this can be attributed to the fact that price variables are not clocks, and can grow at different rates in different locations. Somehow, the combination of simple clocks and probabilities achieves the same.

Combining these ideas might allow us, for eg., to improve Theorem 6 by showing undecidability of time bounded, quantitative reachability in $1\frac{1}{2}$ player STGs with a larger number of clocks. The main challenge is to replace $\Box$ player nodes by stochastic nodes, and adapt the gadgets in such a way that, within a global time bound, the probability of reaching a target is $\frac{1}{2}$ iff all simulations are correct and the two-counter machine does not halt. As another example, if in the first item above, we obtain a probability of $1 - \epsilon^2$ (rather than $\frac{1}{2} - \epsilon^2$), this would settle the (currently open) *qualitative* reachability problem for $2\frac{1}{2}$ games [12].

Coming to decidability results, we have for the first time characterized a family of $1\frac{1}{2}$,$2\frac{1}{2}$ player STGs for whom the quantitative reachability is decidable. The use of exponential distributions is mandatory to get a closed form expression for the probability. It is unclear if this construction can be extended to some larger classes of STGs. Figure 9 in [9] shows an example of a two-clock $\frac{1}{2}$ player game for which the region abstraction fails to give any relevant information on the real "probabilistic" behaviour of the system (lack of so-called fairness); in particular it cannot be used for qualitative, and therefore quantitative, analysis of reachability properties. The decidability of qualitative reachability in $1\frac{1}{2}, 2\frac{1}{2}$, multi-clock STG seems then hard due to the same problem of unfair runs.

References

1 S. Akshay, P. Bouyer, S. N. Krishna, L. Manasa, and A. Trivedi. Stochastic timed games revisited. `http://www.cse.iitb.ac.in/~krishnas/TR16.pdf`.

2 R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. of IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.

4 C. Baier, P. Bouyer, T. Brihaye, and M. Größer. Almost-sure model checking of infinite paths in one-clock timed automata. In *Proc. 23rd Annual Symposium on Logic in Computer Science (LICS'08)*, pages 217–226. IEEE Computer Society Press, 2008.

5 C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(7):524–541, 2003.

6 C Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, 2008.

7 N. Bertrand, P. Bouyer, T. Brihaye, and P. Carlier. Analysing decisive stochastic processes. In *Proc. 43rd International Colloquium on Automata, Languages and Programming (ICALP'16) – Part II*, Leibniz International Proceedings in Informatics. Leibniz-Zentrum für Informatik, July 2016. To appear.

8 N. Bertrand, P. Bouyer, T. Brihaye, and N. Markey. Quantitative model-checking of one-clock timed automata under probabilistic semantics. In *Proc. 5th International Conference on Quantitative Evaluation of Systems (QEST'08).* IEEE Computer Society Press, 2008.

9 N. Bertrand, P. Bouyer, T. Brihaye, Q. Menet, M. Größer, and M. Jurdziński. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4):1–73, 2014.

10 N. Bertrand, T. Brihaye, and B. Genest. Deciding the value 1 problem for reachability in 1-clock decision stochastic timed automata. In *Proc. 11th International Conference on Quantitative Evaluation of Systems (QEST'14)*, pages 313–328. IEEE Computer Society Press, 2014.

11 H.C. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. MODEST: A compositional modeling formalism for hard and softly timed systems. *IEEE Transactions on Software Engineering*, 32(10):812–830, 2006.

12 P. Bouyer and V. Forejt. Reachability in stochastic timed games. In *Proc. 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5556 of *LNCS*, pages 103–114. Springer, 2009.

13 Tomáš Brázdil, Jan Krčál, Jan Křetínský, and Vojtěch Řehák. Fixed-delay events in generalized semi-Markov processes revisited. In *Proc. 22nd International Conference on Concurrency Theory (CONCUR'11)*, volume 6901 of *LNCS*, pages 140–155. Springer, 2011.

14 T. Brihaye, G. Geeraerts, S. N. Krishna, L. Manasa, B. Monmege, and A. Trivedi. Adding negative prices to priced timed games. In *Proc. 25th International Conference on Concurrency Theory (CONCUR'14)*, LIPIcs, pages 560–575. Leibniz-Zentrum für Informatik, 2014.

15 J. Filar and K. Vrieze. *Competitive Markov Decision Processes.* Springer, 1997.

16 V. Forejt, M. Kwiatkowska, G. Norman, and A. Trivedi. Expected reachability-time games. In *Proc. 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'10)*, volume 6246 of *LNCS*, pages 122–136. Springer, 2010.

17 M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *Proc. of 11th International Conference on*

Concurrency Theorey, (CONCUR'00), volume 1877 of *LNCS*, pages 123–137. Springer, 2000.

18 M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, June 2002.

19 F. Laroussinie, N. Markey, and P. Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *LNCS*, pages 387–401. Springer, 2004.

20 J. Ouaknine, A. Rabinovich, and J. Worrell. Time-bounded verification. In *Proc. 20th International Conference on Concurrency Theory (CONCUR'09)*, volume 5710 of *LNCS*, pages 496–510. Springer, 2009.

21 J. Ouaknine and J. Worrell. Towards a theory of time-bounded verification. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP'10)*, volume 6199 of *LNCS*, pages 22–37. Springer, 2010.

22 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 1994.

23 Uppaal case-studies. `http://www.it.uu.se/research/group/darts/uppaal/examples.shtml`.

Inequity Aversion Pricing over Social Networks: Approximation Algorithms and Hardness Results[*]

Georgios Amanatidis[1], Evangelos Markakis[2], and Krzysztof Sornat[3]

1 **Athens University of Economics and Business, Athens, Greece**
`gamana@aueb.gr`
2 **Athens University of Economics and Business, Athens, Greece**
`markakis@gmail.com`
3 **University of Wroclaw, Wroclaw, Poland**
`krzysztof.sornat@cs.uni.wroc.pl`

Abstract

We study a revenue maximization problem in the context of social networks. Namely, we consider a model introduced by Alon, Mansour, and Tennenholtz (EC 2013) that captures *inequity aversion*, i.e., prices offered to neighboring vertices should not be significantly different. We first provide approximation algorithms for a natural class of instances, referred to as the class of single-value revenue functions. Our results improve on the current state of the art, especially when the number of distinct prices is small. This applies, for example, to settings where the seller will only consider a fixed number of discount types or special offers. We then resolve one of the open questions posed in Alon et al., by establishing APX-hardness for the problem. Surprisingly, we further show that the problem is NP-complete even when the price differences are allowed to be relatively large. Finally, we also provide some extensions of the model of Alon et al., regarding the allowed set of prices.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases inequity aversion, social networks, revenue maximization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.09

1 Introduction

We study a differential pricing optimization problem in the presence of network effects. Differential pricing is a well known practice in everyday life and refers to offering a different price to potential customers for the same service or good. Examples include offering cheaper prices when launching a new product, making special offers to gold and silver members of an airline miles program, offering discounts at stores during selected periods, and several others.

We are interested in studying differential pricing in the context of a social network. Imagine a network connecting individuals (who are seen as potential clients here) with their friends, family, or colleagues, i.e., with people who can exert some influence on them. One can have in mind other forms of abstract networks as well, e.g., a node could represent a geographic region, a neighborhood within a city, a type of profession, a social class, and edges can represent interactions or proximity. The presence of such a network creates *externality*

[*] This research was supported by National Science Centre, Poland, 2015/17/N/ST6/03684. It was also supported by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALES.

effects, meaning that the decision of a node to acquire a new product or a new service is affected by the fact that some other nodes within her social circle (her neighborhood in the graph) already did so. A typical example of positive externalities is when someone becomes more likely to buy a new product due to the positive reviews by a friend who already bought it in the past. Modeling positive externalities has led to a series of works that study marketing strategies for maximizing the diffusion of a new product, [7, 14], or the total revenue achieved, [13] (see also the Related Work section).

However, there also exist negative externality effects that can arise in a network. One example is the purchase of a product with the intention to show off and be a locally unique owner, e.g., a new type of expensive car, or clothes (also referred to as *invidious consumption*, see [5]). In such a case, a node may be deterred from buying the same product, if a neighboring node already did so. A second example of negative externalities, which is the focus of our work, and arises from differential pricing, is *inequity aversion*, see e.g., [4] and [8]. This simply means that a customer may experience dissatisfaction if she realizes that other people within her social circle, were offered a better deal for the same service. Hence, significant price differences, can create a negative response of some customers towards a product. Inequity aversion can also arise under a different, but equally applicable, interpretation: nodes may correspond to retail stores and an edge can signify proximity, so that clients could choose among these stores. Again, having significantly different prices to the same products is not desirable.

To capture the need for avoiding such phenomena, the relatively recent work of [2] introduced a model for pricing nodes over a social network. The main idea is to impose constraints on each edge, specifying that the price difference between two neighbors should be bounded by some (endogenous) parameter, determined by the two neighbors. On top of this, the seller is also allowed to not make a price offer to some nodes (referred to as introducing *discontinuities*, see the related discussion in Section 2), in which case the difference constraints do not apply for the edges incident to these nodes. Assuming a finite set of available prices, unit-demand users, and digital goods (i.e., the supply can cover all the demand) the problem is to find a feasible price vector that satisfies the edge constraints and maximizes the total revenue. In its more general form the problem was shown to be NP-complete, but exact or approximation algorithms were derived for some interesting cases.

Contribution: We revisit the model introduced by [2] (namely Model II of [2], which is the more general one), and study the approximability of the underlying revenue maximization problem. We resolve one of the open questions posed in [2], regarding the complexity of the problem under the natural class of the so-called *single-value* revenue functions. Simply put, this means that the revenue extracted by each node is exactly the price offered to her, as long as the price does not exceed her valuation for the product (the usual assumption made in auction settings as well). We first establish APX-hardness for this class answering the question of [2], and we also show that the problem is NP-complete even when the price differences are allowed to be relatively large (a case that could be thought easier to handle). We then provide approximation algorithms that improve some of the currently known results. Our improvement is stronger when the number of distinct prices is small. This applies for example to many settings where the seller will only consider a fixed number of discount types or special offers to selected customers. As the number of available price offers becomes large, the performance of our algorithm degrades to a logarithmic approximation. Finally, we provide an extension of these results to a more general model where the allowed prices come from a set of k arbitrary integers, instead of using price sets of the form $\{1, 2, \ldots, k\}$, as done in [2] (see Subsection 4.3).

Related Work: Price discrimination is well studied in various domains in economics and is also being applied to numerous real life scenarios. The algorithmic problem of differential pricing over social networks is a more recent topic, initiated by [13]. The work of [13] studied a model with positive externalities, where the valuation of a player may increase as more friends acquire a good, and analyzed the performance of a very intuitive class of pricing strategies. Further improvements on the performance of such strategies were obtained later on by [9]. The work of [1] also considers a pricing problem but in an iterative fashion, where the seller is allowed to reprice a good in future rounds. Revenue maximization under a mechanism design approach was also taken in [12] under positive network externalities. Finally, positive externalities have been used to model the diffusion of products on a network, see, among others, the exposition in [15].

Negative externalities within networks, as we focus on here, are less studied in the literature. For the concept of inequity aversion, see e.g., [4, 8]. The work most closely related to ours is [2], which introduced the model that we consider here. Efficient algorithms were obtained for the case where discontinuities are not allowed (even for more general revenue functions), and also for networks with bounded treewidth. An approximation ratio of $1/(\Delta + 1)$ was also provided, where Δ is the maximum degree. Similar results were shown for a stochastic version of the model. Finally, other types of negative externalities have been considered e.g., in [3, 5] which study the effects of invidious consumption.

2 Definitions and Preliminaries

The social network is represented as an undirected graph $G = (V, E)$, with $|V| = n$. We assume that a provider of some good or service has a finite set P of available prices that he could offer to the nodes. In most of our presentation, we assume as in [2], that the available prices are given by $P = \{1, 2, \ldots, k\}$. In Subsection 4.3, we show how to extend the analysis when P is an arbitrary set of k positive integers, i.e., $P = \{p_1, p_2, \ldots, p_k\}$.

We assume every node has a unit-demand for the same product and that the supply of the seller is enough to cover the demand of all nodes. For every node $v \in V$, we associate a revenue function $R_v : \{1, 2, \ldots, k\} \mapsto \mathbb{N}$ that maps an offered price p_v to the revenue that the provider gains from this offer. In this paper, we focus on a simple and intuitive class of revenue functions, also studied in [2]. In particular, for a node $v \in V$, R_v is called a *single value revenue function*, if there exists a value $val(v)$ such that when offered a price p_v:

$$R_v(p_v) = \begin{cases} p_v & \text{if } val(v) \geqslant p_v \\ 0 & \text{if } val(v) < p_v \end{cases}$$

We assume from now on that every node has a single value revenue function. We also assume that $val(v) \in P$, for every $v \in V$. This is because for revenue maximization, that we are interested in, nodes with $val(v) > k$, can only yield a revenue of k, and could be replaced by $val(v) = k$, i.e., the highest possible price. Also for values that are less than k, and not integers, we can again extract only an integer revenue, given the form of P. Finally, any node v with $val(v) < 1$ can be deleted without affecting the optimal revenue (see the concept of *discontinuity* defined below), so we can completely ignore such nodes to begin with. Thus, we consider only instances with $val(v) \in \{1, 2, \ldots, k\}, \forall v \in V$.

Given a vector $\mathbf{p} = (p_v)_{v \in V}$ of prices offered to the nodes, the total revenue is $R(\mathbf{p}) = \sum_{v \in V} R_v(p_v)$. Hence, our goal is to find a price vector that maximizes the total revenue. At the same time, however, we want to capture the effect of *inequity aversion* [4, 8] in social networks. This means that a node may experience dissatisfaction if she sees that other nodes within her social circle, were offered a better deal for the same service. Hence, significant price differences, create negative externalities among users.

To avoid such situations the model introduced in [2] has constraints on each edge, stating that the price difference between two neighbors u, v is bounded, i.e., $p_u - p_v \leqslant \alpha(u,v)$ and $p_v - p_u \leqslant \alpha(v,u)$, for every $(u,v) \in E$. Here, $\alpha(\cdot,\cdot) \geqslant 0$ is integer-valued (given that the prices are also integers) and note that in general is non-symmetric. Furthermore, the seller is also allowed not to make an offer to certain nodes. Formally, this is captured by having one more price option, which we denote by $\perp$, with $R_v(\perp) = 0$. Setting $p_v = \perp$ to a node, means that the provider does not make any offer to v, and there is no price restriction on the edges that are incident to v. We can essentially think about this as deleting these vertices from the graph. We will refer to setting $p_v = \perp$ to a node $v \in V$, as introducing a *discontinuity* on v. Avoiding making an offer can be thought of as choosing not to promote a product or service within a certain region or within a certain social group. In terms of optimization, allowing discontinuities can help the seller in producing much higher revenue (than without discontinuities) as Proposition 3 in Section 3 states.

Given this model, the set of feasible price vectors is then: $\mathcal{F} = \{\mathbf{p} : \forall\ v \in V, p_v \in P \cup \{\perp\}$, and $\forall\ (u,v) \in E,\ p_u \neq \perp\ \wedge\ p_v \neq \perp \Rightarrow p_u - p_v \leqslant \alpha(u,v)\ \wedge\ p_v - p_u \leqslant \alpha(v,u)\}$. Therefore, the problem we study is:

Inequity Aversion Pricing: Given a graph with edge constraints, and a single-value revenue function for each node, find a feasible price vector that maximizes the total revenue, i.e., find $\mathbf{p} \in \mathcal{F}$ that achieves $\max_{\mathbf{p} \in \mathcal{F}} \sum_{v \in V} R_v(p_v)$.

Some cases of this problem, as well as the variant where no discontinuities are allowed, are already known to be polynomial time solvable [2]. Regarding hardness, although the problem is NP-hard for more general revenue functions, it was posed as an open question whether NP-hardness still holds for single value revenue functions (the hardness result in [2] requires instances with revenue functions that cannot be captured by single value ones).

3 Warm-up: Basic Facts and Single-price Solutions

In this section, we present a simple algorithm and some basic observations, which we use later on, in Section 4.

Let $v_{\max} = \max_{v \in V} val(v) \leqslant k$, and $\text{MAX} = \sum_{v \in V} val(v)$. Given an instance of the problem, we denote by OPT the revenue of an optimal solution. The quantity MAX is clearly an upper bound on the optimal revenue, hence $\text{OPT} \leqslant \text{MAX}$.

We will refer to a solution as being a *single-price* solution, if it charges the same price to every node without introducing discontinuities. Note that this is always a feasible solution since all the edge constraints are satisfied. The revenue extracted by a single-price algorithm that uses the price of p for all nodes is equal to $p \cdot |\{v \in V : val(v) \geqslant p\}|$.

To understand whether single-price solution can be of any help for our setting, we can examine the performance of the best possible single price. The following observation suggests that we do not need to try too many values, even if v_{max} is very large.

▶ **Lemma 1.** *In order to find the optimal single-price solution, it suffices to check at most* $\min\{n, v_{\max}\}$ *possible prices.*

Proof. There are at most $min\{n, v_{\max}\}$ different values in the set $\{val(v) : v \in V\}$. It is never optimal to use any price $p \notin \{val(v) : v \in V\}$. Indeed, if $p \in (val(v_1), val(v_2))$, where $val(v_1)$ and $val(v_2)$ are two consecutive distinct values for some nodes $v_1, v_2 \in V$, then it is strictly better to set the price to $val(v_2)$. For the same reason, it is suboptimal to set a price that is less than the minimum value across nodes, while if we use a price $p > v_{\max}$ then we gain no revenue. ◀

Hence in $O(\min\{n, v_{\max}\})$ steps, we can select the best single-price solution. Let us denote by R_{SP} the revenue raised by this solution. The performance of R_{SP} has been analyzed in a different context[1] by [11], where it was shown that it achieves a $\Theta(\ln n)$-approximation. Here we give a slightly tighter statement, which we utilize in later sections for small values of v_{max}.

▶ **Theorem 2.** *For any number n of agents, the optimal single-price solution achieves a $1/H_r$-approximation, where $r = \min\{n, v_{\max}\}$, and H_ℓ is the ℓ-th harmonic number, i.e.,*

$$R_{\text{SP}} \geqslant \frac{\text{MAX}}{H_r} \geqslant \frac{\text{OPT}}{H_r}\,.$$

Furthermore, the approximation guarantee is tight.

The proof is deferred to the full version of the paper. One interesting point here, is that single-price solutions do not use any discontinuities. If R_{ND} is the maximum revenue without using any discontinuities, clearly $R_{\text{ND}} \geqslant R_{\text{SP}}$. And as we mentioned in Section 2, it is possible to find the optimal solution that does not use discontinuities in polynomial time; so why use something worse instead of R_{ND}? Actually, besides being harder to argue about, R_{ND} turns out to be as bad an approximation as R_{SP}, in the worst case. Hence, the proposition below reveals that introducing discontinuities can cause a significant increase in the optimal revenue achievable by the seller, compared to what can be achieved without discontinuities.

▶ **Proposition 3.** *The optimal solution with no discontinuities achieves a $1/H_r$-approximation, where $r = \min\{n, v_{\max}\}$, and this approximation guarantee is tight.*

The proof of Proposition 3 is deferred to the full version of the paper.

4 Approximation of Inequity Aversion Pricing

In this section we present new approximation algorithms for the problem by exploiting ways in which setting discontinuities in certain nodes can help. Our main result is an approximation algorithm, with a ratio of $(H_k - 0.25)^{-1}$. Even though asymptotically this is no better than the optimal single-price algorithm, it does yield better ratios for instances where k is a small constant. The motivation for studying cases where the set of available prices is a small constant is that a seller may be willing to offer only specific types of discount to selected customers, e.g., 20% or 30% off the regular price and so on, rather than using an arbitrary set of prices.

We start below with the case of $k = 2$, before we move to the more general case.

4.1 A 0.8-approximation Algorithm when $P = \{1, 2\}$ via Vertex Cover

In this subsection, we assume the available prices are 1, 2, or $\perp$. Despite this restriction, the problem still remains non-trivial, and it is currently not known if it is NP-complete

[1] The work of [11] studied an auction pricing problem without the presence of social networks.

Algorithm 1: A 0.8-approximation when $P = \{1, 2\}$

```
1 Given the graph $G = (V, E)$, construct the bipartite graph $G' = (V_1, V_2, E')$ with
  $V_i = \{v \in V : val(v) = i\}$ and $E' = \{(u, v) \in E : val(u) = 2, val(v) = 1, \alpha(u, v) = 0\}$
2 Find a minimum vertex cover on $G'$, say $S \subseteq V$
3 Set $\perp$ to all vertices of $S$
4 Set a price of 1 to every $v \in V_1 \setminus S$ and a price of 2 to every $v \in V_2 \setminus S$. Let $R^*$ be the revenue
  obtained by this solution
5 Compute the optimal single-price solution, as described in Section 3, with revenue $R_{\mathrm{SP}}$
6 Return the solution that achieves $\max\{R^*, R_{\mathrm{SP}}\}$
```

or not. Given the discussion in Section 2, we will also assume that for every node $v \in V$, $val(v) \in \{1, 2\}$. For such instances we already have a $\frac{2}{3}$-approximation by Theorem 2, that does not use discontinuities. The difficulty in improving this factor is in finding a way of selecting appropriate nodes to set to $\perp$.

Before we describe our algorithm, let us illustrate the main idea. Consider an instance of the problem on a graph $G = (V, E)$. Suppose we plan to find a feasible price vector, such that for each u, either $p_u = \perp$ or $p_u = val(u)$. Since the possible prices are only 1 and 2, if $val(u) = 1$, then for any $(u, v) \in E$, $\alpha(u, v)$ is not restrictive, while if $val(u) = 2$, then for any $(u, v) \in E$, $\alpha(u, v)$ is restrictive only if $\alpha(u, v) = 0$ and $val(v) = 1$. So, we could remove any edge except from edges in $E' = \{(u, v) \in E : val(u) = 2, val(v) = 1, \alpha(u, v) = 0\}$. Note that this defines a bipartite subgraph $G' = (V_1, V_2, E')$ of G, where $V_i = \{v \in V : val(v) = i\}$. Since this new instance has less restrictions, the optimal revenue OPT' is at least as good as the optimal revenue OPT of the original instance.

Consider a vertex cover S in G'. The crucial observation is that we can satisfy all the edge constraints regarding edges between V_1 and V_2, by introducing discontinuities on the vertices of S. Since S covers all the edges between V_1 and V_2, the edge constraints between V_1 and V_2 in the original graph G are now non-existent. If we also set a price of 1 on the remaining vertices of V_1 and a price of 2 on the remaining vertices of V_2, all the original constraints are satisfied. Thus, we have constructed a feasible solution for G.

The revenue of such a solution is $\mathrm{MAX} - val(S)$, where $\mathrm{MAX} = \sum_{v \in V} val(v) = |V_1| + 2 \cdot |V_2|$ and $val(S) = \sum_{v \in S} val(v)$. Hence, the best outcome of such an algorithm is achieved when S is a minimum weighted vertex cover (using the values as weights) rather than just any vertex cover. For the analysis however, it suffices to compute just a minimum vertex cover (see the Remark after the proof of Theorem 4). Moreover, by the Kőnig–Egerváry Theorem, we can compute this in polynomial time for bipartite graphs (e.g., see [16]).

Finally, the algorithm compares the best of two outcomes, the solution outlined above and the solution discussed in Section 3. Hence, we define $\mathrm{ALG} = \max\{R_{\mathrm{SP}}, \mathrm{MAX} - val(S)\}$, where R_{SP} is the maximum revenue achieved by setting a fixed price to every node.

▶ **Theorem 4.** *Algorithm 1 achieves a* 0.8*-approximation for the Inequity Aversion Pricing problem when* $P = \{1, 2\}$*. Furthermore, this ratio is tight.*

Proof. Let ALG denote the revenue obtained by Algorithm 1 and let β be its approximation ratio that we attempt to determine. Assume that $\beta < 0.8$. Then there exists some $\epsilon > 0$ such that $\beta = 0.8 - \epsilon$. To arrive at a contradiction, we are going to show that $\beta \geqslant \gamma = 0.8 - \epsilon/2$.

We will distinguish some cases, depending on the value of ALG. First of all, note that if $\mathrm{ALG} \geqslant \gamma \cdot \mathrm{MAX}$, then we trivially obtain a γ-approximation: $\frac{\mathrm{ALG}}{\mathrm{OPT}} \geqslant \frac{\gamma \cdot \mathrm{MAX}}{\mathrm{MAX}} \geqslant \gamma$. From now on, assume that $\mathrm{ALG} < \gamma \cdot \mathrm{MAX}$. The following turns out to be a very useful upper bound for OPT.

▶ **Claim 5.** *Let S denote a minimum vertex cover in the graph G' (defined in step 1 of Algorithm 1). Then,* $\mathrm{OPT} \leqslant \mathrm{OPT}' \leqslant \mathrm{MAX} - |S|$.

Proof of Claim 5. The first inequality is straightforward (see also the discussion before the theorem). For the second inequality, note that by the Kőnig–Egerváry Theorem, the maximum matching in G' has the same cardinality as S. Let M be such a maximum matching. By the definition of G', for each edge $(u, v) \in M$ the nodes u and v have different values, say $val(u) = 2$ and $val(v) = 1$. Because of (u, v), an optimal solution must lose at least one unit of revenue in comparison with MAX. Indeed, since $\alpha(u, v) = 0$, an optimal solution either sets a discontinuity on one of these two nodes, or it sets the same price. If this common price is 1, we lose one unit from node v, whereas if it is 2 we do not extract revenue from u. The claim follows. ◁

We know that $\mathrm{ALG} = \mathrm{MAX} - val(S)$ and also $val(S) \leqslant 2|S|$. Thus, $|S| \geqslant \frac{1}{2}(\mathrm{MAX} - \mathrm{ALG})$. If we combine this with Claim 5, we have

$$\mathrm{OPT} \leqslant \frac{1}{2}(\mathrm{MAX} + \mathrm{ALG})\,. \tag{1}$$

To proceed with the analysis, we divide the interval $[0, \gamma \cdot \mathrm{MAX}]$ into smaller subintervals of the form $\left[\frac{i-1}{m} \cdot \gamma \cdot \mathrm{MAX}\,,\ \frac{i}{m} \cdot \gamma \cdot \mathrm{MAX}\right)$ for some fixed large m and $i \in \{1, \cdots, m\}$. Notice that m is just a parameter in the analysis and has nothing to do with the input. We consider cases depending on where exactly the value of ALG falls. In particular, let i^* be the following interval index: $i^* = \left\lceil \frac{m+2}{2-\gamma} \right\rceil$.

Case (i): $\mathrm{ALG} \in \left[\frac{i-1}{m} \cdot \gamma \cdot \mathrm{MAX}, \frac{i}{m} \cdot \gamma \cdot \mathrm{MAX}\right)$ with $i \geqslant i^*$.
Using inequality (1), we have:

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} \geqslant \frac{\frac{i-1}{m} \cdot \gamma \cdot \mathrm{MAX}}{\frac{1}{2}(\mathrm{MAX} + \mathrm{ALG})} \geqslant \frac{\frac{i-1}{m} \cdot \gamma \cdot \mathrm{MAX}}{\frac{1}{2}(\mathrm{MAX} + \frac{i}{m} \cdot \gamma \cdot \mathrm{MAX})} = \frac{\frac{i-1}{m} \cdot \gamma}{\frac{1}{2}(1 + \frac{i}{m} \cdot \gamma)}\,.$$

In order to ensure a γ-approximation, it suffices to have

$$\frac{\frac{i-1}{m} \cdot \gamma}{\frac{1}{2}(1 + \frac{i}{m} \cdot \gamma)} \geqslant \gamma \iff \frac{i-1}{m} \geqslant \frac{1}{2}\left(1 + \frac{i}{m} \cdot \gamma\right) \iff i \geqslant \frac{m+2}{2-\gamma}\,.$$

But this last inequality holds since $i \geqslant i^*$. Therefore, in this case, the algorithm achieves a γ-approximation.

Case (ii): $\mathrm{ALG} < \frac{i^*-1}{m} \cdot \gamma \cdot \mathrm{MAX}$.
Again, we use inequality (1), but now the lower bound of ALG comes from Theorem 2, which gives a guarantee for the optimal single-price solution:

$$\frac{\mathrm{ALG}}{\mathrm{OPT}} \geqslant \frac{R_{\mathrm{SP}}}{\frac{1}{2}(\mathrm{MAX} + \mathrm{ALG})} \geqslant \frac{\frac{1}{H_2}\mathrm{MAX}}{\frac{1}{2}\mathrm{MAX}\left(1 + \gamma \cdot \frac{i^*-1}{m}\right)} = \frac{4/3}{1 + \gamma \cdot \frac{i^*-1}{m}}\,.$$

Like in case (i), it suffices to have

$$\frac{4/3}{1 + \gamma \cdot \frac{i^*-1}{m}} \geqslant \gamma \iff 4 \geqslant 3\gamma\left(1 + \gamma \cdot \frac{i^*-1}{m}\right).$$

Using an obvious upper bound for i^*, it suffices for γ to satisfy the following:

$$4 \geqslant 3\gamma + 3\gamma^2 \cdot \frac{\frac{m+2}{2-\gamma} + 1 - 1}{m} \iff \frac{6}{m}\gamma^2 + 10\gamma - 8 \leqslant 0\,.$$

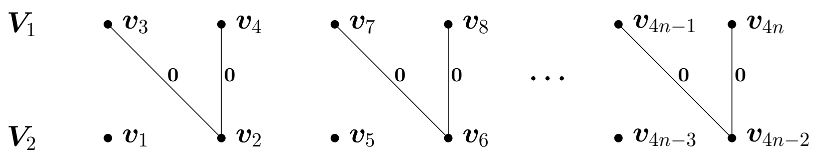


Figure 1 Algorithm 1 is tight on such instances. Only the relevant edges are shown.

Clearly, there is some $m^* \in \mathbb{N}$, such that

$$\frac{6}{m^*}(0.8 - \epsilon/2)^2 + 10(0.8 - \epsilon/2) - 8 \leqslant 0\,.$$

Thus, the approximation ratio β of Algorithm 1 is at least $0.8 - \epsilon/2$, which contradicts the choice of ϵ. Hence, $\beta \geqslant 0.8$.

To see why the ratio of the algorithm is tight, we can construct an infinite family of examples as follows: Consider a graph of 4 nodes $\{v_1, v_2, v_3, v_4\}$ such that $val(v_1) = val(v_2) = 2$, and $val(v_3) = val(v_4) = 1$. There are only two edges, namely (v_2, v_3) and (v_2, v_4). Suppose $\alpha(\cdot,\cdot) = 0$. The optimal revenue here is 5 by offering a price of 1 to v_2, v_3, v_4 and a price of 2 to v_1. On the other hand, the optimal single-price algorithm achieves a revenue of 4, either with a price of 1 or 2. Also, a minimum (weighted or not) vertex cover here is either $\{v_2\}$ or $\{v_3, v_4\}$. In both cases, the revenue by setting $\perp$ to the vertex cover is 4. We can add many copies of this construction (and possibly some extra edges with $\alpha(e) \geqslant 1$ for a connected example) to turn this into an infinite family of tight examples. For an illustration, see Figure 1. ◀

▶ Remark. It seems appealing to try to exploit the fact that we can solve the minimum weighted vertex cover problem in polynomial time for bipartite graphs. However, as our analysis shows, using the weighted version of vertex cover, instead of the unweighted one, does not yield any better approximation.

4.2 An Approximation Algorithm for $k > 2$

We now consider the case where there are more than two available prices. In order to improve the approximation guarantee of Theorem 2, we reduce the problem to the case of $k = 2$, and use the results of the previous subsection.

Consider an instance of the problem, with available prices in $\{\perp, 1, 2, \ldots, k\}$. As discussed in Section 2, we may assume that $val(v) \in \{1, 2, \ldots, k\}$ for every $v \in V$. We create another instance, where we set the value of every node with $val(v) > 1$ to be equal to 2. We can then run Algorithm 1 from Subsection 4.1 on this new instance. At the same time, we can also compute the optimal single-price solution for the original instance, and pick the best among these two solutions. This yields Algorithm 2, described below.

Clearly, Algorithm 2 runs in polynomial time. Note that the solution returned by the algorithm is feasible. Any single-price solution is always feasible, while Algorithm 1 will produce a price vector that is feasible for I', and therefore for I, since the edge restrictions in the two instances are the same. Even though asymptotically, this is still a logarithmic approximation, the algorithm achieves significantly better results for small values of k.

▶ **Theorem 6.** *Algorithm 2 achieves a $\frac{1}{H_{v_{\max}} - 0.25}$-approximation ratio for Inequity Aversion Pricing when the available prices are $\{\perp, 1, 2, \cdots, k\}$, with $k \geqslant 2$.*

```
Algorithm 2: An algorithm for $k > 2$
1 Given an instance $I$, construct a new instance $I'$, where for every $v \in V$,
  $val'(v) = \min\{val(v), 2\}$; everything else remains unchanged
2 Run Algorithm 1 from Subsection 4.1 on instance $I'$, and let $R_*$ be the revenue obtained
3 Compute the optimal single-price solution without discontinuities, on the original instance $I$,
  as described in Section 3, with revenue $R_{\mathrm{SP}}$
4 Return the solution that achieves $\max\{R_*, R_{\mathrm{SP}}\}$
```

Proof. The proof is by induction on $v_{\max}$. For $v_{\max} = 2$ the result follows from Theorem 4 since $0.8 = \frac{1}{H_2 - 0.25}$.

Now assume we have an instance I where $v_{\max} = j > 2$. As usual, let OPT denote the optimal revenue for I and ALG the revenue returned by Algorithm 2. Also, let R_j be the revenue extracted by setting price j at every node, and $V_j = \{v \in V : val(v) = j\}$. We consider two cases.

Case (i): $|V_j| \geqslant \frac{1}{(H_j - 0.25)j} \cdot \mathrm{OPT}$. Then, $\frac{\mathrm{ALG}}{\mathrm{OPT}} \geqslant \frac{R_j}{\mathrm{OPT}} = \frac{j \cdot |V_j|}{\mathrm{OPT}} \geqslant \frac{\frac{1}{H_j - 0.25} \cdot \mathrm{OPT}}{\mathrm{OPT}} = \frac{1}{H_j - 0.25}$.

Case (ii): $|V_j| < \frac{1}{(H_j - 0.25)j} \cdot \mathrm{OPT}$. Let I^* be an instance derived from I by setting $val^*(v) = \min\{val(v), j-1\}$, i.e., we only reduce the valuation of the nodes with $val(v) = v_{\max}$ by 1. Let OPT* denote the optimal revenue for I^*, and ALG* the revenue returned by Algorithm 2. By the inductive hypothesis we have $\mathrm{ALG}^* \geqslant \frac{1}{H_{j-1} - 0.25} \cdot \mathrm{OPT}^*$.

Furthermore, notice that the set of vertices with valuation greater than 1 is the same in both instances. So, Algorithm 2 on input I^* considers exactly the same price vectors as it does on input I, with the exception of the single-price solution that universally uses j. We conclude that $\mathrm{ALG}^* \leqslant \mathrm{ALG}$. Next, we prove the following useful claim.

▶ **Claim 7.** $\mathrm{OPT}^* \geqslant \mathrm{OPT} - |V_j|$.

Proof of Claim 7. Let $\mathbf{p}$ be an optimal price vector for I. Construct the price vector $\mathbf{p}^*$ by decreasing any price that is at least j to $j - 1$. It is straightforward to see that in instance I we have $R(\mathbf{p}^*) \geqslant R(\mathbf{p}) - |V_j| = \mathrm{OPT} - |V_j|$, while in both instances $R(\mathbf{p}^*)$ is the same. What is left to show is that $\mathbf{p}^*$ is feasible for I^*. Observe, however, that the two instances have exactly the same edge restrictions and that, by its definition, $\mathbf{p}^*$ did not increase the price difference between any two vertices compared to $\mathbf{p}$. Thus, $\mathrm{OPT}^* \geqslant R(\mathbf{p}^*) \geqslant \mathrm{OPT} - |V_j|$. ◁

Now, we can write

$$\begin{aligned}\frac{\mathrm{ALG}}{\mathrm{OPT}} &\geqslant \frac{\mathrm{ALG}^*}{\mathrm{OPT}} \geqslant \frac{\frac{1}{H_{j-1} - 0.25} \cdot \mathrm{OPT}^*}{\mathrm{OPT}} \geqslant \frac{\frac{1}{H_{j-1} - 0.25} \cdot (\mathrm{OPT} - |V_j|)}{\mathrm{OPT}} \\ &\geqslant \frac{1}{H_{j-1} - 0.25}\left(1 - \frac{\frac{1}{j(H_j - 0.25)} \cdot \mathrm{OPT}}{\mathrm{OPT}}\right) = \frac{1}{H_{j-1} - 0.25} \cdot \frac{jH_j - 0.25j - 1}{j(H_j - 0.25)} \\ &= \frac{1}{H_{j-1} - 0.25} \cdot \frac{j(H_{j-1} - 0.25)}{j(H_j - 0.25)} = \frac{1}{H_j - 0.25},\end{aligned}$$

which concludes the proof. ◀

4.3 Approximation Algorithms for General Price Sets

We end Section 4 by extending our results when P is an arbitrary set of k positive integers, i.e., $P = \{p_1, p_2, \ldots, p_k\}$. This can be seen as a more realistic model, especially for small values of k. In such a case, one could try to directly apply Theorems 2, 4, or 6 for $P' = \{1, 2, 3, \ldots, p_k\}$.

Table 1 Examples of obtained approximation ratios.

P	$\{1, 2\}$	$\{1, 2, 3\}$	$\{1, \ldots, 100\}$	$\{10, 20, 25\}$	$\{3, 6, 10, 11\}$
$1/H_k$	0.667	0.545	0.193	0.545	0.48
Alg. 2, general α	0.8	0.631	0.202	–	–
Thm. 10, general α || $\alpha = 0$	0.8	0.631	0.202	0.597 || 0.689	0.524 || 0.574

However, this may produce a very poor approximation when k is small but p_k is large, and feasibility is not guaranteed either. In what follows, P_j denotes $\sum_{i=1}^{j} \frac{p_i - p_{i-1}}{p_i}$, where $p_0 = 0$.

We begin with a generalization of Theorem 2.

▶ **Theorem 8.** *For any number n of agents and possible prices $p_1 < p_2 < \ldots < p_k$ the optimal single-price algorithm achieves a ρ-approximation, where $\rho = 1/\min\{H_n, P_k\}$, i.e.,*

$$R_{\text{SP}} \geqslant \frac{\text{MAX}}{\min\{H_n, P_k\}} \geqslant \frac{\text{OPT}}{\min\{H_n, P_k\}},$$

and this approximation guarantee is tight.

For $k = 2$, Theorem 8 yields an approximation ratio of $\frac{p_2}{2p_2 - p_1}$. We can still use the ideas of Theorem 4 to improve this factor. Notice, however, that although all of our results so far are independent of $\alpha(\cdot, \cdot)$, now the improvement will depend on the edge constraints. As in Algorithm 1, we can define a bipartite graph by using a restricted subset of the edges of G. In analogy to the set E' in section 4.1, we let $E' = \{(u, v) \in E : val(u) = p_2, val(v) = p_1,$ and $\alpha(u, v) < p_2 - p_1\}$, and $\alpha = \max_{(v_1, v_2) \in E'} \alpha(v_2, v_1)$. We have the following.

▶ **Theorem 9.** *When $P = \{p_1, p_2\}$ there is a polynomial time ρ-approximation algorithm for the Inequity Aversion Pricing problem, where $\rho = \frac{p_2^2}{2p_2^2 - p_1 p_2 - (p_2 - p_1)\min(p_1, p_2 - p_1 - \alpha)}$. Furthermore, this ratio is tight.*

Notice that Theorem 9 yields a 0.8-approximation when $P = \{1, 2\}$. Finally, based on the improved approximation for two prices, we can get an analog of Theorem 6 for any number of distinct prices. Given an instance I, let I' be the new instance where for every $v \in V$, $val'(v) = \min\{val(v), p_2\}$, while the constraints remain the same.

▶ **Theorem 10.** *Let $P = \{p_1, p_2, \cdots, p_k\}$, and suppose that on instance I' (described above) the algorithm implied by Theorem 9 gives a $\frac{1}{P_2 - x}$-approximate solution. Then, we can get a $\frac{1}{P_k - x}$-approximate solution for the original instance of the Inequity Aversion Pricing problem in polynomial time.*

The proofs of all results in this subsection are deferred to the full version of the paper. We note however, that the algorithms and the proofs for Theorems 9 and 10 are similar to the corresponding algorithms and proofs for Theorems 4 and 6 respectively.

Table 1 summarizes approximation ratios obtained by the optimal single price solution, Algorithm 2, as well as the algorithm implied by Theorem 10 for different sets of prices.

5 Hardness for Single Value Revenue Functions

In [2] there is an $n^{1-\epsilon}$ inapproximability result for Inequity Aversion Pricing, but for general revenue functions and $\alpha(u, v) = 1$ for every edge. An NP-hardness proof is also given for these edge constraints when single value and constant revenue functions are allowed. The NP-hardness of Inequity Aversion Pricing as we study it here, i.e., allowing only single value revenue functions, was left as an open question. We resolve this question by proving that the problem remains NP-complete even if we restrict the revenue functions to be single value. Our reduction implies that the result holds even when the price differences are allowed to be close to the maximum possible price k. Further, when $\alpha(u,v)=0$ for every edge, we are able to show APX-hardness.

The reduction, below, is from the decision version of *3-Terminal Node Cut*: Given a graph $G(V,E)$, a set $S=\{v_1,v_2,v_3\}\subseteq V$, and an integer q, is there a subset of q vertices that can be deleted, so that any two vertices of S are in different connected components of the resulting graph? The NP-completeness of the weighted version of 3-Terminal Node Cut is discussed in [6], while the APX-hardness of the unweighted version we use here is discussed in [10]; in either case no explicit proof is given. The NP-completeness result we need follows from Theorem 15 as well.

▶ **Theorem 11.** *Let $\epsilon>0$ be any small constant. The decision version of Inequity Aversion Pricing (for single value revenue functions) is NP-complete even when $\alpha(u,v)$ is as large as $k^{1-\epsilon}$ for all $(u,v)\in E(G)$, where k is the maximum possible price.*

Proof. It is immediate that the problem is in NP. To facilitate the presentation, we prove the NP-hardness when $\alpha(\cdot,\cdot)$ is upper bounded by $k^{1/3}/3$. As discussed at the end of the proof, the reduction can be easily adjusted when the upper bound of $\alpha(\cdot,\cdot)$ is $k^{1-\epsilon}$, for constant ϵ.

Let us consider an instance of 3-Terminal Node Cut, i.e., a graph $G(V,E)$, with $|V(G)|=n$, a set $S=\{v_1,v_2,v_3\}$ of non adjacent vertices of G, and an integer q. We may assume that $q\leqslant n-3$, otherwise the question is trivial. Next we give a construction of an appropriate instance for Inequity Aversion Pricing.

Let H be the graph obtained from G as follows. We replace every vertex $v\in S$ by n^3 vertices, where each such vertex has the same neighbors as v, i.e., if u_v is a vertex in the bundle of vertices replacing v, then for every edge $(v,x)\in E(G)$ we add the edge (u_v,x) to $E(H)$. For any $v\in S$, we call such a set of vertices in H a v-bundle. The set of prices is $\{\perp,1,2,\dots,k\}$, where $k=n^3+n^2$. Finally, for any $(u,v)\in E(H)$ we set $\alpha(u,v)$ and $\alpha(v,u)$ arbitrarily, as long as they are at most $k^{1/3}/3$. Note that $|V(H)|=n-3+3n^3$, and $|E(H)|\leqslant|E(G)|+3(n-1)n^3\leqslant 3n^4$.

Next we define the single value revenue functions for the vertices of H. For every $v\in V(G)\setminus S$, let $val(v)=n^3+n^2$, and for every $v_i\in S$, let $val(u_{v_i})=n^3+\frac{i-1}{2}n^2$ for all u_{v_i} in the v_i-bundle. We show below that G has a subset of at most q vertices that separate all the vertices of S, if and only if there is a feasible choice of prices for the vertices of H that gives revenue at least R_q, where $R_q=(n-3-q)\,n^3+\sum_{i=1}^{3}n^3\left(n^3+\frac{i-1}{2}n^2\right)$.

One direction is easy. Let A be a subset of at most q vertices of G that separate the three vertices of S. For all $v\in A$ we put a discontinuity on the corresponding v in H. If we think of these vertices as removed from H, this creates several connected components. For any other vertex $u\in V(H)$, if u is in the same component as some v_i-bundle (or itself is one of the vertices of the v_i-bundle), set its price to $n^3+\frac{i-1}{2}n^2$, otherwise set its price to n^3+n^2. Notice that any vertex without a discontinuity produces revenue at least n^3, while any vertex u_{v_i} in a v_i-bundle with $v_i\in S$ produces revenue exactly $n^3+\frac{i-1}{2}n^2$. Now, it is straightforward to check that this price vector $\mathbf{p}$ is feasible and gives enough revenue: $R(\mathbf{p})=\sum_{u\in V(H)}R(u)\geqslant(n-3-q)\,n^3+\sum_{i=1}^{3}n^3\left(n^3+\frac{i-1}{2}n^2\right)=R_q$.

For the opposite direction we begin with a couple of observations. Assume that there is a price vector $\mathbf{p}_*$ that gives revenue at least R_q. We claim that $\mathbf{p}_*$ can have only a few discontinuities.

▶ **Claim 12.** *There is no feasible price vector* $\mathbf{p}$ *with* $R(\mathbf{p}) \geqslant R_q$ *and more than* q *discontinuities.*

One immediate implication of Claim 12 is that for any $v \in S$ not every vertex in the v-bundle has price $\perp$. This holds because the v-bundle has n^3 vertices and only $q \leqslant n-3$ of them can get $\perp$. This is crucial, because if we think of the vertices with price $\perp$ as removed from H, then no two vertices are separated because of discontinuities in the v-bundles. In particular, we can completely ignore those discontinuities with respect to connectivity.

Let $D_{\mathbf{p}} = \{v \in V(G) \setminus S \mid \mathbf{p}_v = \perp\}$, i.e., $D_{\mathbf{p}}$ is the set of non terminal vertices in G that their corresponding vertices in H have discontinuities in $\mathbf{p}$. So far, by Claim 12, we have that $|D_{\mathbf{p}_*}| \leqslant q$. What is left to be shown is that these discontinuities separate the v-bundles, for any $v \in S$.

▶ **Claim 13.** *There is no feasible price vector* $\mathbf{p}$ *such that* $R(\mathbf{p}) \geqslant R_q$*, and for some* $v_i, v_j \in S$ *vertices from both the* v_i*-bundle and the* v_j*-bundle are in the same connected component of the graph* $H' = H - \{v \in V(H) \mid v \text{ is not in a bundle and } \mathbf{p}_v = \perp\}$*.*

We conclude that $D_{\mathbf{p}_*}$ is a set of at most q vertices of G that separate all the vertices of S. This completes the proof for the case where $\alpha(\cdot,\cdot)$ is upper bounded by $k^{1/3}/3$.

▶ **Claim 14.** *The above reduction generalizes for* $\alpha(\cdot,\cdot)$ *upper bounded by* $k^{1-\epsilon}$ *for any possitive constant* ϵ*.*

The proofs of Claims 12, 13, and 14 are deferred to the full version of the paper. ◀

For the special case where all the differences are 0, we show that the problem is APX-hard. In doing so, we prove that 3-Terminal Node Cut is MAX SNP-hard, and thus APX-hard. As noted already, MAX SNP-hardness of 3-Terminal Node Cut is discussed —but not explicitly proved— in [10]. Here, having this reduction is crucial, and we have therefore obtained an explicit construction, since eventually we need to show that 3-Terminal Node Cut restricted in a specific set of instances is MAX SNP-hard.

▶ **Theorem 15.** *Multi-Terminal Node Cut is MAX SNP-hard even for* 3 *terminals and all the weights equal to* 1*.*

▶ **Theorem 16.** *Inequity Aversion Pricing (for single value revenue functions) is APX-hard when* $\alpha(e) = 0$ *for all* $e \in E(G)$*.*

The proofs of Theorems 15 and 16 are deferred to the full version of the paper.

▶ Remark. The maximum price k in the instance constructed in the proof of Theorem 16 does not depend on the size of the problem. Given that there is some constant ρ beyond which it is hard to approximate 3-Terminal Node Cut, this means that there exists some constant k^* for which Inequity Aversion Pricing does not have a PTAS. Note that for such a k^* we do have a constant factor approximation, with factor $H_{k^*}^{-1}$.

6 Concluding remarks

We studied a revenue maximization problem under inequity aversion for the natural class of single-value revenue functions. Apart from establishing the first hardness results for this class, we also derived approximation algorithms based on combinatorial and graph-theoretic tools, which improve the state of the art when the set of available prices is small. We find this to be a realistic setting as special price offers are usually derived by specific discount and promotion policies. Several questions still remain open. Even for $k = 2$, it is not known if the problem is NP-hard, or whether we can have better approximation ratios. Clearly, it would also be interesting to resolve the approximability for general k, i.e., can we have a better than $O(1/H_k)$-approximation for large k? Exploring further models of negative externalities is another attractive direction that is not as well studied as the case of positive externalities.

References

1 H. Akhlaghpour, M. Ghodsi, N. Haghpanah, H. Mahini, V. S. Mirrokni, and A. Nikzad. Optimal iterative pricing over social networks (extended abstract). In *Proceedings of the 6th Workshop on Internet and Network Economics, WINE 2010*, pages 415–423, 2010.

2 N. Alon, Y. Mansour, and M. Tennenholtz. Differential pricing with inequity aversion in social networks. In *ACM Conference on Economics and Computation, EC 2013*, pages 9–24, 2013.

3 S. Bhattacharya, J. Kulkarni, K. Munagala, and X. Xu. On allocations with negative externalities. In *Proceedings of the 7th Workshop on Internet and Network Economics, WINE 2011*, pages 25–36, 2011.

4 G. E. Bolton and A. Ockenfels. A theory of equity, reciprocity and competition. *American Economic Review*, 100:166–193, 2000.

5 Z. Cao, X. Chen, X. Hu, and C. Wang. Pricing in social networks with negative externalities. In *Proceedings of the 4th International Conference on Computational Social Networks, CSoNet 2015*, pages 14–25, 2015.

6 W. H. Cunningham. The optimal multiterminal cut problem. In *Reliability Of Computer And Communication Networks, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, December 2-4, 1989*, pages 105–120, 1989.

7 P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the 7th ACM International Conference on Knowledge Discovery and Data Mining, KDD 2001*, pages 57–66, 2001.

8 E. Fehr and K. M. Schmidt. A theory of fairness, competition and co-operation. *Quarterly Journal of Economics*, 114:817–868, 1999.

9 D. Fotakis and P. Siminelakis. On the efficiency of influence-and-exploit strategies for revenue maximization under positive externalities. In *Proceedings of the 8th Workshop on Internet and Network Economics, WINE 2012*, pages 270–283, 2012.

10 N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in node weighted graphs. *J. Algorithms*, 50(1):49–61, 2004.

11 A. Goldberg, J. Hartline, A. Karlin, M. Saks, and A. Wright. Competitive auctions. *Games and Economic Behavior*, 55(2):242–269, 2006.

12 N. Haghpanah, N. Immorlica, V. S. Mirrokni, and K. Munagala. Optimal auctions with positive network externalities. In *Proceedings of the 12th ACM Conference on Electronic Commerce, EC 2011*, pages 11–20, 2011.

13 J. Hartline, V. S. Mirrokni, and M. Sundararajan. Optimal marketing strategies over social networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 189–198, 2008.

14 D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

15 J. Kleinberg. Cascading behavior in networks: Algorithmic and economic issues. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*, chapter 24, pages 613–632. Cambridge University Press, Cambridge, 2007.

16 M. D. Plummer and L. Lovász. *Matching Theory*. North-Holland Mathematics Studies. Elsevier Science, 1986.

Trading Determinism for Time in Space Bounded Computations

Vivek Anand T Kallampally[1] and Raghunath Tewari[2]

1 **Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur, India**
vivekana@cse.iitk.ac.in

2 **Department of Computer Science & Engineering, Indian Institute of Technology, Kanpur, India**
rtewari@cse.iitk.ac.in

Abstract

Savitch showed in 1970 that nondeterministic logspace (NL) is contained in deterministic $\mathcal{O}(\log^2 n)$ space but his algorithm requires quasipolynomial time. The question whether we can have a deterministic algorithm for every problem in NL that requires polylogarithmic space and simultaneously runs in polynomial time was left open.

In this paper we give a partial solution to this problem and show that for every language in NL there exists an unambiguous nondeterministic algorithm that requires $\mathcal{O}(\log^2 n)$ space and simultaneously runs in polynomial time.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.3 Tradeoffs between Complexity Measures

Keywords and phrases space complexity, unambiguous computations, Savitch's Theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.10

1 Introduction

Deciding reachability between a pair of vertices in a graph is an important computational problem from the perspective of space bounded computations. It is well known that reachability in directed graphs characterizes the complexity class nondeterministic logspace (NL). For undirected graphs the problem was known to be hard for the class deterministic logspace (L) and in a breakthrough result Reingold showed that is contained in L as well [20]. Several other restrictions of the reachability problem are known to characterize other variants of space bounded complexity classes [12, 5, 6].

Unambiguous computations are a restriction of general nondeterministic computations where the Turing machine has at most one accepting computation path on every input. In the space bounded domain, unambiguous logspace (in short UL) is the class of languages for which there is a nondeterministic logspace bounded machine that has a unique accepting path for every input in the language and zero accepting path otherwise. UL was first formally defined and studied in [8, 2]. In 2000 Reinhardt and Allender showed that the class NL is contained in a non-uniform version of UL [21]. In a subsequent work it was shown that under the hardness assumption that deterministic linear space has functions that cannot be computed by circuits of size $2^{\epsilon n}$, it can be shown that NL = UL [1]. Although it is widely believed that NL and UL are the same unconditionally and in a uniform setting, the question still remains open.

Savitch's Theorem states that reachability in directed graphs is in $\mathsf{DSPACE}(\log^2 n)$, however the algorithm requires quasipolynomial time [22]. On the other hand standard graph traversal algorithms such as DFS and BFS can decide reachability in polynomial time (in fact linear time) but require linear space. Wigderson asked the question that can we solve reachability in $\mathcal{O}(n^{1-\epsilon})$ space and polynomial time simultaneously, for some $\epsilon > 0$ [26]. Barnes et. al. gave a partial answer to this question by giving a $\mathcal{O}(n/2^{\sqrt{\log n}})$ space and polynomial time algorithm for the problem [4]. Although this bound has been improved for several subclasses such as planar graphs [16], layered planar graphs [10], minor-free and bounded genus graphs [9], for general directed graphs (and hence for the class NL) we still do not have a better deterministic space upper bound simultaneously with polynomial time.

1.1 Main Result

In this paper we show that directed graph reachability can be decided by an unambiguous $\mathcal{O}(\log^2 n)$ space algorithm that simultaneously requires only polynomial time. Thus we get an improvement in the time required by Savitch's algorithm by sacrificing determinism. Formally, we show the following theorem.

▶ **Theorem 1.** $\mathsf{NL} \subseteq \mathsf{poly}{-}\mathsf{USPACE}(\log^2 n)$.

For the remainder of this paper all graphs that we consider are directed graphs unless stated otherwise.

1.2 Min-uniqueness of Graphs

An important ingredient of our proof is the *min-uniqueness* property of graphs. A graph G is said to be min-unique with respect to an edge weight function W if the minimum weight path between every pair of vertices in G is unique with respect to W. This turns out to be an important property and has been studied in earlier papers [27, 15, 21]. In fact, the fundamental component of Reinhardt and Allender's paper is a UL algorithm for testing whether a graph is min-unique and then deciding reachability in min-unique graphs in UL [21]. They achieve this by proposing a *double inductive counting* technique which is a clever adaptation of the inductive counting technique of Immerman and Szelepcsényi [17, 23]. As a result of Reinhardt and Allender's algorithm, in order to show that reachability in a class of graphs can be decided in UL, one only needs to design an efficient algorithm which takes as input a graph from this class and outputs an $\mathcal{O}(\log n)$ bit weight function with respect to which the graph is min-unique. This technique was successfully used to show a UL upper bound on the reachability problem in several natural subclasses of general graphs such as planar graphs [7], graphs with polynomially many paths from the start vertex to every other vertex [19], bounded genus graphs [11] and minor-free graphs [3]. For the latter two classes of graphs reachability was shown to be in UL earlier as well by giving reductions to planar graphs [18, 24]. Note that Reinhardt and Allender defines min-uniqueness for unweighted graphs where the minimum length path is unique, whereas we define it for weighted graphs where the minimum weight path is unique. However it can easily be seen that both these notions are equivalent.

1.3 Overview of the Proof

We prove Theorem 1 in two parts. We first show how to construct an $\mathcal{O}(\log^2 n)$ bit weight function W with respect to which the input graph G becomes min-unique. Our construction of the weight function W uses an iterative process to assign weights to the edges of G. We start by considering a subgraph of G having a fixed radius and construct an $\mathcal{O}(\log n)$ bit weight function with respect to which this subgraph becomes min-unique. For this we first observe that there are polynomially many paths in such a subgraph and then use the prime based hashing scheme of Fredman, Komlós and Szemerédi [14] to give distinct weights to all such paths. Thereafter, in each successive round of the algorithm, we construct a new weight function with respect to which a subgraph of double the radius of the previous round becomes min-unique and the new weight function has an additional $\mathcal{O}(\log n)$ bits. Hence in $\mathcal{O}(\log n)$ many rounds we get a weight function which has $\mathcal{O}(\log^2 n)$ bits and with respect to which G is min-unique. We show that this can be done by an unambiguous, polynomial time algorithm using $\mathcal{O}(\log^2 n)$ space. This technique is similar to the isolating weight construction in [13], but their construction is in $\mathsf{quasi{-}NC}$.

We then show that given a graph G and an $\mathcal{O}(\log^2 n)$ bit weight function with respect to which G is min-unique, reachability in G can be decided by an unambiguous, polynomial time algorithm using $\mathcal{O}(\log^2 n)$ space. Note that a straightforward application of Reinhardt and Allender's algorithm will not give the desired bound. This is because "unfolding" a graph with $\mathcal{O}(\log^2 n)$ bit weights will result in a quasipolynomially large graph. As a result we will not achieve a polynomial time bound. We tackle this problem by first observing that although there are $2^{\mathcal{O}(\log^2 n)}$ many different weight values, the weight of a shortest path can only use polynomial number of distinct such values. Using this observation we give a modified version of Reinhardt and Allender's algorithm that iterates over the "good" weight values and ignores the rest. This allows us to give a polynomial time bound.

The rest of the paper is organized as follows. In Section 2 we define the various notations and terminologies used in this paper. We also state prior results that we use in this paper. In Section 3 we give the proof of Theorem 1.

2 Preliminaries

For a positive integer n, let $[n] = \{1, 2, \ldots, n\}$. Let $G = (V, E)$ be a directed graph on n vertices and let $E = \{e_1, e_2, \ldots, e_m\}$ be the set of edges in G. Let s and t be two fixed vertices in G. We wish to decide whether there exists a path from s to t in G. The *length* of a path P is the number of edges in P and is denoted as $\mathrm{len}(P)$. The *center* of a path P is a vertex x in P such that the length of the path from either end point of P to x is at most $\lceil \mathrm{len}(P)/2 \rceil$ and x is no farther from the tail of P than from the head of P.

A *weight function* $w : E \to \mathbb{N}$ is a function which assigns a positive integer to every edge in G. The weight function w is said to be *polynomially bounded* if there exists a constant k such that $w(e) \leq \mathcal{O}(n^k)$ for every edge e in G. We use G_w to denote the weighted graph G with respect to a weight function w. For a graph G_w, the *weight of a path* P denoted by $w(P)$ is defined as the sum of weights of the edges in the path. A *shortest path* from u to v in G_w is a path from u to v with minimum weight. Let $\mathcal{P}^i_w(u, v)$ denote the set of shortest paths from u to v of length at most i in G_w. Thus in particular, the set of shortest paths from u to v in G_w, $\mathcal{P}_w(u, v) = \mathcal{P}^n_w(u, v)$.

We define the *distance* function with respect to a weight function and a nonnegative integer i as

$$\mathrm{dist}^i_w(u, v) = \begin{cases} w(P) & \text{for } P \in \mathcal{P}^i_w(u, v) \\ \infty & \text{if } \mathcal{P}^i_w(u, v) = \emptyset \end{cases}$$

Correspondingly we define the function l which represents the minimum length of such paths as

$$\mathrm{l}_w^i(u,v) = \begin{cases} \min_{P \in \mathcal{P}_w^i(u,v)} \{\mathrm{len}(P)\} & \text{if } \mathcal{P}_w^i(u,v) \neq \emptyset \\ \infty & \text{otherwise} \end{cases}$$

A graph G_w is said to be *min-unique* for paths of length at most i, if for any pair of vertices u and v, the shortest path from u to v with length at most i, is unique. G_w is said to be min-unique if G_w is min unique for paths of arbitrary length. Define weight function

$$w_0(e_i) := 2^{i-1}, \text{ where } i \in [m].$$

It is straightforward to see that for any graph G, w_0 is an n bit weight function and G_{w_0} is min-unique. Wherever it is clear from the context that there is only one weight function w, we will drop the subscript w in our notations.

For a graph G_w, vertex u in G, length i and weight value k, we define the quantities $c_k^i(u)$ and $D_k^i(u)$ as the number of vertices at a distance at most k from u, using paths of length at most i and the sum of the distances to all such vertices respectively. Formally,

$$c_k^i(u) = |\{v \mid \mathrm{dist}_w^i(u,v) \leq k\}|$$
$$D_k^i(u) = \sum_{v \mid \mathrm{dist}_w^i(u,v) \leq k} \mathrm{dist}_w^i(u,v).$$

An *unambiguous Turing machine* is a nondeterministic Turing machine that has at most one accepting computation path on every input [25]. We shall consider unambiguous computations in the context of space bounded computations. $\mathsf{USPACE}(s(n))$ denotes the class of languages decided by an unambiguous machine using $\mathcal{O}(s(n))$ space. In particular, $\mathsf{UL} = \mathsf{USPACE}(\log n)$. $\mathsf{TIME{-}USPACE}(t(n), s(n))$ denotes the class of languages decided by an unambiguous machine using $\mathcal{O}(s(n))$ space and $\mathcal{O}(t(n))$ time simultaneously. In particular, when $t(n)$ is a polynomial, we define

$$\mathsf{poly{-}USPACE}(s(n)) = \bigcup_{k \geq 0} \mathsf{TIME{-}USPACE}(n^k, s(n)).$$

For graphs having polynomially many paths, we use the well known hashing technique due to Fredman, Komlós and Szemerédi [14] to compute a weight function that assigns distinct weights to all such paths. We state the result below in a form that will be useful for our purpose.

▶ **Theorem 2.** *[14, 19] For every constant c there is a constant c' so that for every set S of n bit integers with $|S| \leq n^c$ there is a $c' \log n$ bit prime number p so that for all $x \neq y \in S$, $x \not\equiv y \bmod p$.*

Henceforth we will refer to Theorem 2 as the FKS hashing lemma.

3 Min-unique Weight Assignment

Reinhardt and Allender [21] showed that for every n there is a sequence of n^2 $\mathcal{O}(\log n)$ bit weight functions such that every graph G on n vertices is min-unique with respect to at least one of them. For each weight function they construct an unweighted graph (say G_w) by replacing every edge with a path of length equal to the weight of that edge. Since the weights are $\mathcal{O}(\log n)$ bit values therefore G_w is polynomially large in n. Next they show

Algorithm 1: Computes a min-unique weight function and checks for an $s-t$ path in G

```
   Input: (G, s, t)
   Output: weight function $W := W_q$, true if there is a path from $s$ to $t$ and false
           otherwise
1  begin
2      $q := \log n$; $W_0 := 0$
3      for $j \leftarrow 1$ to $q$ do
4          $i := 2^j$; $p := 2$
5          repeat
               /* By the FKS hashing lemma $p$ is bounded by a polynomial in $n$,
                  say $n^{c'}$.  We define $B := n^{c'+2}$.                        */
6              $W_j := B \cdot W_{j-1} + (w_0 \bmod p)$
7              Check whether $(G, W_j, i)$ is min-unique using Algorithm 2
8              $p :=$ next prime
9          until $(G, W_j, i)$ is min-unique
10     endfor
11     if $\mathrm{dist}^n_{W_q}(s,t) \leq B^q$ then return $(W_q, \mathsf{true})$
12     else return $(W_q, \mathsf{false})$
13 end
```

that using the double inductive counting technique one can check unambiguously using a logspace algorithm if G_w is min-unique, and if so then check if there is a path from s to t as well. They iterate over all weight functions until they obtain one with respect to which G_w is min-unique and use the corresponding graph G_w to check reachability. Since we use an $\mathcal{O}(\log^2 n)$ bit weight function with respect to which the input graph is min-unique, we cannot construct an unweighted graph by replacing every edge with a directed path of length equal to the corresponding edge weight.

In Section 3.1 we give an algorithm that computes an $\mathcal{O}(\log^2 n)$ bit, min-unique weight function and decides reachability in directed graphs. In Section 3.2 we check if a graph is min-unique. Although we use $\omega(\log n)$ bit weight functions, our algorithm still runs in polynomial time. In Section 3.3 we show how to compute the $\mathrm{dist}^i_w(u,v)$ function unambiguously.

3.1 Construction of the weight function

Theorem 3 shows how to construct the desired weight function.

▶ **Theorem 3.** *There is a nondeterministic algorithm that takes as input a directed graph G and outputs along a unique computation path, an $\mathcal{O}(\log^2 n)$ bit weight function W such that G_W is min-unique, while all other computation paths halt and reject. For any two vertices s and t the algorithm also checks whether there is a path from s to t in G. The algorithm uses $\mathcal{O}(\log^2 n)$ space and runs in polynomial time.*

Since directed graph reachability is complete for NL, Theorem 1 follows from Theorem 3.

Proof of Theorem 3. To prove Theorem 3 we design an algorithm that outputs the desired weight function. The formal description of the construction is given in Algorithm 1. The algorithm works in an iterative manner for $\log n$ number of rounds. Initially we consider all paths in G of length at most l where $l = 2^1$. The number of such paths is bounded by n^l and therefore by the FKS hashing lemma there exist a $c' \log n$ bit prime p_1 such that with respect to the weight function $W_1 := w_0 \bmod p_1$, G_{w_1} is min-unique for paths of length at most l. To find the right prime p_1 we iterate over all $c' \log n$ bit primes and use Lemma 7 to check whether G_{w_1} is min-unique for paths of length at most l.

We prove this by induction on the number of rounds, say j. Assume that $G_{W_{j-1}}$ is min-unique for paths of length at most 2^{j-1}. In the j-th round, the algorithm considers all paths of length at most 2^j. By applying Lemma 4 we get a weight function W_j from W_{j-1} which uses $\mathcal{O}(j \cdot \log n)$ bits and G_{W_j} is min-unique for paths of length at most 2^j. Hence in $\log n$ many rounds we get a weight function $W := W_{\log n}$ such that G_W is min-unique. Note that the inner **repeat-until** loop runs for at most $n^{c'}$ iterations due to the FKS hashing lemma.

Let p_j be the prime used in the j-th round of Algorithm 1. Define $p' := \max\{p_j \mid j \in [\log n]\}$. By the FKS hashing lemma p' is bounded by a polynomial in n, say $n^{c'}$. We set $B := n^{c'+2}$. This implies that for any weight function of the form $w = w_0 \bmod p_j$ and any path P in G, $w(P) < B$. Observe that with respect to the final weight function W, for any path P in G, $W(P) < B^q$.

Once we compute an $\mathcal{O}(\log^2 n)$ bit weight function W such that G_W is min-unique, there exist a path from s to t if and only if $\text{dist}^n_W(s,t) \leq B^q$. This can be checked using Algorithm 5 in $\mathcal{O}(\log^2 n)$ space and polynomial time. Also Algorithm 5 is a nondeterministic algorithm which returns true or false along a unique computation path while all other computation paths halt and reject.

In each round the size of W_j increases by $\mathcal{O}(\log n)$ bits and after $\log n$ rounds $W_{\log n}$ is an $\mathcal{O}(\log^2 n)$ bit weight function. By Lemma 7 checking whether a graph is min-unique with respect to an $\mathcal{O}(\log^2 n)$ bit weight function requires $\mathcal{O}(\log^2 n)$ space. Thus the total space complexity of Algorithm 1 is $\mathcal{O}(\log^2 n)$.

The FKS hashing lemma guarantees that in each round only a polynomial number of primes need to be tested to find a weight function which is min-unique for paths of length at most 2^j. By Lemma 7 checking whether a graph is min-unique for paths of length at most 2^j can be done in polynomial time. Thus each round runs in polynomial time. There are only $\log n$ many round and hence Algorithm 1 runs in polynomial time.

By Lemma 7, Algorithm 2 is a nondeterministic algorithm which outputs its answer along a unique computation path, while all other computation paths halt and reject. All other steps in Algorithm 1 are deterministic. This shows the unambiguity requirement of the theorem. ◀

▶ **Lemma 4.** *There is a nondeterministic algorithm $\mathcal{A}$, that takes as inputs (G, w) where G is a graph on n vertices and w is a k bit weight function such that G_w is min-unique for paths of length at most l. $\mathcal{A}$ outputs a $(k + \mathcal{O}(\log n))$ bit weight function w' such that $G_{w'}$ is min-unique for paths of length at most $2l$, along a unique computation path while all other computation paths halt and reject. $\mathcal{A}$ uses $\mathcal{O}(k + \mathcal{O}(\log n))$ space and runs in polynomial time.*

▶ Remark. The encoding of the output weight function w' is the concatenation of the k bit representation of the input weight function w and an $\mathcal{O}(\log n)$ bit prime number p. The output weight function w' is calculated as $w' := B \cdot w + w_0 \bmod p$, where B is the number defined in Algorithm 1. Multiplication using B is used just to left shift w and make room for the new function $w_0 \bmod p$.

Lemma 4 proves the correctness of each iteration of the outer **for** loop of Algorithm 1. Before proving the lemma, we will show that if G_w is min-unique for paths of length at most l, then the number of minimum weight paths with respect to w of length at most $2l$ is bounded by a polynomial independent of l. Hence it allows us to use the FKS hashing lemma to isolate such paths.

▶ **Lemma 5.** *Let G be a graph with n vertices and w be a weight function such the graph G_w is min-unique for paths of length at most l. Then for any pair of vertices u and v, $\left|\mathcal{P}_w^{2l}(u,v)\right|$ is at most n.*

Proof. Let P be a shortest path from u to v in G_w with length at most $2l$ with center vertex x. That is $P \in \mathcal{P}_w^{2l}(u,v)$. Let P_1 and P_2 be the subpaths from u to x and x to v. Since x is the center of P, P_1 has length at most l. Note that P_1 is the unique shortest path of length at most l from u to x in G_w. This is because if there exists another path of length at most l with a smaller weight than P_1 from u to x then replacing P_1 with this path in P will result in a path of length at most $2l$ from u to v with a lower weight than P. But this cannot happen since P is a shortest path from u to v.

▶ **Claim 6.** *There is only one shortest path of length at most $2l$ from u to v with x as its center.*

Proof. Assume there is another shortest path P' of length at most $2l$ from u to v with x as its center. Let P_1' be the subpath of P' from u to x. Since x is the center of P', P_1' is of length at most l. Similar to P_1, P_1' is a shortest path of length at most l from u to x. This means there are two shortest paths of length at most l from u to x. This is a contradiction since G is min-unique for paths of length at most l. ◀

Therefore each vertex can be the center of at most one path of length at most $2l$ from u to v. Thus the total number of shortest paths of length at most $2l$ from u to v in G_w is at most n. Hence $\left|\mathcal{P}_w^{2l}(u,v)\right| \leq n$. This completes the proof of Lemma 5. ◀

When we sum over all possible pairs of u and v, the total number of shortest paths of length at most $2l$ in G_w is at most n^3.

Proof of Lemma 4. G_w is min-unique for paths of length at most l. Therefore by Lemma 5 the number of shortest paths between all pairs of vertices with at most $2l$ edges in G is at most n^3. Let $\mathcal{S}$ be the set of these n^3 shortest paths. With respect to the weight function w_0 (see Section 2) each element of $\mathcal{S}$ gets a distinct weight. So by using the FKS hashing lemma we get a constant c' and a $c' \log n$ bit prime number p such that with respect to the weight function $\widehat{w}$ such that $\widehat{w} := w_0 \bmod p$, each element of $\mathcal{S}$ gets a distinct weight. Moreover, in G between any pair of vertices the shortest path in $\mathcal{S}$ is unique.

Let B be the number as defined in Algorithm 1. Now consider the weight function $w' := B \cdot w + \widehat{w}$. Since w is a k bit weight function and $\widehat{w}$ is an $\mathcal{O}(\log n)$ bit weight function therefore w' is a $(k + \mathcal{O}(\log n))$ bit weight function. Clearly w has higher precedence than $\widehat{w}$ in w'. So for any two paths P_1 and P_2 in G , we have if $w'(P_1) < w'(P_2)$ then either $w(P_1) < w(P_2)$ or both the predicates $w(P_1) = w(P_2)$ and $\widehat{w}(P_1) < \widehat{w}(P_2)$ are true. Additionally if $w'(P_1) = w'(P_2)$ then $w(P_1) = w(P_2)$ and $\widehat{w}(P_1) = \widehat{w}(P_2)$.

All the unique shortest paths of length at most $2l$ in G_w, will be unique shortest paths of length at most $2l$ in $G_{w'}$ also. If there are multiple shortest paths of length at most $2l$ from u to v in G_w, $\widehat{w}$ gives a unique weight to each of these paths. So $G_{w'}$ is min-unique for paths of length at most $2l$.

Algorithm 2: Check whether G is min-unique for paths of length at most i

```
Input: $(G, w, i)$
Output: true if $G_w$ is not min-unique for paths of length at most $i$ and false otherwise
1  begin
2      BAD.WEIGHT := false
       /* BAD.WEIGHT is set to true whenever the weight function does not
          make the graph min-unique. Otherwise it remains false. It is a
          boolean variable shared between Algorithms 4 and 2                */
3      for each vertex $v$ do
4          $c_0^i(v) := 1$; $D_0^i(v) := 0$; $k' := 0$
5          repeat
6              $k := k'$; $c_k^i(v) := c_{k'}^i(v)$; $D_k^i(v) := D_{k'}^i(v)$
7              Find next $k'$ from $(G, w, v, i, k, c_k^i(v), D_k^i(v))$ using Algorithm 3
8              if $k' = \infty$ then break
9              Compute $(c_{k'}^i(v), D_{k'}^i(v))$ from $(G, w, v, i, k, c_k^i(v), D_k^i(v), k')$ using
               Algorithm 4
10         until BAD.WEIGHT = true
11         if BAD.WEIGHT = true then break
12     endfor
13     return BAD.WEIGHT
14 end
```

We can check whether a graph $G_{w'}$ is min-unique for paths of length at most $2l$ using Lemma 7. Since p is an $c' \log n$ bit prime number, we can iterate over all the $c' \log n$ bit primes and find p. ◀

3.2 Checking for min-uniqueness

The next lemma shows how to check whether G_w is min-unique for paths of length at most l in an unambiguous manner.

▶ **Lemma 7.** *There is a nondeterministic algorithm that takes as input a directed graph G, a k bit weight function w and a length i and outputs along a unique computation path whether or not the graph G_w is min-unique for paths of length at most i, while all other computation paths halt and reject. The algorithm uses $\mathcal{O}(k + \log n)$ space and runs in polynomial time.*

For every vertex v in the G_w we check whether there are two minimum weight paths of length at most i to some other vertex in G. Algorithm 2 gives a formal description of this process. The algorithm iterates over all shortest path weight values that can be achieved by some path of length at most i.

In the k-th stage of the algorithm it considers a ball of radius k consisting of vertices which have a shortest path of weight at most k from v and length at most i. $c_k^i(v)$ denotes the number of vertices in this ball and $D_k^i(v)$ denotes the sum of the weights of the shortest paths to all such vertices. Initially $k = 0$, $c_0^i(v) = 1$ (consisting of only the vertex v) and $D_0^i(v) = 0$.

A direct implementation of the double inductive counting technique of Reinhardt and Allender [21] does not work since this would imply that we cycle over all possible weight values, which we cannot afford. We bypass this hurdle by considering only the relevant

Algorithm 3: Find the next smallest weight value $k' > k$ among all paths of length at most i from u

```
Input: $(G, w, u, i, k, c_k^i(u), D_k^i(u))$
Output: $k' := \min\{\text{dist}_w^i(u,v) \mid \text{dist}_w^i(u,v) > k,\ v \in V\}$
begin
    $k' := \infty$
    for each vertex $v$ do
        if $\neg(\text{dist}_w^i(u,v) \le k)$ then
            $\text{min.dist}_w^i(u,v) := \infty$
            for each $x$ such that $(x,v)$ is an edge do
                if $\text{dist}_w^i(u,x) \le k$ and $\text{l}_w^i(u,x) + 1 \le i$ then
                    if $\text{min.dist}_w^i(u,v) > \text{dist}_w^i(u,x) + w(x,v)$ then
                        $\text{min.dist}_w^i(u,v) := \text{dist}_w^i(u,x) + w(x,v)$
                    endif
                endif
            endfor
            if $k' > \text{min.dist}_w^i(u,v)$ then $k' := \text{min.dist}_w^i(u,v)$
        endif
    endfor
    return $k'$
end
```

weight values. We compute the immediate next shortest path weight value k', and use k' as the weight value for the next stage of the algorithm. This computation is implemented in Algorithm 3). Lemma 8 proves the correctness of this process. Note that the number of shortest path weight values from a fixed vertex is bounded by the number of vertices in the graph. This ensure that the number of iterations of the inner **repeat-until** loop of Algorithm 2 is bounded by n.

▶ **Lemma 8.** *Given $(G, w, u, i, k, c_k^i(u), D_k^i(u))$, Algorithm 3 correctly computes the value $\min\{\text{dist}_w^i(u,v) \mid \text{dist}_w^i(u,v) > k,\ v \in V\}$.*

To see the correctness of Lemma 8 observe that for every vertex v such that $\text{dist}_w^i(u,v) > k$, the algorithm cycles through all vertices x such that there is an edge from x to v and the length of the path from u to x is at most $i-1$. It computes the minimum weight of such a path and store it in the variable $\text{min.dist}_w^i(u,v)$. It then computes the minimum value of $\text{min.dist}_w^i(u,v)$ over all possible vertices v and outputs it as k', as required.

After we get the appropriate weight value k', we then compute the values of $c_{k'}^i(v)$ and $D_{k'}^i(v)$ by using a technique similar to Reinhardt and Allender (implemented in Algorithm 4). Additionally we also maintain a shared flag value BAD.WEIGHT between Algorithms 2 and 4, which is set to true if G_w is not min-unique for paths of length at most i, else it is false.

3.3 Computing the $\text{dist}_w^i(u,v)$ function

In Algorithms 3 and 4, an important step is to check whether $\text{dist}_w^i(u,v) \le k$ and if so, get the values of $\text{dist}_w^i(u,v)$ and $\text{l}_w^i(u,v)$. These values are obtained from Algorithm 5. Algorithm 5 describes a nondeterministic procedure that takes as input a weighted graph G_w, which is min-unique for paths of length at most i and weight at most k from a source

Algorithm 4: Compute $c^i_{k'}(u)$ and $D^i_{k'}(u)$ and check whether G_w is min-unique for paths with length at most i and weight at most k' from u

```
   Input: $(G, w, u, i, k, c^i_k(u), D^i_k(u), k')$
   Output: $(c^i_{k'}(u), D^i_{k'}(u))$ and also flag BAD.WEIGHT
 1 begin
 2     $c^i_{k'}(u) := c^i_k(u)$; $D^i_{k'}(u) := D^i_k(u)$
 3     for each vertex $v$ do
 4         if $\neg(\mathrm{dist}^i_w(u,v) \leq k)$ then
 5             for each $x$ such that $(x,v)$ is an edge do
 6                 if $\mathrm{dist}^i_w(u,x) \leq k$ and $\mathrm{dist}^i_w(u,x) + w(x,v) = k'$ and $\mathrm{l}^i_w(u,x) + 1 \leq i$
                   then
 7                     $c^i_{k'}(u) := c^i_{k'}(u) + 1$; $D^i_{k'}(u) := D^i_{k'}(u) + k'$
 8                     for each $x' \neq x$ such that $(x',v)$ is an edge do
 9                         if $\mathrm{dist}^i_w(u,x') \leq k$ and $\mathrm{dist}^i_w(u,x') + w(x',v) = k'$ and
                           $\mathrm{l}^i_w(u,x') + 1 \leq i$ then
10                             BAD.WEIGHT := true
11                         endif
12                     endfor
13                 endif
14             endfor
15         endif
16     endfor
17     return $(c^i_{k'}(u), D^i_{k'}(u))$
18 end
```

vertex u and the values $c^i_k(u)$ and $D^i_k(u)$. For any vertex v, if $\mathrm{dist}^i_w(u,v) \leq k$ then it outputs true and the values of $\mathrm{dist}^i_w(u,v)$ and $\mathrm{l}^i_w(u,v)$ along a unique computation path. Otherwise it outputs false along a unique computation path with ∞ as the values of $\mathrm{dist}^i_w(u,v)$ and $\mathrm{l}^i_w(u,v)$. All other computation paths halt and reject. As a result we can compute the predicate $\neg(\mathrm{dist}^i_w(u,v) \leq k)$ along a unique path as well.

Note that Algorithm 5 is the only algorithm where we use non-determinism. The algorithm is similar to the unambiguous subroutine of Reinhardt and Allender [21] with the only difference being that here we consider weight of a path instead of length of a path. The algorithm assumes that the subgraph induced by all the paths of length at most i and weight at most k from u is min-unique.

In Line 5 of Algorithm 5, for each vertex x the routine non-deterministically guesses whether $\mathrm{dist}^i_w(u,x) \leq k$ and if the guess is 'true', it then guesses a path of length at most k from u to x. If the algorithm incorrectly guesses for some vertex x that $\mathrm{dist}^i_w(u,x) > k$, then the variable *count* will never reach $c^i_k(u)$ and the routine will reject. If it guesses incorrectly that $\mathrm{dist}^i_w(u,x) \leq k$ it will fail to guess a correct path in Line 7 and again reject that computation. Thus the only computation paths that exit the **for** loop in Line 16 and satisfy the first condition of the **if** statement in Line 17, are the ones that correctly guess exactly the set $\{x \mid \mathrm{dist}^i_w(u,x) \leq k\}$. If the algorithm ever guesses incorrectly the weight d of the shortest path to x, then if $\mathrm{dist}^i_w(u,x) > d$ no path of weight d will be found, and if $\mathrm{dist}^i_w(u,x) < d$ then the variable *sum* will be incremented by a value greater than $\mathrm{dist}^i_w(u,x)$. In the latter case, at the end of the algorithm, *sum* will be greater than $D^i_k(u)$, and the routine will reject.

Algorithm 5: An unambiguous routine to determine if $\mathrm{dist}_w^i(u,v) \leq k$ and find $\mathrm{dist}_w^i(u,v)$ and $\mathrm{l}_w^i(u,v)$

```
Input: $(G, w, u, i, k, c_k^i(u), D_k^i(u), v)$
Output: (true or false), $\mathrm{dist}_w^i(u,v)$, $\mathrm{l}_w^i(u,v)$
begin
    $count := 0$; $sum := 0$; $path.to.v :=$ false
    $\mathrm{dist}_w^i(u,v) := \infty$; $\mathrm{l}_w^i(u,v) := \infty$
    for each $x \in V$ do
        Guess non deterministically if $\mathrm{dist}_w^i(u,x) \leq k$ in $G_w$
        if the guess is $\mathrm{dist}_w^i(u,x) \leq k$ then
            Guess a path of weight $d \leq k$ and length $l \leq i$ from $u$ to $x$
            (If this fails then halt and reject)
            $count := count + 1$; $sum := sum + d$
            if $x = v$ then
                $path.to.v :=$ true
                $\mathrm{dist}_w^i(u,v) := d$
                $\mathrm{l}_w^i(u,v) := l$
            endif
        endif
    endfor
    if $count = c_k^i(u)$ and $sum = D_k^i(u)$ then
        return $(path.to.v, \mathrm{dist}_w^i(u,v), \mathrm{l}_w^i(u,v))$
    else
        halt and reject
    endif
end
```

Since G_w is min-unique for paths of length at most i and weight at most k from u, only for exactly one computation path *sum* and *count* will match with $c_k^i(u)$ and $D_k^i(u)$. So except one computation path which made all the guesses correct, all other paths halt and reject. If $\mathrm{dist}_w^i(u,v) \leq k$ then even though the algorithm uses non-deterministic choices, it outputs 'true' along a single computation path while all other paths halt and reject. Also if $\mathrm{dist}_w^i(u,v) > k$, the algorithm outputs 'false' along a single computation path while all other paths halt and reject. The space complexity of the algorithm is bounded by the size of the weight function w.

As a corollary of Theorem 1 we get the following result.

▶ **Corollary 9.** *For $s(n) \geq \log n$,* $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{TIME{-}USPACE}(2^{\mathcal{O}(s(n))}, s^2(n))$.

References

1 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

2 Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

3 Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing isolation lemma for $k_{3,3}$-free and k_5-free bipartite graphs. In *33rd Symposium on Theoretical Aspects*

of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France, pages 10:1–10:15, 2016.

4 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. In *Structure in Complexity Theory Conference, 1992., Proceedings of the Seventh Annual*, pages 27–33, 1992.

5 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

6 David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *15th International Symposium on Theoretical Aspects of Computer Science (STACS)*, Volume 1373 in Lecture Notes in Computer Science, pages 74–83. Springer, 1998.

7 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. In *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, pages 217–221, 2007. `doi:10.1109/CCC.2007.9`.

8 Gerhard Buntrock, Birgit Jenner, Klaus-Jörn Lange, and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In *Proceedings of the 8th International Conference on Fundamentals of Computation Theory (FCT'91)*, Volume 529 Lecture Notes in Computer Science, pages 168–179. Springer-Verlag, 1991.

9 Diptarka Chakraborty, Aduri Pavan, Raghunath Tewari, N. V. Vinodchandran, and Lin F. Yang. New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 585–595, 2014.

10 Diptarka Chakraborty and Raghunath Tewari. An $O(n^\epsilon)$ space and polynomial time algorithm for reachability in directed layered planar graphs. In *In 26th International Symposium on Algorithms and Computation, ISAAC 2015, Nagoya, Japan, December 9-11, 2015*, pages 614–624, 2015.

11 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space Complexity of Perfect Matching in Bounded Genus Bipartite Graphs. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 579–590, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.STACS.2011.579`.

12 Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, June 1997.

13 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. *CoRR*, abs/1601.06319, 2016. URL: `http://arxiv.org/abs/1601.06319`.

14 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, June 1984. `doi:10.1145/828.1884`.

15 Anna Gál and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Struct. Algorithms*, 9(1-2):99–111, 1996. URL: `http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1098-2418(199608/09)9:1/2<99::AID-RSA7>3.0.CO;2-6/abstract`.

16 T. Imai, K. Nakagawa, A. Pavan, N.V. Vinodchandran, and O. Watanabe. An $O(n^{1/2+\epsilon})$-Space and Polynomial-Time Algorithm for Directed Planar Reachability. In *Computational Complexity (CCC), 2013 IEEE Conference on*, pages 277–286, 2013. `doi:10.1109/CCC.2013.35`.

17 Neil Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

18 Jan Kynčl and Tomáš Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Transactions on Computation Theory*, 1(3):1–11, 2010. `doi:10.1145/1714450.1714451`.

19 Aduri Pavan, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in log-space. *Computational Complexity*, 21(4):643–670, 2012. `doi:10.1007/s00037-012-0047-3`.

20 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4), 2008.

21 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29(4):1118–1131, 2000. `doi:10.1137/S0097539798339041`.

22 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4:177–192, 1970.

23 Robert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

24 Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free Graphs and K_5-free Graphs is in Unambiguous Log-Space. In *17th International Conference on Foundations of Computation Theory (FCT)*, Lecture Notes in Computer Science 5699, pages 323–334. Springer-Verlag, 2009.

25 Leslie G. Valiant. Relative complexity of checking and evaluating. *Inf. Process. Lett.*, 5(1):20–23, 1976. `doi:10.1016/0020-0190(76)90097-1`.

26 Avi Wigderson. The complexity of graph connectivity. *Mathematical Foundations of Computer Science 1992*, pages 112–132, 1992.

27 Avi Wigderson. $NL/poly \subseteq \oplus L/poly$ (preliminary version). In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference, Amsterdam, The Netherlands, June 28 - July 1, 1994*, pages 59–62, 1994. `doi:10.1109/SCT.1994.315817`.

Families of DFAs as Acceptors of ω-Regular Languages*

Dana Angluin[1], Udi Boker[2], and Dana Fisman[3]

1 Yale University, New Haven, CT, USA
2 The Interdisciplinary Center (IDC), Herzliya, Israel
3 University of Pennsylvania, Philadelphia, PA, USA

Abstract

Families of DFAs (FDFAs) provide an alternative formalism for recognizing ω-regular languages. The motivation for introducing them was a desired correlation between the automaton states and right congruence relations, in a manner similar to the Myhill-Nerode theorem for regular languages. This correlation is beneficial for learning algorithms, and indeed it was recently shown that ω-regular languages can be learned from membership and equivalence queries, using FDFAs as the acceptors.

In this paper, we look into the question of how suitable FDFAs are for defining ω-regular languages. Specifically, we look into the complexity of performing Boolean operations, such as complementation and intersection, on FDFAs, the complexity of solving decision problems, such as emptiness and language containment, and the succinctness of FDFAs compared to standard deterministic and nondeterministic ω-automata.

We show that FDFAs enjoy the benefits of deterministic automata with respect to Boolean operations and decision problems. Namely, they can all be performed in nondeterministic logarithmic space. We provide polynomial translations of deterministic Büchi and co-Büchi automata to FDFAs and of FDFAs to nondeterministic Büchi automata (NBAs). We show that translation of an NBA to an FDFA may involve an exponential blowup. Last, we show that FDFAs are more succinct than deterministic parity automata (DPAs) in the sense that translating a DPA to an FDFA can always be done with only a polynomial increase, yet the other direction involves an inevitable exponential blowup in the worst case.

1998 ACM Subject Classification F.1.1 Automata, D.2.4 Formal methods

Keywords and phrases Finite automata, Omega Regular Languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.11

1 Introduction

The theory of finite-state automata processing infinite words was developed in the early sixties, starting with Büchi [3] and Muller [13], and motivated by problems in logic and switching theory. Today, automata for infinite words are extensively used in verification and synthesis of *reactive systems*, such as operating systems and communication protocols.

An automaton processing finite words makes its decision according to the last visited state. On infinite words, Büchi defined that a run is accepting if it visits a designated set of states infinitely often. Since then several other accepting conditions were defined, giving rise to various ω-automata, among which are Muller, Rabin, Streett and parity automata.

* Research of the third author was supported by US NSF grant CCF-1138996

The theory of ω-regular languages is more involved than that of finite words. This was first evidenced by Büchi's observation that nondeterministic Büchi automata are more expressive than their deterministic counterpart. While for some types of ω-automata the nondeterministic and deterministic variants have the same expressive power, none of them possesses all the nice qualities of acceptors for finite words. In particular, none has a corresponding Myhill-Nerode theorem [16], i.e. a direct correlation between the states of the automaton and the equivalence classes corresponding to the canonical right congruence of the recognized language.

The absence of a Myhill-Nerode like property in ω-automata has been a major drawback in obtaining learning algorithms for ω-regular languages, a question that has received much attention lately due to applications in verification and synthesis, such as black-box checking [17], assume-guarantee reasoning [14], error localization [5], regular model checking [15] and more. The reason is that learning algorithms typically build on this correspondence between the automaton and the right congruence.

Recently, two algorithms for learning an unknown ω-regular language were proposed, both using non-conventional acceptors. One uses a reduction due to [4] named $L_{\$}$-automata of ω-regular languages to regular languages [6], and the other uses a representation termed *families of* DFA*s* [1]. Both representations are founded on the following well known property of ω-regular languages: two ω-regular languages are equivalent iff they agree on the set of ultimately periodic words. An ultimately periodic word uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$, can be represented as a pair of finite words (u, v). Both $L_{\$}$-automata and families of DFAs process such pairs and interpret them as the corresponding ultimately periodic words. Families of DFAs have been shown to be up to exponentially more succinct than $L_{\$}$-automata [1].

A family of DFAs (FDFA) is composed of a *leading automaton* $\mathcal{Q}$ with no accepting states and for each state q of $\mathcal{Q}$, a *progress* DFA $\mathcal{P}_q$. Intuitively, the leading automaton is responsible for processing the non-periodic part u, and depending on the state q reached when $\mathcal{Q}$ terminated processing u, the respective progress DFA $\mathcal{P}_q$ processes the periodic part v, and determines whether the pair (u, v), which corresponds to uv^ω, is accepted. (The exact definition is more subtle and is provided in Section 3.) If the leading automaton has n states and the size of the maximal progress DFA is k, we say that the FDFA is of size (n, k). An earlier definition of FDFAs, given in [9], provided a machine model for the *families of right congruences* of [10]. They were redefined in [1], where their acceptance criterion was adjusted, and their size was reduced by up to a quadratic factor. We follow the definition of [1].

In order for an FDFA to properly characterize an ω-regular language, it must satisfy the *saturation* property: considering two pairs (u, v) and (u', v'), if $uv^\omega = u'v'^\omega$ then either both (u, v) and (u', v') are accepted or both are rejected (cf. [4, 20]). Saturated FDFAs are shown to exactly characterize the set of ω-regular languages. Saturation is a semantic property, and the check of whether a given FDFA is saturated is shown to be in PSPACE. Luckily, the FDFAs that result from the learning algorithm of [1] are guaranteed to be saturated.

Saturated FDFAs bring an interesting potential – they have a Myhill-Nerode like property, and while they are "mostly" deterministic, a nondeterministic aspect is hidden in the separation of the prefix and period parts of an ultimately periodic infinite word. This gives rise to the natural questions of how "dominant" are the determinism and nondeterminism in FDFAs, and how "good" are they for representing ω-regular languages. These abstract questions translate to concrete questions that concern the succinctness of FDFAs and the complexity of solving their decision problems, as these measures play a key role in the usefulness of applications built on top of them.

Our purpose in this paper is to analyze the FDFA formalism and answer these questions.

Specifically, we ask: What is the complexity of performing the Boolean operations of complementation, union, and intersection on saturated FDFAs? What is the complexity of solving the decision problems of membership, emptiness, universality, equality, and language containment for saturated FDFAs? How succinct are saturated FDFAs, compared to deterministic and nondeterministic ω-automata?

We show that saturated FDFAs enjoy the benefits of deterministic automata with respect to Boolean operations and decision functions. Namely, the Boolean operations can be performed in logarithmic space, and the decision problems can be solved in nondeterministic logarithmic space. The constructions and algorithms that we use extend their counterparts on standard DFAs. In particular, complementation of saturated FDFAs can be obtained on the same structure, and union and intersection is done on a product of the two given structures. The correctness proof of the latter is a bit subtle.

As for the succinctness, which turns out to be more involved, we show that saturated FDFAs properly lie in between deterministic and nondeterministic ω-automata. We provide polynomial translations from deterministic ω-automata to FDFAs and from FDFAs to nondeterministic ω-automata, and show that an exponential state blowup in the opposite directions is inevitable in the worst case.

Specifically, a saturated FDFA of size (n, k) can always be transformed into an equivalent nondeterministic Büchi automaton (NBA) with $O(n^2k^3)$ states. As for the other direction, transforming an NBA with n states to an equivalent FDFA is shown to be in $2^{\Theta(n \log n)}$. This is not surprising since, as shown by Michel [12], complementing an NBA involves a $2^{\Omega(n \log n)}$ state blowup, while FDFA complementation requires no state blowup.

Considering deterministic ω-automata, a Büchi or co-Büchi automaton (DBA or DCA) with n states can be transformed into an equivalent FDFA of size $(n, 2n)$, and a deterministic parity automaton (DPA) with n states and k colors can be transformed into an equivalent FDFA of size (n, kn). As for the other direction, since DBA and DCA do not recognize all the ω-regular languages, while saturated FDFAs do, a transformation from an FDFA to a DBA or DCA need not exist. Comparing FDFAs to DPAs, which do recognize all ω-regular languages, we get that FDFAs can be exponentially more succinct: We show a family of languages $\{L_n\}_{n\geq 1}$, such that for every n, there exists an FDFA of size $(n+1, n^2)$ for L_n, but any DPA recognizing L_n must have at least 2^{n-1} states. (A deterministic Rabin or Streett automaton for L_n is also shown to be exponential in n, requiring at least $2^{\frac{n}{2}}$ states.)

Due to lack of space, some proofs are omitted and can be found in the full version, on the authors' home pages.

2 Preliminaries

An *alphabet* Σ is a finite set of symbols. The set of finite words over Σ is denoted by Σ^*, and the set of infinite words, termed ω-words, over Σ is denoted by Σ^ω. As usual, we use x^*, x^+, and x^ω to denote finite, non-empty finite, and infinite concatenations of x, respectively, where x can be a symbol, a finite word, or a langugae. We use ϵ for the empty word and Σ^+ for $\Sigma^* \setminus \{\epsilon\}$. An infinite word w is *ultimately periodic* if there are two finite words $u \in \Sigma^*$ and $v \in \Sigma^+$, such that $w = uv^\omega$. A *language* is a set of finite words, that is, a subset of Σ^*, while an ω-language is a set of ω-words, that is, a subset of Σ^ω. For natural numbers i and j and a word w, we use $[i..j]$ for the set $\{i, i+1, \ldots, j\}$, $w[i]$ for the i-th letter of w, and $w[i..j]$ for the subword of w starting at the i-th letter and ending at the j-th letter, inclusive.

An *automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, \iota, \delta \rangle$ consisting of an alphabet Σ, a finite set Q of states, an initial state $\iota \in Q$, and a transition function $\delta : Q \times \Sigma \to 2^Q$. A run of an automaton on a finite word $v = a_1a_2 \dots a_n$ is a sequence of states $q_0, q_1, \dots, q_n$ such that $q_0 = \iota$, and for each $i \geq 0$, $q_{i+1} \in \delta(q_i, a_i)$. A run on an infinite word is defined similarly and results in an infinite sequence of states. The transition function is naturally extended to a function $\delta : Q \times \Sigma^* \to 2^Q$, by defining $\delta(q, \epsilon) = \{q\}$, and $\delta(q, av) = \cup_{p\in\delta(q,a)}\delta(p, v)$ for $q \in Q$, $a \in \Sigma$, and $v \in \Sigma^*$. We often use $\mathcal{A}(v)$ as a shorthand for $\delta(\iota, v)$ and $|\mathcal{A}|$ for the number of states in Q. We use $\mathcal{A}^q$ to denote the automaton $\langle\Sigma, Q, q, \delta\rangle$ obtained from $\mathcal{A}$ by replacing the initial state with q. We say that $\mathcal{A}$ is *deterministic* if $|\delta(q, a)| \leq 1$ and *complete* if $|\delta(q, a)| \geq 1$, for every $q \in Q$ and $a \in \Sigma$. For simplicity, we consider all automata to be complete. (As is known, every automaton can be linearly translated to an equivalent complete automaton.)

By augmenting an automaton with an acceptance condition α, thereby obtaining a tuple $\langle\Sigma, Q, \iota, \delta, \alpha\rangle$, we get an *acceptor*, a machine that accepts some words and rejects others. An acceptor accepts a word if at least one of the runs on that word is accepting. For finite words the acceptance condition is a set $F \subseteq Q$ of *accepting states*, and a run on a word v is accepting if it ends in an accepting state, i.e., if $\delta(\iota, v)$ contains an element of F. For infinite words, there are various acceptance conditions in the literature; here we mention three: Büchi, co-Büchi, and parity. The Büchi and co-Büchi acceptance conditions are also a set $F \subseteq Q$. A run of a Büchi automaton is accepting if it visits F infinitely often. A run of a co-Büchi automaton is accepting if it visits F only finitely many times. A parity acceptance condition is a map $\kappa : Q \to [1..k]$ assigning each state a color (or rank). A run is accepting if the minimal color visited infinitely often is odd. We use $[\![\mathcal{A}]\!]$ to denote the set of words accepted by a given acceptor $\mathcal{A}$, and say that $\mathcal{A}$ *accepts* or *recognizes* $[\![\mathcal{A}]\!]$. Two acceptors $\mathcal{A}$ and $\mathcal{B}$ are *equivalent* if $[\![\mathcal{A}]\!] = [\![\mathcal{B}]\!]$.

We use three letter acronyms to describe acceptors, where the first letter is either D or N depending on whether the automaton is *deterministic* or *nondeterministic*, respectively. The second letter is one of {F,B,C,P}: F if this is an acceptor over finite words, B, C, or P if it is an acceptor over infinite words with Büchi, co-Büchi, or parity acceptance condition, respectively. The third letter is always A for acceptor.

For finite words, NFA and DFA have the same expressive power. A language is said to be *regular* if it is accepted by an NFA. For infinite words, the theory is more involved. While NPAs, DPAs, and NBAs have the same expressive power, DBAs, NCAs, and DCAs are strictly weaker than NBAs. An ω-language is said to be *ω-regular* if it is accepted by an NBA.

3 Families of DFAs (FDFAs)

It is well known that two ω-regular languages are equivalent if they agree on the set of ultimately periodic words (this is a consequence of McNaughton's theorem [11]). An ultimately periodic word uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$, is usually represented by the pair (u, v). A canonical representation of an ω-regular language can thus consider only ultimately periodic words, namely define a language of pairs $(u, v) \in \Sigma^* \times \Sigma^+$. Such a representation $\mathcal{F}$ should satisfy the *saturation* property: considering two pairs (u, v) and (u', v'), if $uv^\omega = u'v'^\omega$ then either both (u, v) and (u', v') are accepted by $\mathcal{F}$ or both are rejected by $\mathcal{F}$.

A family of DFAs (FDFA) accepts such pairs (u, v) of finite words. Intuitively, it consists of a *leading automaton* $\mathcal{Q}$ with no acceptance condition that runs on the prefix-word u, and for each state q of $\mathcal{Q}$, a *progress automaton* $\mathcal{P}_q$, which is a DFA that runs on the period-word v.

A straightforward definition of acceptance for a pair (u, v), could have been that the run of the leading automaton $\mathcal{Q}$ on u ends at some state q, and the run of the progress automaton $\mathcal{P}_q$ on v is accepting. This goes along the lines of $L_\$$-automata [4]. However, such an acceptance definition does not fit well the saturation requirement, and might enforce very large automata [1]. The intuitive reason is that every progress automaton might need to handle the period-words of all prefix-words.

To better fit the saturation requirement, the acceptance condition of an FDFA is defined with respect to a *normalization* of the input pair (u, v). The normalization is a new pair (x, y), such that $xy^\omega = uv^\omega$, and in addition, the run of the leading automaton $\mathcal{Q}$ on xy^i ends at the same state for every natural number i. Over the normalized pair (x, y), the acceptance condition follows the straightforward approach discussed above. This normalization resembles the implicit flexibility in the acceptance conditions of ω-automata, such as the Büchi condition, and allows saturated FDFAs to be up to exponentially more succinct than $L_\$$-automata [1].

Below, we formally define an FDFA, the normalization of an input pair (u, v), and the acceptance condition. We shall use Σ^{*+} as a shorthand for $\Sigma^* \times \Sigma^+$, whereby the input to an FDFA is a pair $(u, v) \in \Sigma^{*+}$.

▶ **Definition 1** (A family of DFAs (FDFA)). [1]

- A *family of* DFA*s* (FDFA) is a pair $(\mathcal{Q}, \mathbf{P})$, where $\mathcal{Q} = (\Sigma, Q, \iota, \delta)$ is a deterministic *leading* automaton, and $\mathbf{P}$ is a set of $|\mathcal{Q}|$ DFAs, including for each state $q \in Q$, a *progress* DFA $\mathcal{P}_q = (\Sigma, P_q, \iota_q, \delta_q, F_q)$.
- Given a pair $(u, v) \in \Sigma^{*+}$ and an automaton $\mathcal{A}$, the *normalization* of (u, v) w.r.t $\mathcal{A}$ is the pair $(x, y) \in \Sigma^{*+}$, such that $x = uv^i$, $y = v^j$, and $i \geq 0$, $j \geq 1$ are the smallest numbers for which $\mathcal{A}(uv^i) = \mathcal{A}(uv^{i+j})$. (Since we consider complete automata, such a unique pair (x, y) is guaranteed.)
- Let $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$ be an FDFA, $(u, v) \in \Sigma^{*+}$, and $(x, y) \in \Sigma^{*+}$ the normalization of (u, v) w.r.t $\mathcal{Q}$. We say that (u, v) is *accepted* by $\mathcal{F}$ iff $\mathcal{Q}(x) = q$ for some state q of $\mathcal{Q}$ and $\mathcal{P}_q(y)$ is an accepting state of $\mathcal{P}_q$.
- We use $[\![\mathcal{F}]\!]$ to denote the set of pairs accepted by $\mathcal{F}$.
- We define the *size* of $\mathcal{F}$, denoted $|\mathcal{F}|$, as the pair $(|\mathcal{Q}|, \max\{|\mathcal{P}_q|\}_{q \in \mathcal{Q}})$.
- An FDFA $\mathcal{F}$ is saturated if for every two pairs (u, v) and (u', v') such that $uv^\omega = u'v'^\omega$, either both (u, v) and (u', v') are in $[\![\mathcal{F}]\!]$ or both are not in $[\![\mathcal{F}]\!]$.

A saturated FDFA can be used to characterize an ω-regular language (see Theorem 10), while an unsaturated FDFA cannot.

An unsaturated FDFA is depicted in Figure 1 on the left. Consider the pairs (b, a) and (ba, aa). Though $b(a)^\omega = ba(aa)^\omega$, (b, a) is normalized to (b, aa) and $\mathcal{P}_l^U$ accepts aa but (ba, aa) is normalized to itself and $\mathcal{P}_r^U$ rejects aa. A saturated FDFA is depicted in Figure 1 on the right. It accepts pairs of the forms (Σ^*, a^+) and (Σ^*, b^+), and characterizes the ω-regular language $(a + b)^*(a^\omega + b^\omega)$.

4 Boolean Operations and Decision Procedures

We provide below algorithms for performing the Boolean operations of complementation, union, and intersection on saturated FDFAs, and deciding the basic questions on them, such as emptiness, universality, and language containment. All of these algorithms can be done in

[1] The FDFAs defined here follow the definition in [1], which is a little different from the definition of FDFAs in [9]; the latter provide a machine model for the families of right congruences introduced in [10]. The main differences between the two definitions are: i) In [9], a pair (u, v) is accepted by an FDFA $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$ if there is some factorization (x, y) of (u, v), such that $\mathcal{Q}(u) = q$ and $\mathcal{P}_q$ accepts v; and ii) in [9], the FDFA $\mathcal{F}$ should also satisfy the constraint that for all words $u \in \Sigma^*$ and $v, v' \in \Sigma^+$, if $\mathcal{P}_{\mathcal{Q}(u)}(v) = \mathcal{P}_{\mathcal{Q}(u)}(v')$ then $\mathcal{Q}(uv) = \mathcal{Q}(uv')$.

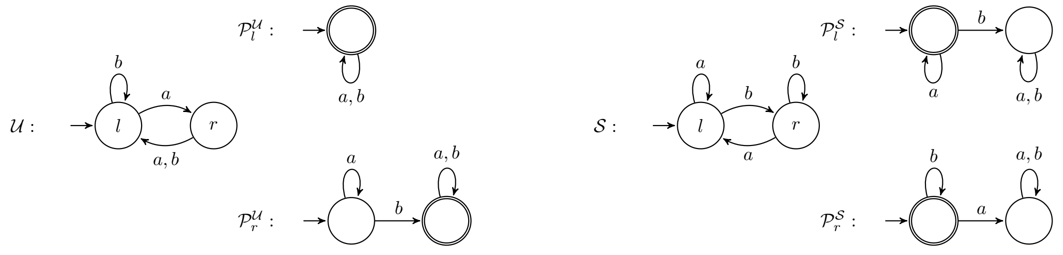

$\mathcal{U}$:
$\mathcal{P}_l^{\mathcal{U}}$:
$\mathcal{P}_r^{\mathcal{U}}$:
$\mathcal{S}$:
$\mathcal{P}_l^{\mathcal{S}}$:
$\mathcal{P}_r^{\mathcal{S}}$:
l
r
a
b
a, b

Figure 1 Left: an unsaturated FDFA with the leading automaton $\mathcal{U}$ and progress DFAs $\mathcal{P}_l^{\mathcal{U}}$ and $\mathcal{P}_r^{\mathcal{U}}$. Right: a saturated FDFA with the leading automaton $\mathcal{S}$ and progress DFAs $\mathcal{P}_l^{\mathcal{S}}$ and $\mathcal{P}_r^{\mathcal{S}}$.

nondeterministic logarithmic space, taking advantage of the partial deterministic nature of FDFAs.[2] We conclude the section with the decision problem of whether an arbitrary FDFA is saturated, showing that it can be resolved in polynomial space.

Boolean operations

Saturated FDFAs are closed under Boolean operations as a consequence of Theorem 10, which shows that they characterize exactly the set of ω-regular languages. We provide below explicit algorithms for these operations, showing that they can be done effectively.

Complementation of an FDFA is simply done by switching between accepting and non-accepting states in the progress automata, as is done with DFAs.

▶ **Theorem 2.** *Let $\mathcal{F}$ be an* FDFA. *There is a constant-space algorithm to obtain an* FDFA *$\mathcal{F}^c$, such that $[\![\mathcal{F}^c]\!] = \Sigma^{*+} \setminus [\![\mathcal{F}]\!]$, $|\mathcal{F}^c| = |\mathcal{F}|$, and $\mathcal{F}^c$ is saturated iff $\mathcal{F}$ is.*

Union and intersection of saturated FDFAs also resemble the case of DFAs, and are done by taking the product of the leading automata and each pair of progress automata. Yet, the correctness proof is a bit subtle, and relies on the following lemma, which shows that for a normalized pair (x, y), the period-word y can be manipulated in a certain way, while retaining normalization.

▶ **Lemma 3.** *Let $\mathcal{Q}$ be an automaton, and let (x, y) be the normalization of some $(u, v) \in \Sigma^{*+}$ w.r.t. $\mathcal{Q}$. Then for every $i \geq 0$, $j \geq 1$ and finite words y', y'' such that $y = y'y''$, we have that $(xy^iy', (y''y')^j)$ is the normalization of itself w.r.t. Q.*

▶ **Theorem 4.** *Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be saturated* FDFA*s of size (n_1, k_1) and (n_2, k_2), respectively. There exist logarithmic-space algorithms to obtain saturated* FDFA*s $\mathcal{H}$ and $\mathcal{H}'$ of size $(n_1n_2,\ k_1k_2)$, such that $[\![\mathcal{H}]\!] = [\![\mathcal{F}_1]\!] \cap [\![\mathcal{F}_2]\!]$ and $[\![\mathcal{H}']\!] = [\![\mathcal{F}_1]\!] \cup [\![\mathcal{F}_2]\!]$.*

Decision procedures

All of the basic decision problems can be resolved in nondeterministic logarithmic space, using the Boolean operations above and corresponding decision algorithms for DFAs.

The first decision question to consider is that of *membership*: given a pair (u, v) and an FDFA $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$, does $\mathcal{F}$ accept (u, v)? The question is answered by normalizing (u, v)

[2] Another model that lies in between deterministic and nondeterministic automata are "semi-deterministic Büchi automata" [25], which are Büchi automata that are deterministic in the limit: from every accepting state onward, their behaviour is deterministic. Yet, as opposed to FDFAs, complementation of semi-deterministic Büchi automata might involve an exponential state blowup [2].

into a pair (x, y) and evaluating the runs of $\mathcal{Q}$ over x and of $\mathcal{P}_{\mathcal{Q}(x)}$ over y. A normalized pair is determined by traversing along $\mathcal{Q}$, making up to $|Q|$ repetitions of v. Notice that memory wise, x and y only require a logarithmic amount of space, as they are of the form $x = uv^i$ and $y = v^j$, where the representation of i and j is bounded by $\log |Q|$. The overall logarithmic-space solution follows from the complexity of algorithms for deterministically traversing along an automaton.

▶ **Proposition 5.** *Given a pair $(u, v) \in \Sigma^{*+}$ and an* FDFA *$\mathcal{F}$ of size (n, k), the membership question, of whether $(u, v) \in [\![\mathcal{F}]\!]$, can be resolved in deterministic space of $O(\log n + \log k)$.*

The next questions to consider are those of *emptiness* and *universality*, namely given an FDFA $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$, whether $[\![\mathcal{F}]\!] = \emptyset$, and whether $[\![\mathcal{F}]\!] = \Sigma^{*+}$, respectively. Notice that the universality problem is equivalent to the emptiness problem over the complement of $\mathcal{F}$. For nondeterministic automata, the complement automaton might be exponentially larger than the original one, making the universality problem much harder than the emptiness problem. Luckily, FDFA complementation is done in constant space, as is the case with deterministic automata, making the emptiness and universality problems equally easy.

The emptiness problem for an FDFA $(\mathcal{Q}, \mathbf{P})$ cannot be resolved by only checking whether there is a nonempty progress automaton in $\mathbf{P}$, since it might be that the accepted period v is not part of any normalized pair. Yet, the existence of a prefix-word x and a period-word y, such that $\mathcal{Q}(x) = \mathcal{Q}(xy)$ and $\mathcal{P}_{\mathcal{Q}(x)}$ accepts y is a sufficient and necessary criterion for the nonemptiness of $\mathcal{F}$. This can be tested in NLOGSPACE. Hardness in NLOGSPACE follows by a reduction from graph reachability [8].

▶ **Theorem 6.** *Emptiness and universality for* FDFA*s are NLOGSPACE-complete.*

The complexity for *equality* and *containment* is easily derived from that of emptiness, intersection and complementation.

▶ **Proposition 7.** *Equality and containment for saturated* FDFA*s are NLOGSPACE-complete.*

Saturation check

All of the operations and decision problems above assumed that the given FDFAs are saturated. This is indeed the case when learning FDFAs via the algorithm of [1], and when translating ω-automata to FDFAs (see Section 5). We show below that the decision problem of whether an arbitrary FDFA is saturated is in PSPACE. We leave the question of whether it is PSPACE-complete open.

▶ **Theorem 8.** *The problem of deciding whether a given* FDFA *is saturated is in PSPACE.*

Proof Sketch. We first show that if an FDFA $\mathcal{F}$ of size (n, k) is unsaturated then there exist words u, v', v'' such that $|u| \leq n$ and $|v'|, |v''| \leq n^n k^{2k}$, and non-negative integers $l, r \leq k$ such that $(u, (v'v'')^l) \in \mathcal{F}$ while $(uv', (v''v')^r) \notin \mathcal{F}$.

Let $\mathcal{Q}$, $\mathcal{P}$, and $\mathcal{P}'$ be the leading automaton and two relevant progress automata, with state spaces Q, P and P', respectively. We achieve the bound on the length of v' and v'', by considering for every word $v \in \Sigma^*$, the function χ_v from (Q, P, P') to (Q, P, P') defined as $\chi_v(q, p, p') = (\delta_{\mathcal{Q}}(q, v), \delta_{\mathcal{P}}(p, v), \delta_{\mathcal{P}'}(p', v))$. Note that there are up to $n^n k^{2k}$ different such functions. Hence, if v' and v'' are longer than $n^n k^{2k}$, we can replace them with shorter words that are completely equivalent w.r.t. $\mathcal{Q}$, $\mathcal{P}$, and $\mathcal{P}'$.

A coNPSPACE algorithm guesses integers $l, r \leq k$ and, letter by letter, some words u, v', v'' such that $|u| \leq n$ and $|v'|, |v''| \leq n^n k^{2k}$. Along the way, it constructs $\chi_{v'v''}$ and $\chi_{v''v'}$.

It then verifies whether one of $(\chi_{v'v''})^l$ and $(\chi_{v''v'})^r$, applied to the relevant initial states, is accepting and the other is not. By Savitch's and Immerman–Szelepcsényi's theorems, the problem is in PSPACE. ◀

5 Translating To and From ω-Automata

As two ω-regular languages are equivalent iff they agree on the set of ultimately periodic words [11], an ω-regular language can be characterized by a language of pairs of finite words, and in particular by a saturated FDFA. We shall write $L \equiv L'$ to denote that a language $L \subseteq \Sigma^{*+}$ characterizes an ω-regular language L'. Formally:

▶ **Definition 9.** A language $L \subseteq \Sigma^{*+}$ *characterizes* an ω-regular language $L' \subseteq \Sigma^\omega$, denoted by $L \equiv L'$, if for every pair $(u, v) \in L$, we have $uv^\omega \in L'$, and for every ultimately periodic word $uv^\omega \in L'$, we have $(u, v) \in L$.

The families of DFAs defined in [9], as well as the analogous families of right congruences of [10], are known to characterize exactly the set of ω-regular languages [9, 10]. This is also the case with our definition of saturated FDFAs.

▶ **Theorem 10.** *Every saturated* FDFA *characterizes an ω-regular language, and for every ω-regular language, there is a saturated* FDFA *characterizing it.*

Proof. The two directions are proved in Theorems 12 and 16, below. ◀

5.1 From ω-Automata to FDFAs

We show that DBA, DCA, and DPA have polynomial translations to saturated FDFAs, whereas translation of NBAs to FDFAs may involve an inevitable exponential blowup.

From deterministic ω-automata. The constructions of a saturated FDFA that characterize a given DBA, DCA, or DPA $\mathcal{D}$ share the same idea: The leading automaton is equivalent to $\mathcal{D}$, except for ignoring the acceptance condition, and each progress automaton consists of several copies of $\mathcal{D}$, memorizing the acceptance level of the period-word. For a DBA or a DCA, two such copies are enough, memorizing whether or not a Büchi (co-Büchi) accepting state was visited. For a DPA with k colors, k such copies are required.

▶ **Theorem 11.** *Let $\mathcal{D}$ be a* DBA *or a* DCA *with n states. There exists a saturated* FDFA *$\mathcal{F}$ of size $(n, 2n)$, such that $[\![\mathcal{F}]\!] \equiv [\![\mathcal{D}]\!]$.*

▶ **Theorem 12.** *Let $\mathcal{D}$ be a* DPA *with n states and k colors. There exists a saturated* FDFA *$\mathcal{F}$ of size (n, kn), such that $[\![\mathcal{F}]\!] \equiv [\![\mathcal{D}]\!]$.*

From nondeterministic ω-automata. An NBA $\mathcal{A}$ can be translated into a saturated FDFA $\mathcal{F}$, by first determinizing $\mathcal{A}$ into an equivalent DPA $\mathcal{A}'$ [18, 7] (which might involve a $2^{O(n \log n)}$ state blowup and $O(n)$ colors [23]), and then polynomially translating $\mathcal{A}'$ into an equivalent FDFA (Theorem 12).

▶ **Proposition 13.** *Let $\mathcal{B}$ be an* NBA *with n states. There is a saturated* FDFA *that characterizes $[\![\mathcal{B}]\!]$ with a leading automaton and progress automata of at most $2^{O(n \log n)}$ states each.*

A $2^{O(n \log n)}$ state blowup in this case is inevitable, based on the lower bound for complementing NBAs [12, 26, 22], the constant complementation of FDFAs, and the polynomial translation of a saturated FDFA to an NBA:

▶ **Theorem 14.** *There exists a family of* NBA*s* $\mathcal{B}_1, \mathcal{B}_2, \ldots$, *such that for every* $n \in \mathbb{N}$, $\mathcal{B}_n$ *is of size* n, *while a saturated* FDFA *that characterizes* $[\![\mathcal{B}_n]\!]$ *must be of size* (m, k), *such that* $\max(m, k) \geq 2^{\Omega(n \log n)}$.

Proof. Michel [12] has shown that there exists a family of languages $\{L_n\}_{n \geq 1}$, such that for every n, there exists an NBA of size n for L_n, but an NBA for L_n^c, the complement of L_n, must have at least $2^{n \log n}$ states.

Assume, towards a contradiction, that exist $n \in \mathbb{N}$ and a saturated FDFA $\mathcal{F}$ of size (m, k) that characterizes L_n, such that $\max(m, k) < 2^{\Omega(n \log n)}$. Then, by Theorem 2, there is a saturated FDFA $\mathcal{F}^c$ of size (m, k) that characterizes L_n^c. Thus, by Theorem 16, we have an NBA of size smaller than $(2^{\Omega(n \log n)})^5 = 2^{\Omega(n \log n)}$ for L_n^c. Contradiction. ◀

5.2 From FDFAs to ω-Automata

We show that saturated FDFAs can be polynomially translated into NBAs, yet translations of saturated FDFAs to DPAs may involve an inevitable exponential blowup.

To nondeterministic ω-automata. We show below that every saturated FDFA can be polynomially translated to an equivalent NBA. Since an NBA can be viewed as a special case of an NPA, a translation of saturated FDFAs to NPAs follows. Translating saturated FDFAs to NCAs is not always possible, as the latter are not expressive enough.

The translation goes along the lines of the construction given in [4] for translating an $L_\$$-automaton into an equivalent NBA. We prove below that the construction is correct for saturated FDFAs, despite the fact that saturated FDFAs can be exponentially smaller than $L_\$$-automata.

We start with a lemma from [4], which will serve us for one direction of the proof.

▶ **Lemma 15** ([4]). *Let* $M, N \subseteq \Sigma^*$ *such that* $M \cdot N^* = M$ *and* $N^+ = N$. *Then for every ultimately periodic word* $w \in \Sigma^\omega$ *we have that* $w \in M \cdot N^\omega$ *iff there exist words* $u \in M$ *and* $v \in N$ *such that* $uv^\omega = w$.

We continue with the translation and its correctness.

▶ **Theorem 16.** *For every saturated* FDFA $\mathcal{F}$ *of size* (n, k), *there exists an* NBA $\mathcal{B}$ *with* $O(n^2 k^3)$ *states, such that* $[\![\mathcal{F}]\!] \equiv [\![\mathcal{B}]\!]$.

Proof. *Construction:* Consider a saturated FDFA $\mathcal{F} = (\mathcal{Q}, \mathbf{P})$, where $\mathcal{Q} = \langle \Sigma, Q, \iota, \delta \rangle$, and for each state $q \in Q$, $\mathbf{P}$ has the progress DFA $\mathcal{P}_q = \langle \Sigma, P_q, \iota_q, \delta_q, F_q \rangle$.

For every $q \in Q$, let M_q be the language of finite words on which $\mathcal{Q}$ reaches q, namely $M_q = \{u \in \Sigma^* \mid \mathcal{Q}(u) = q\}$. For every $q \in Q$ and for every accepting state $f \in F_q$, let $N_{q,f}$ be the language of finite words on which $\mathcal{Q}$ makes a self-loop on q, $\mathcal{P}_q$ reaches f, and $\mathcal{P}_q$ makes a self-loop on f, namely $N_{q,f} = \{v \in \Sigma^* \mid (\delta(q, v) = q) \wedge (\mathcal{P}_q(v) = f) \wedge (\delta_q(f, v) = f)\}$. We define the ω-regular language

$$L = \bigcup_{\{(q,f) \mid (q \in Q) \wedge (f \in F_q)\}} M_q \cdot N_{q,f}^\omega \tag{1}$$

One can construct an NBA $\mathcal{B}$ that recognizes L and has up to $O(n^2 k^3)$ states.

Correctness: Consider an ultimately periodic word $uv^\omega \in [\![\mathcal{B}]\!]$. By the construction of $\mathcal{B}$, $uv^\omega \in L$, where L is defined by Equation (1). Hence, $uv^\omega \in M_q \cdot N^\omega_{q,f}$, for some $q \in Q$ and $f \in F_q$. By the definitions of M_q and $N_{q,f}$, we get that M_q and $N_{q,f}$ satisfy the hypothesis of Lemma 15, namely $N^+_{q,f} = N_{q,f}$ and $M_q \cdot N^*_{q,f} = M_q$. Therefore, by Lemma 15, there exist finite words $u' \in M_q$ and $v' \in N_{q,f}$ such that $u'v'^\omega = uv^\omega$. From the definitions of M_q and $N_{q,f}$, it follows that the run of $\mathcal{Q}$ on u' ends in the state q, and $\mathcal{P}_q$ accepts v'. Furthermore, by the definition of $N_{q,f}$, we have $\delta(q, v') = q$, implying that (u', v') is the normalization of itself. Hence, $(u', v') \in [\![\mathcal{F}]\!]$. Since $\mathcal{F}$ is saturated and $u'v'^\omega = uv^\omega$, it follows that $(u, v) \in [\![\mathcal{F}]\!]$, as required.

As for the other direction, consider a pair $(u, v) \in [\![\mathcal{F}]\!]$, and let (x, y) be the normalization of (u, v) w.r.t. $\mathcal{Q}$. We will show that $xy^\omega \in L$, where L is defined by Equation (1), implying that $uv^\omega \in [\![\mathcal{B}]\!]$. Let $q = \mathcal{Q}(x)$, so we have that $\mathcal{P}_q(y)$ reaches some accepting state f of $\mathcal{P}_q$. Note, however, that it still does not guarantee that $y \in N_{q,f}$, since it might be that $\delta_q(f, y) \neq f$.

To prove that $xy^\omega \in L$, we will show that there is a pair $(x, y') \in \Sigma^{*+}$ and an accepting state $f' \in \mathcal{P}_q$, such that $y' = y^t$ for some positive integer t, and $y' \in N_{q,f'}$; namely $\delta(q, y') = q$, $\mathcal{P}_q(y') = f'$, and $\delta_q(f', y') = f'$. Note first that since $\mathcal{F}$ is saturated, it follows that for every positive integer i, $(x, y^i) \in [\![\mathcal{F}]\!]$, as $x(y^i)^\omega = xy^\omega$.

Now, for every positive integer i, $\mathcal{P}_q$ reaches some accepting state f_i when running on y^i. Since $\mathcal{P}_q$ has finitely many states, for a large enough i, $\mathcal{P}_q$ must reach the same accepting state $\hat{f}$ twice when running on y^i. Let h be the smallest positive integer such that $\mathcal{P}_q(y^h) = \hat{f}$, and r the smallest positive integer such that $\delta_q(\hat{f}, y^r) = \hat{f}$. Now, one can verify that choosing t to be an integer that is bigger than or equal to h and is divisible by r guarantees that $\delta(q, y^t) = q$ and $\delta_q(f', y^t) = f'$, where $f' = \mathcal{P}_q(y^t)$. ◀

To deterministic ω-automata. Deterministic Büchi and co-Büchi automata are not expressive enough for recognizing every ω-regular language. We thus consider the translation of saturated FDFAs to deterministic parity automata. A translation is possible by first polynomially translating the FDFA into an NBA (Theorem 16) and then determinizing the latter into a DPA (which might involve a $2^{O(n \log n)}$ state blowup [12]).

▶ **Proposition 17.** *Let $\mathcal{F}$ be a saturated* FDFA *of size (n, k). There exists a* DPA *$\mathcal{D}$ of size $2^{O(n^2k^3 \log n^2k^3)}$, such that $\mathcal{F} \equiv \mathcal{D}$.*

We show below that an exponential state blowup is inevitable. The family of languages $\{L_n\}_{n\geq 1}$ below demonstrates the inherent gap between FDFAs and DPAs; an FDFA for L_n may only "remember" the smallest and biggest read numbers among $\{1, 2, ..., n\}$, using n^2 states, while a DPA for it must have at least 2^{n-1} states.

We define the family of languages $\{L_n\}_{n\geq 1}$ as follows. The alphabet of L_n is $\{1, 2, ..., n\}$, and a word belongs to it iff the following two conditions are met:

- A letter i is always followed by a letter j, such that $j \leq i + 1$. For example, $533245\ldots$ is a bad prefix, since 2 was followed by 4, while $55234122\ldots$ is a good prefix.
- The number of letters that appear infinitely often is odd. For example, $2331(22343233)^\omega$ is in L_n, while $1(233)^\omega$ is not.

We show below how to construct, for every $n \geq 1$, a saturated FDFA of size polynomial in n that characterizes L_n.

▶ **Lemma 18.** *Let $n \geq 1$. There is a saturated* FDFA *of size $(n + 1, n^2)$ characterizing L_n.*

Proof Sketch. The leading automaton handles the safety condition of L_n, having $n+1$ states, and ensuring that a letter i is always followed by a letter j, such that $j \leq i+1$. The progress automata, which are identical, maintain the smallest and biggest number-letters that appeared, denoted by s and b, respectively. Since a number-letter i cannot be followed by a number-letter j, such that $j > i+1$, it follows that the total number of letters that appeared is equal to $b-s+1$. Then, a state is accepting iff $b-s+1$ is odd. ◀

A DPA for L_n cannot just remember the smallest and largest letters that were read, as these letters might not appear infinitely often. Furthermore, we prove below that the DPA must be of size exponential in n, by showing that its state space must be doubled when moving from L_n to L_{n+1}.

▶ **Lemma 19.** *Every* DPA *that recognizes L_n must have at least 2^{n-1} states.*

Proof. The basic idea behind the proof is that the DPA cannot mix between 2 cycles of n different letters each. This is because a mixed cycle in a parity automaton is accepting/rejecting if its two sub-cycles are, while according to the definition of L_n, the mixed cycle should reject if both its sub-cycles accept, and vice versa. Hence, whenever adding a letter, the state space must be doubled.

In the formal proof below, we dub a reachable state from which the automaton can accept some word a *live state*. Consider a DPA $\mathcal{D}_n$ that recognizes L_n, and let q be some live state of $\mathcal{D}_n$. Observe that $[\![\mathcal{D}_n^q]\!]$, namely the language of the automaton that we get from $\mathcal{D}_n$ by changing the initial state to q, is the same as L_n except for having some restriction on the word prefixes. More formally, if a word $w \in [\![\mathcal{D}_n^q]\!]$ then $w \in L_n$, and if $w \in L_n$ then there is a finite word u, such that $uw \in [\![\mathcal{D}_n^q]\!]$. For every $n \in \mathbb{N}$, and every $u \in \Sigma^*$, let $L_{n,u} = \{w \mid uw \in L_n\}$ and let $\mathfrak{L}_n$ denote the set of languages $\{L_{n,u} \mid u \in \Sigma^*\}$. Note that there is actually only a finite number of prefixes u to consider (this follows e.g. from [10, Thm. 22]). Moreover, for every state q of $\mathcal{D}_n$ there is a corresponding word u_q such that $[\![\mathcal{D}_n^q]\!] = L_{n,u_q}$.

We prove by induction on n the following claim, from which the statement of the lemma immediately follows: Let $\mathcal{D}_n$ be a DPA over $\Sigma = \{1, 2, \ldots, n\}$ that recognizes some language in $\mathfrak{L}_n$. Then there are finite words $u, v \in \Sigma^*$, such that:

(i) v contains all the letters in Σ;
(ii) the run of $\mathcal{D}_n$ on u reaches some live state p; and
(iii) the run of $\mathcal{D}_n$ on v from p returns to p, while visiting at least 2^{n-1} different states.

The base cases, for $n \in \{1, 2\}$, are trivial, as they mean a cycle of size at least 1 over v, for $n = 1$, and a cycle of size 2 for $n = 2$.

In the induction step, for $n \geq 2$, we consider a DPA $\mathcal{D}_{n+1}$ that recognizes some language $L \in \mathfrak{L}_{n+1}$. We shall define $\mathcal{D}'$ and $\mathcal{D}''$ to be the DPAs that result from $\mathcal{D}_{n+1}$ by removing all the transitions over the letter $n+1$ and by removing all the transitions over the letter 1, respectively.

Observe that for every state q that is live w.r.t. $\mathcal{D}_{n+1}$, we have that $[\![\mathcal{D}'^q]\!] \in \mathfrak{L}_n$, namely the language of the DPA that results from $\mathcal{D}_{n+1}$ by removing all the transitions over the letter $n+1$ and setting the initial state to q is in $\mathfrak{L}_n$. (Note that q might only be reachable via the letter $n+1$, yet it must have outgoing transitions over letters in $[2..n]$.) Analogously, $[\![\mathcal{D}''^q]\!]$ is isomorphic to a language in $\mathfrak{L}_n$ via the alphabet mapping of $i \mapsto (i-1)$. Hence, for every state q that is live w.r.t. $\mathcal{D}_{n+1}$, the induction hypothesis holds for $\mathcal{D}'^q$ and $\mathcal{D}''^q$.

We shall prove the induction claim by describing words $u, v \in \Sigma^*$, and showing that they satisfy the requirements above w.r.t. $\mathcal{D}_{n+1}$. We construct u by iteratively concatenating the words u'_i, v'_i, u''_i, and v''_i, which we define below, until the starting and ending states in some iteration k are the same. We then define the word v to be the last iteration, namely $u'_k\, v'_k\, u''_k\, v''_k$. Let q_1 be the initial state of $\mathcal{D}_{n+1}$. We define for every $i \in [1..k]$:

- u'_i and v'_i are the words that follow from the induction hypothesis on $\mathcal{D}'^{q_i}$, where q_i is the state that $\mathcal{D}_{n+1}$ reaches when reading $u'_1\, v'_1\, u''_1\, v''_1 \dots u'_{i-1}\, v'_{i-1}\, u''_{i-1}\, v''_{i-1}$.
- u''_1 and v''_1 are the words that follow from the induction hypothesis on $\mathcal{D}''^{q'_i}$, where q'_i is the state that $\mathcal{D}_{n+1}$ reaches when reading $u'_1\, v'_1\, u''_1\, v''_1 \dots u'_{i-1}\, v'_{i-1}\, u''_{i-1}\, v''_{i-1}\, u'_i\, v'_i$.

The word v obviously contains all the letters in Σ, as it is composed of subwords that contain all the letters in $\Sigma \setminus \{1\}$ and in $\Sigma \setminus \{n+1\}$. By the definition of u and v, we also have that the run of $\mathcal{D}_{n+1}$ on u reaches some live state p, and the run of $\mathcal{D}_{n+1}$ on v from p returns to p. Now, we need to prove that the run of $\mathcal{D}_{n+1}$ on v from p visits at least 2^n states.

We claim that when $\mathcal{D}_{n+1}$ runs on v from p, it visits disjoint set of states when reading v'_k and v''_k. This will provide the required result, as $\mathcal{D}_{n+1}$ visits at least 2^{n-1} states when reading each of v'_k and v''_k.

Assume, by way of contradiction, that $\mathcal{D}_{n+1}$ visits some state s both when reading v'_k and when reading v''_i. Let l' and r' be the parts of v'_k that $\mathcal{D}_{n+1}$ reads before and after reaching s, respectively, and l'' and r'' the analogous parts of v''_k. Now, define the infinite words $m' = u\, u'_k\, (l'\, r')^\omega$, $m'' = u\, u'_k\, l'\, (r''\, l'')^\omega$, and $m = u\, u'_k\, (l'\, r''\, l''\, r')^\omega$.

Observe that m' and m'' both belong or both do not belong to L, since there is the same number of letters (n) that appear infinitely often in each of them. The word m, on the other hand, belongs to L if m' and m'' do not belong to L, and vice versa, since $n+1$ letters appear infinitely often in it. However, the set of states that are visited infinitely often in the run of $\mathcal{D}_{n+1}$ on m is the union of the sets of states that appear infinitely often in the runs of $\mathcal{D}_{n+1}$ on m' and m''. Thus, if $\mathcal{D}_{n+1}$ accepts both m' and m'' it also accepts m, and if it rejects both m' and m'' it rejects m. (This follows from the fact that the minimal number in a union of two sets is even/odd iff the minimum within both sets is even/odd.) Contradiction. ◀

▶ **Theorem 20.** [3] *There is a family of languages $\{L_n\}_{n\geq 1}$ over the alphabet $\{1, 2, \dots, n\}$, such that for every $n \geq 1$, there is a saturated* FDFA *of size $(n+1, n^2)$ that characterizes L_n, while a* DPA *for L_n must be of size at least 2^{n-1}.*

Proof. By Lemmas 18 and 19. ◀

6 Discussion

The interest in FDFAs as a representation for ω-regular languages stems from the fact that they possess a correlation between the automaton states and the language right congruences, a property that traditional ω-automata lack. This property is beneficial in the context of

[3] A small adaptation to the proof of Lemma 19 shows an inevitable exponential blowup also when translating a saturated FDFA to a deterministic ω-automaton with a stronger acceptance condition of Rabin [19] or Streett [24]: A mixed cycle in a Rabin automaton is rejecting if its two sub-cycles are, and a mixed cycle in a Streett automaton is accepting if its two sub-cycles are. Hence, the proof of Lemma 19 holds for both Rabin and Streett automata if proceeding in the induction step from an alphabet of size n to an alphabet of size $n+2$, yielding a Rabin/Streett automaton of size at least $2^{\frac{n}{2}}$.

As for translating a saturated FDFA to a deterministic Muller automaton [13], it is known that translating a DPA of size n into a deterministic Muller automaton might require the latter to have an accepting set of size exponential in n [21]. (The family of languages in [21] uses an alphabet of size exponential in the number of states of the DPA, however it can easily be changed to use an alphabet of linear size.) Hence, by Theorem 12, which shows a polynomial translation of DPAs to FDFAs, we get that translating an FDFA to a deterministic Muller automaton entails an accepting set of exponential size, in the worst case.

learning, and indeed an algorithm for learning ω-regular languages by means of saturated FDFAs was recently provided [1]. Analyzing the succinctness of saturated FDFAs and the complexity of their Boolean operations and decision problems, we believe that they provide an interesting formalism for representing ω-regular languages. Indeed, Boolean operations and decision problems can be performed in nondeterministic logarithmic space and their succinctness lies between deterministic and nondeterministic ω-automata.

References

1 D. Angluin and D. Fisman. Learning regular omega languages. In Peter Auer, Alexander Clark, Thomas Zeugmann, and Sandra Zilles, editors, *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, volume 8776 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2014. doi:10.1007/978-3-319-11662-4_10.

2 F. Blahoudek, M. Heizmann, S. Schewe, J. Strejcek, and M.H. Tsai. Complementing semi-deterministic büchi automata. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9636 of *LNCS*, pages 770–787. Springer, 2016.

3 J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

4 H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational *w*-languages. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 554–566, London, UK, 1994. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=645738.666444.

5 M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the language of error. In *13th Int. Symp. on Automated Technology for Verification and Analysis*, pages 114–130, 2015.

6 A. Farzan, Y. Chenand E.M. Clarke, Y. Tsay, and B. Wang. Extending automated compositional verification to the full class of omega-regular languages. In C.R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 2–17. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-78800-3_2.

7 D. Fisman and Y. Lustig. A modular approach for Büchi determinization. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 368–382. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. URL: http://www.dagstuhl.de/dagpub/978-3-939897-91-0, doi:10.4230/LIPIcs.CONCUR.2015.368.

8 N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 1975.

9 N. Klarlund. A homomorphism concept for omega-regularity. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1994. doi:10.1007/BFb0022276.

10 O. Maler and L. Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997. doi:10.1016/S0304-3975(96)00312-X.

11 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966. doi:10.1016/S0019-9958(66)80013-X.

12 M. Michel. Complementation is much more difficult with automata on infinite words. In *Manuscript, CNET*, 1988.

13 D.E. Muller. Infinite sequences and finite machines. In *Proc. 4th IEEE Symp. on Switching Circuit Theory and Logical design*, pages 3–16, 1963.

14 W. Nam and R. Alur. Learning-based symbolic assume-guarantee reasoning with automatic decomposition. In *Automated Technology for Verification and Analysis, 4th International Symposium, ATVA 2006, Beijing, China, October 23-26, 2006.*, pages 170–185, 2006.

15 D. Neider and N. Jansen. Regular model checking using solver technologies and automata learning. In *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, pages 16–31, 2013.

16 A. Nerode. Linear automaton transformations. *Proc. of the American Mathematical Society*, 9(4):541–544, 1858.

17 D. Peled, M. Vardi, and M. Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2002.

18 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 255–264. IEEE Computer Society, 2006. URL: `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=11132`, `doi:10.1109/LICS.2006.28`.

19 M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

20 B. L. Saëc. Saturating right congruences. *ITA*, 24:545–560, 1990.

21 S. Safra. *Complexity of automata on infinite objects.* PhD thesis, Weizmann Institute of Science, 1989.

22 S. Schewe. Büchi complementation made tight. In *Proc. 26th Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 3 of *LIPIcs*, pages 661–672, 2009.

23 S. Schewe. Tighter bounds for the determinization of Büchi automata. In *Proc. 12th Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 167–181, 2009.

24 R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.

25 M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.

26 Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. 33rd Int. Colloq. on Automata, Languages, and Programming (ICALP)*, volume 4052 of *LNCS*, pages 589–600, 2006.

On the Complexity of Probabilistic Trials for Hidden Satisfiability Problems*

Itai Arad[1], Adam Bouland[2], Daniel Grier[3], Miklos Santha[4], Aarthi Sundaram[5], and Shengyu Zhang[6]

1 **Center for Quantum Technologies, National University of Singapore, Singapore**
`arad.itai@fastmail.com`

2 **Massachusetts Institute of Technology, Cambridge, MA USA**
`adam@csail.mit.edu,`

3 **Massachusetts Institute of Technology, Cambridge, MA USA**
`grierd@mit.edu`

4 **Center for Quantum Technologies, National University of Singapore, Singapore and**
CNRS, IRIF, Université Paris Diderot 75205 Paris, France
`miklos.santha@gmail.com`

5 **Center for Quantum Technologies, National University of Singapore, Singapore**
`aarthims@gmail.com`

6 **Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong**
`syzhang@cse.cuhk.edu.hk`

Abstract

What is the minimum amount of information and time needed to solve 2SAT? When the instance is known, it can be solved in polynomial time, but is this also possible without knowing the instance? Bei, Chen and Zhang (STOC '13) considered a model where the input is accessed by proposing possible assignments to a special oracle. This oracle, on encountering some constraint unsatisfied by the proposal, returns only the constraint index. It turns out that, in this model, even 1SAT cannot be solved in polynomial time unless P=NP. Hence, we consider a model in which the input is accessed by proposing probability distributions over assignments to the variables. The oracle then returns the index of the constraint that is most likely to be violated by this distribution. We show that the information obtained this way is sufficient to solve 1SAT in polynomial time, even when the clauses can be repeated. For 2SAT, as long as there are no repeated clauses, in polynomial time we can even learn an equivalent formula for the hidden instance and hence also solve it. Furthermore, we extend these results to the quantum regime. We show that in this setting 1QSAT can be solved in polynomial time up to constant precision, and 2QSAT can be learnt in polynomial time up to inverse polynomial precision.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases computational complexity, satisfiability problems, trial and error, quantum computing, learning theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.12

* Research was supported by the Singapore Ministry of Education and the National Research Foundation by the Tier 3 Grant MOE2012-T3-1-009, by the European Commission IST STREP project Quantum Algorithms (QALGO) 600700, the French ANR Blanc Program Contract ANR-12-BS02-005. A.B. was supported in part by the NSF Graduate Research Fellowship under grant no. 1122374 and by the NSF Alan T. Waterman award under grant no. 1249349. S.Z.'s research was supported in part by RGC of Hong Kong (Project no. CUHK419413).

1 Introduction

SAT has been a pivotal problem in theoretical computer science ever since the advent of the Cook-Levin Theorem [7, 13] proving its NP-completeness. It has a wide array of applications in operations research, artificial intelligence and bioinformatics. Moreover, it continues to be studied under various specialized models such as as random-SAT and building efficient SAT solvers for real-life scenarios. In the complexity theoretic setting, we know that while 3SAT is NP-complete [7, 13], 2SAT can be solved in linear time [12, 9, 3]. Given the fundamental nature of 2SAT, in this paper, we consider the following question:

What is the minimum amount of information needed to solve 2SAT *in polynomial time?*

More precisely, what happens if there is no direct access to the problem instance? Are there settings where one can *solve* 2SAT without ever *learning* the instance under consideration? We can also pose the same question for the quantum setting where the quantum analogue of SAT (QSAT) can be seen as a central problem in condensed matter physics. Complexity theoretically, we know that 2QSAT can be solved in linear time [2, 8] while 3QSAT is hard for QMA_1 [10], where QMA_1 is a quantum complexity class analogous to NP. We approach these questions through the "trial and error" model. In this model, one guesses a solution to an unknown constraint satisfaction problem and tests if it is valid. If so, then the problem is solved. Otherwise, the trial fails, and some information about what was wrong with the trial is revealed. This type of problem arises in a number of natural scenarios, in which one has incomplete or limited information about the problem they are trying to solve [4]. For example, the CSP may be instantiated by a complex biological or physical process to which one does not have access.

This approach to problem solving was first formalized by Bei, Chen and Zhang [4]. They considered several types of CSPs and analyzed the computational complexity in the "unknown input" setting. Specifically, they consider an oracle model where one can propose solutions to the CSP, and if the solution is not satisfying, then the oracle reveals the identity of a constraint which was violated. For example, if the CSP is an instance of Boolean satisfiability (SAT), then after an unsuccessful trial, one may learn that "clause 7 was violated", but not anything further. In particular, literals present in clause 7 will not be revealed - only the label of violated clause is known. Furthermore, if there are several violated constraints, then the oracle reveals only one of them, in a possibly adversarial manner. In this paper, we will refer to this as the "arbitrary violated constraint" oracle.

This model gives extremely limited access to the instance. In fact, Ivanyos et al. [11] showed that even if the underlying CSP is a 1SAT instance, accessing it with the BCZ oracle, one cannot determine if it is satisfiable in polynomial time unless $\text{P} = \text{NP}$. This drastically increases the difficulty of deciding a trivial problem like 1SAT (assuming $\text{P} \neq \text{NP}$). Interestingly, if there is access to a SAT solver, then 1SAT (and even generic SAT) in this setting can be solved with polynomially many trials [4]. So in some sense, their model reveals a sufficient amount of *information* to solve the 1SAT instance. However, *decoding* this information requires superpolynomial time (assuming $\text{P} \neq \text{NP}$). In short the information needed to solve the problem is present, but it is not accessible to poly-time algorithms.

In this paper, we ask if there are any meaningful modifications of their model which allow us to solve simple CSPs like 1SAT and 2SAT in polynomial time. A natural starting point is to randomize the "arbitrary violated constraints" model. One obvious way to do that is to consider allowing randomized queries to the oracle. This however does not significantly decrease the complexity of the problems. A second approach to randomize is to let the oracle return a violated clause at random. Contrary to the previous approach, this model trivializes the problem, since by repeating the same trial many times the oracle will reveal all violated clause indices with high probability. This in turn allows one to learn the entire instance, and therefore trivially, to solve 1SAT and 2SAT.

Motivated by these unfruitful approaches we consider a model which does not allow one to completely learn the underlying instance, but it still yields polynomial time algorithms for 1SAT and 2SAT. Specifically, in this model one can propose a probability distribution D over assignments, and the oracle reveals the index of the clause which is most likely to be violated by this trial. If there are multiple clauses with the same probability of violation under D, then the oracle can break ties arbitrarily. In particular, product distributions over the variables suffice for our application, so one merely specifies the probability p_i that each variable x_i is set to 1 in the assignment, to $1/\operatorname{poly}$ precision. We show that in this model, there exist cases where one cannot learn the underlying 1SAT or 2SAT instance. However, despite this limitation, one can still solve in polynomial time 1SAT and a restricted version of 2SAT where clauses are not repeated. In the course of the algorithm for the restricted version of 2SAT, we actually learn an equivalent formula with the same set of satisfying assignments. Furthermore, we are able extend this model to the quantum setting, and show that one can solve, in polynomial time, Quantum 1SAT (1QSAT) up to constant precision. We also show that in polynomial time we can learn Quantum 2SAT (2QSAT) up to inverse polynomial precision. This, however, seems insufficient to solve the hidden instance in polynomial time due to some subtle precision issues, which we discuss in Section 7.

Relation to prior work. As previously mentioned, Bei Chen and Zhang [4] introduced the trial and error model. They considered several examples of CSPs and analyzed their complexity under the unknown input model with the "arbitrary violated constraint" oracle. With regards to SAT, they showed an algorithm to solve hidden-SAT using polynomially many queries to the oracle (given access to a SAT oracle). Furthermore, they showed that one cannot efficiently learn generic SAT instances in this model, because it takes $\Omega(2^n)$ queries to the oracle to learn a clause involving all n variables of the instance.

Subsequently, Ivanyos *et al.* [11] characterized the complexity of classical CSPs in several hidden input models. In particular, they consider the "arbitrary violated constraint" model described above, as well as models which reveal more information such as the variables involved in the violated clause or the relation of the violated clause. They show a generic "transfer theorem" which classifies the complexity of hidden versions of CSPs given properties of the base CSP. In particular, their transfer theorem implies that the hidden version of 1SAT with arbitrary violated constraints cannot be solved in polynomial time unless P = NP. This indicates that the "arbitrary violated constraint" model is fairly restrictive.

In parallel, Bei, Chen and Zhang [5] considered a version of the trial and error model for linear programming. Suppose you have a linear program, and you are trying to determine whether or not it is feasible (By standard reductions this is as difficult as solving a generic LP). They consider a model in which one can propose a point, and the oracle will return the index of an arbitrary violated constraint (half-plane) in the linear program. They show that in this model, one requires exponentially many queries to the oracle to determine if an LP is feasible. However, they then consider a relaxation of this model, in which the oracle returns the index of the *worst-violated* constraint, i.e. the half-plane which is furthest (in Euclidean distance) from the proposed point. Surprisingly, they show (using a variant of the ellipsoid algorithm) that one can still solve linear programs in this model in polynomial time. Our model can be seen as an analogue of the "worst violated constraint" model of Bei, Chen and Zhang [5] for the case of hidden SAT (H–SAT).

Our Results. Our results can be broken into several sections. First, we consider a relaxation of the "arbitrary violated constraint" model of Bei, Chen and Zhang [4], in which the oracle reveals which subset of clauses are violated by each assignment[1]. We show that in some sense these models are almost "too easy"

▶ **Theorem** (Informal statement). *In the "all violated constraints" model, there is an algorithm which either learns an arbitrary* kSAT *instance on n variables and m clauses, or else finds a satisfying assignment to the instance, in time $O(mn^k)$.*

We then explore the "worst violated constraint" model for the rest of the paper. We provide an example for why this model is more powerful than the "arbitrary violated constraint" model of Bei, Chen and Zhang [4]. They showed that it requires $\Omega(2^n)$ time to learn a SAT clause involving all n variables. Our example states that

▶ **Proposition.** *Given a hidden* WIDESAT *instance on n variables and m distinct clauses where $m \leq n$, we can learn an equivalent instance in $O(\binom{n}{m-1}2^m + n)$ time.*

The proofs of the above results are omitted owing to space constraints[2]. Among our main results is the analysis of the computational complexity of H–1SAT and that of H–2SAT.

▶ **Theorem** (Informal statement). *Given a hidden* SAT *formula Φ on n variables and n clauses, it is possible to find a satisfying assignment for Φ in polynomial time if Φ is a*
(a) 1SAT *formula or*
(b) 2SAT *formula with no repeated clauses.*

Our algorithm for H–1SAT, in Section 3, works even when clauses are repeated multiple times in the instance, despite the fact that it's not possible to *learn* the instance in this setting. This is in sharp contrast to the "arbitrary violated constraint" model, where even H–1SAT cannot be solved in polynomial time unless P = NP [11]. The main difficulty in deriving our algorithm for H–1SAT comes from dealing with repeated clauses, which allow the oracle to obscure information about the instance. Unlike the H–1SAT case, the algorithm for H–2SAT discussed in Section 4, works by attempting to learn the instance; it either succeeds in learning an equivalent instance (in which case one can solve the problem using any 2SAT algorithm), or it accidentally stumbles upon a satisfying assignment in the meantime and aborts. The problem of solving H–2SAT with repeated clauses similar to H–1SAT is left for future work.

Following this we generalize these results to the quantum case. In this case the goal is to determine if a set of 1-qubit or a set of 2-qubit projectors is mutually satisfiable or not. We consider an analogue of this model in which one can propose a probability distribution over quantum states (i.e. a density matrix), and the oracle returns the index of the clause which is most likely to be violated. Our results for hidden QSAT (H–QSAT) show that

▶ **Theorem** (Informal statement). *Given a* H–QSAT *instance H defined on n qubits with m projectors and $\epsilon > 0$, it is possible to*

[1] This is equivalent to a model in which the oracle reveals a random violated clause - by repeating each query many times one can learn the set of violated clauses with high probability.
[2] Omitted details and proofs can be found in the full version of the paper at `http://arxiv.org/abs/1606.03585`.

(a) *solve H to a precision ϵ in time $O(n^{\log(1/\epsilon)})$ if H is a* 1QSAT *instance and*
(b) *learn each projector of H up to precision ϵ in time $O(n^4+n^2\log(1/\epsilon))$, if H is a* 2QSAT *instance as long as the interaction graph of H is not star-like.*

By star-like, we mean the interaction graph contains an edge that is incident to all other edges in the graph. At this point it is worth comparing the notions of *learning* and *solving* hidden instances both in the classical and quantum settings. The classical case is more straightforward where learning an instance means learning all the literals present in each clause, whereas solving means finding a satisfying assignment. For example, our algorithm for H–2SAT without repetitions learns the instance, while our algorithm for H–1SAT solves the instance without learning it. For hidden versions of 1SAT and 2SAT, learning the instance in polynomial time automatically triggers solving it in polynomial time as well.

However, in the quantum setting this simple relation between learning and solving breaks down. The continuous nature of QSAT means we can only learn a projector or find a satisfying assignment up to a specified precision ϵ. The latter is accomplished with our H–1QSAT algorithm in Section 6. However in the case of hidden 2QSAT learning the instance up to precision ϵ does not imply that one can solve the instance up to precision poly(n,ϵ) in polynomial time. This can be attributed to current algorithms for 2QSAT being very sensitive to precision errors. This issue of divergence between the notions of learning and solving H–2QSAT instances is further discussed in Section 7.

2 Notations and Preliminaries

Boolean Satisfiability. The Boolean satisfiability problem, generally referred to as SAT, is a constraint satisfaction problem defined on n variables $\mathbf{x} = \{x_1,\ldots,x_n\}$ where we are given a formula represented as a conjunction of m clauses and each clause is a disjunction of *literals* (variables, x_j, or negated variables, $\overline{x}_j$). The problem is solved if we can find an assignment to the variables (i.e. $\forall\ i,\ x_i \in \{0,1\}$) that sets the value of every clause to 1. In particular, if each clause involves at most k literals, then this problem is classified as kSAT. It is well known that while 2SAT can be solved in linear time [12, 9, 3], kSAT for $k \geq 3$ is NP-complete [7, 13]. A useful notion is that of *clause types* which is defined as the unordered set of literals present in the clause. Specifically, the clause type for $C_j = (x_a \vee \overline{x}_b \vee x_c)$ is denoted by $T(C_j) = \{x_a, \overline{x}_b, x_c\}$. So, all possible clause types for 2SAT would be $\{\{x_a,x_b\},\{x_a,\overline{x}_b\},\{\overline{x}_a,x_b\},\{\overline{x}_a,\overline{x}_b\} \mid a,b\in[n]$ and $a\neq b\}$, where $[n]$ denotes the set $\{1,\ldots,n\}$. From this definition, it is clear that 2SAT has $O(n^2)$ clause types and similarly, kSAT would have $\binom{2n}{k} = O(n^k)$ clause types. . Given a SAT formula ϕ, we say that the SAT formula ϕ' is *equivalent* to ϕ if for all assignments $\mathbf{x}\in\{0,1\}^n$, $\mathbf{x}$ satisfies ϕ if and only if it satisfies ϕ'. For any formula ϕ, SAT$(\phi) := \{\mathbf{x}\in\{0,1\}^n \mid \phi(\mathbf{x}) = 1\}$.

Hidden SAT. While considering the unknown input version of SAT (resp. kSAT), the boolean formula is considered as hidden and accessible only via an oracle that accepts an assignment and reveals some form of violation information. In our case, this is the "worst violated oracle" which accepts a *probabilistic* assignment and reveals a clause that has the *highest probability* of being violated with ties being broken arbitrarily. A probabilistic assignment for a set of n variables is a function $\mathbf{a} : [n] \to [0,1]$ such that $Pr[x_i = 1] = \mathbf{a}(i)$ and $Pr[x_i = 0] = Pr[\overline{x}_i = 1] = 1 - \mathbf{a}(i)$. For the sake of concise notation, these are usually written as $x_i = \mathbf{a}(i)$ and $\overline{x}_i = 1 - \mathbf{a}(i)$. This naturally translates to the notion of the probability of a clause C_j being violated which is defined as $Pr[C_j = 0] := \prod_{\ell\in T(C_j)} Pr[\ell = 0] = \prod_{\ell\in T(C_j)}(1-\ell)$ which allows the oracle to calculate the probability for each clause being violated. Here we are using ℓ to refer both to the identity of a literal as well as to the probability that literal ℓ is set to true. Now, the problem H–SAT (resp. H–kSAT) consists of finding a satisfying assignment for a hidden SAT (resp. kSAT) formula by proposing probabilistic assignments to the "worst violated oracle". One way we do this is also by *learning* an *equivalent formula* to the hidden instance and solve it to find a satisfying assignment. By learning we mean the process of using the information from a series of violations to determine what a clause in the hidden instance could be.

Note that it's possible for an instance to contain clauses which will never be returned by the oracle. For instance, given clauses C_i and C_j, if $T(C_i) \subset T(C_j)$, then clause C_i will always be at least as violated as C_j. Hence the oracle might never return clause C_j. For this reason we will say that C_i *obscures* C_j if $T(C_i) \subset T(C_j)$. An obscured clause might never be returned by the oracle.

The complexity of the algorithms in the following sections is in terms of the total running time where one query to the oracle takes unit time.

3 Hidden 1SAT

In this section, we will consider the problem of a hidden 1SAT instance Φ, possibly with repetitions. Our goal will be to determine whether or not Φ is satisfiable. A natural approach one might take to solve this problem would be to learn the identity of each clause in the instance Φ. Unfortunately, in the case that the 1SAT instance has repetitions, this is not possible.

▶ **Proposition 1.** *There is no algorithm which, given an instance Φ which is unsatisfiable, learns all the literals present in Φ (even granted arbitrary numbers of queries to the oracle).*

Here the difficulty in learning an unsatisfiable instance does not lie in the repetition of clauses, but rather in determining for which i do both x_i and $\overline{x}_i$ appear in Φ. This shows that no algorithm can learn the hidden 1SAT instance [3] (proof omitted owing to space constraints). Hence if there is an algorithm to solve 1SAT in this hidden setting, then it must solve the instance despite the fact that it cannot deduce the underlying instance. Surprisingly, this turns out to be possible.

▶ **Theorem 1.** *Given a hidden* 1SAT *instance Φ on n variables and m clauses, it is possible to determine if Φ is satisfiable in time $O(mn^2)$.*

Proof Consider an ordering of the variables $x_1 ... x_n$. The algorithm will work by inductively constructing a series of lists $L_1, L_2, \dots L_n$. Each list L_i will contain a list of partial assignments to the variables $x_1 \dots x_i$. Each list will be of size at most m, with the exception of L_n which will be of size at most $2m$. Let us call a partial assignment p to $x_1 \dots x_i$ *good* if there exists an assignment p' to the variables $x_{i+1} \dots x_n$ such that the assignment $p \cup p'$ satisfies Φ. Correspondingly, call p *bad* if it cannot be extended to a satisfying assignment of Φ. (Note in the case of 1SAT, every partial assignment is either good or bad.) Our algorithm will guarantee that, if Φ is satisfiable, then at least one assignment in each list is "good". Therefore, by constructing the list L_n, then trying all assignments in L_n, we will be guaranteed to find a satisfying assignment if one exists.

[3] Note, however, it is still possible that there exists an algorithm to learn the 1SAT instance when the instance is promised to be satisfiable.

We now describe how to construct the lists $\{L_i\}_{i\in[n-1]}$ by induction. The base case of L_1 is trivial - just add both $x_1 = 0$ and $x_1 = 1$ to the list. We now show how to construct L_{i+1} given L_i. First, let $\tilde{L}_{i+1}$ be all possible extensions of the assignments in L_i to the variable x_{i+1}. Clearly if one of the assignments in L_i was good, then one of the assignments in $\tilde{L}_{i+1}$ is good. However, when $i + 1 < n$, the size of $\tilde{L}_{i+1}$ could become too large - it is of size $2|L_i|$ which could at some point become larger than m. So we need to reduce the size of $\tilde{L}_{i+1}$ so that it contains at most m partial assignments. To decide which partial assignments to keep, we will perform the following oracle queries: for each partial assignment $y \in \tilde{L}_{i+1}$, propose the following query q_y to the oracle: set $x_1 ... x_{i+1}$ to 0 or 1 according to y, and set all other variables to value 1/2. The oracle will return the identity of a clause C_j which is worst violated by this fractional assignment. Now partition the elements of $\tilde{L}_{i+1}$ according to which clause C_j was returned by the query. This divides the elements of $\tilde{L}_{i+1}$ into at most m equivalence classes. To construct L_{i+1}, simply pick one element from each equivalence class of $\tilde{L}_{i+1}$.

Clearly L_{i+1} has size at most m by construction. To complete the proof, we need to show that at least one element of L_{i+1} is good. First, by the induction hypothesis, at least one element of L_i is good. This implies at least one element $y^* \in \tilde{L}_{i+1}$ is good as well. Consider what happens when we perform the query q_{y^*}. Since y^* is good, q_{y^*} must satisfy all clauses involving the variables $x_1 \dots x_{i+1}$. If there are no clauses involving the remaining variables $x_{i+2} \dots x_n$, then q_{y^*} satisfies the instance, so the oracle will tell us this and we can terminate the algorithm. Otherwise, there is a clause involving some variable in $\{x_{i+2} \dots x_n\}$. When we query q_{y^*}, the worst violated clause will be some clause C_k involving a variable in $\{x_{i+2} \dots x_n\}$, which will be violated with probability 1/2. So the equivalence class corresponding to C_k will contain a good assignment. Furthermore, since C_k involves one of the variables in $\{x_{i+2} \dots x_n\}$, it will never be returned as the worst violated clause for query $q_{y'}$ for any bad assignment $y' \in \tilde{L}_{i+1}$, because any bad assignment will violate a clause involving $\{x_1 \dots x_{i+1}\}$ by 1, while C_k will be violated only with probability 1/2. Therefore the equivalence class corresponding to C_k will contain only good assignments. So by picking one assignment from each equivalence class, we will ensure L_{i+1} contains at least one good assignment, as claimed.

The time to construct each list is $O(mn)$, and the algorithm constructs n lists. Hence the algorithm runs in time $O(mn^2)$. ☐

4 Hidden 2SAT without repetitions

In this section, we consider a hidden 2SAT formula Φ which is promised to contain no two clauses that are the same. Although Proposition 1 shows that we cannot always hope to learn Φ directly, it does not rule out the possibility of learning some Φ' such that $\mathrm{SAT}(\Phi') = \mathrm{SAT}(\Phi)$. In fact, this is exactly the approach we take. The full proof of this is omitted to conserve space, but the most interesting aspect is contained in the theorem below.

▶ **Theorem 2.** *Suppose Φ is a hidden repetition-free* 2SAT *instance on n variables. Then it is possible to generate a satisfying assignment in time $O(n^2)$.*

Proof The idea is to attempt to learn each clause present in the formula. Suppose we wish to determine if the clause $(x_i \vee x_j)$ is present in Φ (an analogous procedure works to determine if a 1SAT clause x_i is in Φ). We can assume that the clause is unobscured because the presence of an obscured clause does not affect the set of satisfying assignments. Run the following procedure:

Table 1 Violation of the clauses based on the fractional assignments of 1/4 and 3/4.

	$(x_i \vee x_j)$	$(x_i \vee x_k)$	$(x_j \vee x_k)$	$(x_{k_1} \vee x_{k_2})$	$(x_{k_1} \vee \bar{x}_{k_2})$	$(x_i \vee \bar{x}_k)$	$(x_j \vee \bar{x}_k)$	$(\bar{x}_{k_1} \vee \bar{x}_{k_2})$
1/4	1	3/4	3/4	9/16	3/16	1/4	1/4	1/16
3/4	1	1/4	1/4	1/16	3/16	3/4	3/4	9/16

1. First query the oracle with the assignment $x_i = 0$, $x_j = 0$, $x_k = 0$ for $k \neq i, j$. If this is a satisfying assignment, then we are done. Otherwise, we know that there must exist a clause of type:
   (a) $(x_i \vee x_j)$;
   (b) $(x_i \vee x_k)$ for $k \neq i, j$;
   (c) $(x_j \vee x_k)$ for $k \neq i, j$; or
   (d) $(x_{k_1} \vee x_{k_2})$ for $k_1, k_2 \neq i, j$.
5. Now query the oracle with the assignment $x_i = 0$, $x_j = 0$, $x_k = 1$ for $k \neq i, j$. As before, if this is satisfying, we are done. Otherwise, we know that there must exist a clause of type:
   (a) $(x_i \vee x_j)$;
   (b) $(x_i \vee \bar{x}_k)$ for $k \neq i, j$;
   (c) $(x_j \vee \bar{x}_k)$ for $k \neq i, j$; or
   (d) $(\bar{x}_{k_1} \vee \bar{x}_{k_2})$ for $k_1, k_2 \neq i, j$.
5. We can now construct an explicit test for the presence of the clause $(x_i \vee x_j)$. We will propose two fractional assignments to the oracle. If $(x_i \vee x_j)$ is present, then the clause returned each time will be the same. If it is not present, then the returned clause will be different. Formally, query the oracle with the assignment $x_i = 0$, $x_j = 0$, $x_k = \frac{1}{4}$ for $k \neq i, j$ and then with the assignment $x_i = 0$, $x_j = 0$, $x_k = \frac{3}{4}$ for $k \neq i, j$. Table 1 shows the accompanying violations.
   It is clear that if $(x_i \vee x_j)$ is present in the formula, then it is returned on both assignments. If it is not present, then from the table we can also see that one of the clauses known to exist from our first query must be returned on the 1/4 fractional assignment. However, one of the clauses known to exist from our second query must be returned on the 3/4 fractional assignment. Thus, the clause returned by the oracle changes when $(x_i \vee x_j)$ is not present.

Notice that the above procedure also works to detect all 1SAT and 2SAT clause types. Therefore, if we complete the above procedure with all $O(n^2)$ clause types without finding a satisfying assignment, then we have identified all unobscured clauses in the formula. It is clear that the conjunction of these clauses forms a formula Φ' such that $\text{SAT}(\Phi') = \text{SAT}(\Phi)$. Therefore, we can use any 2SAT algorithm which runs in time $O(n^2)$ on Φ' to find some satisfying assignment of Φ. ☐

While the above procedure may seem elementary, it acts as a stepping stone to tackle the harder problem of learning an unknown input instance of quantum 2SAT, which is introduced and discussed in the subsequent sections.

5 Quantum SAT Preliminaries

Notations. A quantum system of n qubits is described using a Hilbert space $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \ldots \otimes \mathcal{H}_n$ where each $\mathcal{H}_i$ is a two-dimensional Hilbert space of the i^{th} qubit. Vectors in $\mathcal{H}$ are called *pure states* and they describe a state of the system. By adding a subscript i to the vector $|\alpha\rangle$ we indicate that $|\alpha\rangle_i$ is defined in the local Hilbert space $\mathcal{H}_i$ of the i^{th}

qubit. Similarly, $|\psi\rangle_{ij}$ denotes a 2-qubit state $|\psi\rangle$ in $\mathcal{H}_i \otimes \mathcal{H}_j$. In any local qubit space $\mathcal{H}_i$, we pick an orthonormal basis $|0\rangle, |1\rangle$ so that every 1-qubit state $|\alpha\rangle$ can be expanded as $|\alpha\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$. We define its orthogonal state by $|\alpha^\perp\rangle := \alpha_1|0\rangle - \alpha_0|1\rangle$;[4] clearly, $\langle\alpha|\alpha^\perp\rangle = 0$. A standard geometrical representation of the state space of a single qubit is the Bloch sphere. The interested reader is referred to [14] for details on the exact correspondence between quantum states and points on the Bloch Sphere.

A more general way to describe a quantum state is by its *density matrix.* Density matrices can be viewed as statistical ensembles of pure states that are described by vectors. A density matrix representation a single pure state $|\psi\rangle$ is given by the matrix $\rho = |\psi\rangle\langle\psi|$. General density matrices are given as a convex sum of density matrices of the pure states with the coefficient summing up to 1: $\sigma = \sum_i p_i|\psi_i\rangle\langle\psi_i|$ where $\forall i, p_i \geq 0$ and $\sum_i p_i = 1$. Alternatively, they are defined as semi-definite operators whose trace is equal to 1. For instance, the density matrix $\frac{1}{2}\mathbb{I}$ can be written as $\frac{1}{2}\mathbb{I} = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|$. The state of a quantum system can always be fully specified by a density matrix.

Observables in quantum mechanics are associated with Hermitian operators. The eigenvalues of such an operator correspond to the possible outcomes of a measurement. Given such a Hermitian operator A and a pure state $|\psi\rangle$, the expression $\langle\psi|A|\psi\rangle$ is the *expectation value* of A. It is the result we get if we measure A over many copies of the same state $|\psi\rangle$ and average the result. One can use the Chernoff bound to deduce that, with high probability, if we measure A over $\mathrm{poly}(n)$ copies of a state $|\psi\rangle$, we obtain an approximation to $\langle\psi|A|\psi\rangle$ with an additive error of $1/\mathrm{poly}(n)$.

The expectation value of A with respect to a state which is described by a density matrix ρ is given as $\mathrm{Tr}(\rho A)$. Note that if ρ is given by $\rho = \sum_i p_i|\psi_i\rangle\langle\psi_i|$ with $\sum_i p_i = 1$, then $\mathrm{Tr}(\rho A) = \sum_i p_i\langle\psi_i|A|\psi_i\rangle$, which justifies the interpretation of ρ as a statistical ensemble of pure states. Like in the pure state case, using $\mathrm{poly}(n)$ identical copies of ρ, one can estimate the expectation value $\mathrm{Tr}(\rho A)$ up to an additive error of $1/\mathrm{poly}(n)$.

Local Hamiltonians and Quantum SAT. While classically SAT is given as a CSP, quantum kSAT (kQSAT) is defined as a special case of the k-local Hamiltonian problem. A k-local Hamiltonian on n qubits is a Hermitian operator $H = \sum_{e=1}^m h_e$, where each h_e is a *local* Hermitian operator acting non-trivially on at most k qubits. Formally, it is written as $h_e = \hat{h}_e \otimes \mathbb{I}_{rest}$, where $\hat{h}_e$ is defined on the Hilbert space of k qubits, and $\mathbb{I}_{rest}$ is the identity operator on the Hilbert space of the rest of the qubits. When it is clear from the context, we often use h_e instead of $\hat{h}_e$, even while referring to its action on the local Hilbert space.

In physics, k-local Hamiltonians model the local interactions between particles in a many-body system and are the central tool for describing the physics of such systems. The *energy* of the system for every state $|\psi\rangle$ is defined by $E_\psi(H) := \langle\psi|H|\psi\rangle = \sum_e\langle\psi|h_e|\psi\rangle$. The lowest possible energy of the system is called the *ground energy* and is denoted by $E_0(H)$. It is easy to verify that $E_0(H)$ is the lowest eigenvalue of H. The corresponding eigenspace is called the *ground space* of the system, and its eigenvectors are called *ground states.* A central task in condensed matter physics is to understand the properties of the ground space, as it determines the low-temperature physics of the system.

There is a deep connection between the problem of approximating the ground energy of a local Hamiltonian and the classical problem of finding an assignment with minimal violations in a local CSP. In both cases, one tries to minimize a global function that is given

[4] There are, of course, continuously many orthogonal states for every $|\alpha\rangle$, so here we simply choose one in a canonical way.

in terms of local constraints. This connection is evident if we consider the special case when the local Hermitian operators h_e are given as local *projectors* Π_e. Then for any state $|\psi\rangle$, the local energy $\langle\psi|\Pi_e|\psi\rangle$ is a number between 0 and 1 that can be viewed as a measure to how much the state is 'violating' the quantum clause Π_e. When the local energy is 0, the state is inside the null space of the projector Π_e and is said to satisfy the constraint. The total energy of the system, $E_\psi = \langle\psi|H|\psi\rangle = \sum_e \langle\psi|\Pi_e|\psi\rangle$ then corresponds to the total violation of the state $|\psi\rangle$. When the ground energy of the system is 0, necessarily the ground space is the non-vanishing intersection of all the null spaces of the local projectors, and we say that the system is satisfiable. From a physical point of view, such a system is called *frustration-free*, since any ground state of the global system also minimizes the energy of every local term Π_e.

The quantum kQSAT problem is analogous to the classical kSAT problem. Whereas in the kSAT case we are asked to decide whether a k-local CSP is satisfiable or not, in the kQSAT problem we are asked to determine whether the ground energy of a k-local Hamiltonian made of projectors is 0 or not. Unlike the truth values of SAT clauses, however, the ground energy of a k-local Hamiltonian is a continuous function that is sensitive to any infinitesimal change in the form of the local projectors. To make the kQSAT problem more physically relevant, we define it using a promise: Given a k-local Hamiltonian of projectors over n qubits and a value $b > \frac{1}{n^\alpha}$ for some constant α, decide if the ground energy of H is 0 (the YES case) or the ground energy of H is at least b (the NO case). Bravyi [6] showed that kQSAT for $k \geq 4$ is QMA$_1$-complete while Gosset and Nagaj [10] showed that 3QSAT is also QMA$_1$-complete. The class QMA$_1$ stands for 'Quantum Merlin Arthur' with one-way error, and is the quantum generalization of the classical MA$_1$ class with one-way error. The differences are that the witness can be a quantum state over poly(n) qubits, and the verifier can be an efficient quantum machine. In Ref. [6] it was known that 2QSAT has an $O(n^4)$ classical algorithm, and is therefore in P. More recently linear time algorithms for the same problem have been constructed [2, 8].

As the Hamiltonian in a 2QSAT instance is a sum of 2-qubit projectors, every local projector is defined on a 4-dimensional Hilbert space and is of rank $1, 2$ or 3. The non-zero subspace of each projector (the subspace on which it projects) is commonly referred to as the *forbidden space* of that projector and the orthogonal subspace is its *solution space*. Finally, we say that H has *no repetitions* if there does not exist any pair of different projectors $\Pi_e, \Pi_{e'}$ which act non-trivially on the same set of qubits. In the case of repetition free 2QSAT, each projector can also be indexed by the qubit pairs it acts on and the instance can be written as $H = \sum_{(u,v)\in S} \Pi_{uv}$, where $S \subseteq [n] \times [n]$ and each Π_{uv} is non-zero. For any projector Π and a state $|\psi\rangle$, we say that $|\psi\rangle$ *satisfies* Π up to ϵ if $E_\psi(\Pi) := \langle\psi|\Pi|\psi\rangle \leq \epsilon^2$. The energy $E_\psi(\Pi)$ is the *violation energy* of $|\psi\rangle$ with respect to the projector Π. Notice that when the state of the system is described by a density matrix ρ, its violation energy with respect to Π is given by $E_\rho(\Pi) := \mathrm{Tr}(\rho\Pi)$

Finally, a 2QSAT Hamiltonian H is said to have a *Star-like* configuration if there exists a pair of qubits u, v with $\Pi_{u,v} \neq 0$ such that *all* projectors involve either u or v.

Hidden QSAT. The hidden version of QSAT is defined analogously to the classical case. Our task is to decide whether a k-local Hamiltonian $H = \sum_e \Pi_e$ that is made of m k-local projectors over n qubits is frustration-free with $E_0 = 0$ (YES instance) or $E_0 > m \cdot 2\epsilon^2$ (NO instance). Here, $\epsilon > 0$ is some threshold parameter that can be assumed to be inverse polynomially small in n. Moreover, as in H–SAT, here we do not know the Hamiltonian itself; instead we can only send *quantum* states to a "worst violated oracle", which will return the index e of the projector Π_e with the highest violation energy. Since we want to generalize the notion of a probabilistic assignment that is used in H–SAT, we allow ourselves to send the oracle qubits that hold a general quantum state ρ, which can only be described by a density matrix. Recall from the previous section that this can be regarded as an ensemble of pure quantum states. Then the oracle will return the the index e for which $\mathrm{Tr}(\Pi_e \rho)$ is maximized. If the total energy of the proposed state is $\leq m \cdot \epsilon^2$ then the oracle will indicate that a satisfying assignment has been found.

6 Hidden Quantum 1SAT

The algorithm used to solve H–1SAT can be extended to solve the H–1QSAT problem as well. A 1-local projector defined on $\mathbb{C}^2$ is satisfiable if it is of rank at most 1 and can be viewed as setting the direction of the qubit on the Bloch sphere. Unlike the classical case, where we may view the 1SAT clauses as either the $|0\rangle\langle 0|$ or $|1\rangle\langle 1|$ projectors, here the projectors can point in any direction in the Bloch sphere. To handle the continuous nature of the Bloch Sphere, we consider discretizing it by using an ϵ-net that covers the whole sphere. This allows us to generalize the lists of $0-1$ strings used in H–1SAT into lists of n-qubit product states where each qubit is assigned an element of the ϵ-net.

Given a 1-local projector $|\psi\rangle\langle\psi|$, its zero space is spanned by $|\psi^\perp\rangle$. We can divide the Bloch sphere into two hemispheres, one hemisphere containing states $|\phi\rangle$ having $|\langle\psi|\phi\rangle| \leq \frac{1}{2}$ and the other with states having $|\langle\psi|\phi\rangle| > \frac{1}{2}$. An n-qubit state $a = |a_1\rangle|a_2\rangle\ldots|a_n\rangle$ is called *good* if for each qubit i, where $|\psi_i\rangle$ is its forbidden state, $|\langle\psi_i|a_i\rangle| \leq \frac{1}{2}$ and *bad* if $\forall i, |\langle\psi_i|a_i\rangle| > \frac{1}{2}$. For the n-qubit state $a = |a_1\rangle|a_2\rangle\ldots|a_n\rangle$, let $a' := |a_1^\perp\rangle|a_2^\perp\rangle\ldots|a_n^\perp\rangle$.

Now, we can sketch the H–1QSAT algorithm. Adapting the process described in Theorem 1 for an arbitrary n-qubit state a gives a list of n-qubit states, $L_{a/a'}$, where at least one state is *good*. This is formally stated in Lemma 3 and the proof is omitted to owing to space constraints.

▶ **Lemma 3.** *Let $a = |a_1\rangle \otimes \ldots \otimes |a_n\rangle$ where $|a_i\rangle, |a_i^\perp\rangle$ is a basis for qubits i, for $i = 1, \ldots, n$. Then one can produce a list, $L_{a/a'} \subset \bigotimes_{i=1}^n \{|a_i\rangle, |a_i^\perp\rangle\}$ of at most $2mn$ states such that, if the instance is satisfiable, there is at least one* good *n-qubit state in the list. The time taken to produce this list is $O(n^2 m)$.*

However, this only gives us an assignment that violates each projector by $\leq \frac{1}{4}$ while we require assignments that violate each projector by $\leq \epsilon^2$. The key observation involves constructing two lists $L_{a/a'}$ and $L_{b/b'}$ where $b \neq a, a'$ and picking a state from each list. Consider the case when both states are good. Let the states on qubit i from each list be $|a_i\rangle$ and $|b_i\rangle$ respectively. Each state defines a hemisphere R_{i,a_i} and R_{i,b_i} containing all the states that are bad with respect to the forbidden state for qubit i, $|\psi_i\rangle$. Then, $|\psi_i\rangle$, should be contained in $R_{i,a_i b_i} := R_{i,a_i} \cap R_{i,b_i}$. The optimal choice for b_i, given a_i, would be one where $|R_{i,a_i b_i}| \leq \frac{|R_{i,a_i}|}{2}$. Then, similar to performing a binary search on the Bloch Sphere, repeating this process $\log_2\left(\frac{1}{\epsilon}\right)$ times, will give a region consisting of good approximations to the forbidden state.

▶ **Theorem 4.** *Let $\epsilon > 0$. Given a* H–1QSAT *on n qubits containing m projectors, there exists an an $O((2mn)^{2\log\frac{1}{\epsilon}} \cdot mn^2)$ time algorithm, with the property that*

(a) *for a frustration free instance, it outputs an assignment where for each projector, the forbidden state is violated with probability $\leq \epsilon^2$ and*

(b) *for a* NO *instance, the algorithm outputs* UNSAT.

Proof Initially, with no information, for each qubit i, R_i = Bloch sphere. Now the algorithm executes the following steps:

- Start by picking an arbitrary state, say $\bar{a} = |0\rangle^{\otimes n}$, and construct $L_{|0\rangle^{\otimes n}/|1\rangle^{\otimes n}}$ as per the procedure in Lemma 3. For each $a \in L_{|0\rangle^{\otimes n}/|1\rangle^{\otimes n}}$:
  - a defines the region R_{i,a_i} in this branch of the iteration.
  - For $i = 1, \ldots, n$ pick a basis $\{|b_i\rangle, |b_i^{\perp}\rangle\}$ such that their equator bisects R_{i,a_i}.
  - Set $\bar{b} = b_1 \ldots b_n$, construct $L_{\bar{b}/\bar{b}'}$ and for each $b \in L_{\bar{b}/\bar{b}'}$:
    - The tuples (a, b) define the region $R_{i,a_i b_i}$ in this branch.
    - Repeat the process to find $\bar{c}$ to bisect each $R_{i,a_i b_i}$;
    - Find a new region $R_{i,a_i b_i c_i}$ for each $c \in L_{\bar{c}/\bar{c}'}$.
    - Continue the recursion up to $\log_2\left(\frac{1}{\epsilon}\right)$ depth and let the last list be $L_{z/z^{\perp}}$.
    - Propose $|\phi^{\perp}\rangle = \bigotimes_{i=0}^{n} |\varphi_i^{\perp}\rangle$ where $\forall i, |\varphi_i\rangle \in R_{i,a_i b_i \ldots z_i}$ to the oracle. Output $|\phi^{\perp}\rangle$ if the oracle returns YES, otherwise continue.
- Output UNSAT if none of the trials satisfy the instance.

This algorithm essentially creates a recursion tree with each new string created where the width of the recursion at each point is $2mn$ and the depth is $\log_2\left(\frac{1}{\epsilon}\right)$. This leads to $(2mn)^{\log_2 \frac{1}{\epsilon}}$ trials to be proposed at the end and the number of lists created is also $(2mn)^{\log_2 \frac{1}{\epsilon}}$, each at a cost of $O(mn^2)$. Hence, the total running time of the algorithm is $O((2mn)^{\log \frac{1}{\epsilon}} \cdot mn^2)$.

To argue the correctness of this algorithm, we analyze region $R_{i,a_i b_i \ldots z_i}$ obtained at the leaf of the recursion tree. At the beginning, let $\forall\ i$, $|R_i| = 1$ (the complete Bloch sphere) and the only guarantee for each list is that there is at least one good string in it. Tracing the path in the recursion tree to the leaf, let us assume that each step of the recursion picks a good string i.e. $a, b, \ldots, z$ are all good strings. For a and $\forall i$, the forbidden state $|\psi_i\rangle$ is in the opposite hemisphere to $|a_i\rangle$ which reduces the size of the region to $|R_{i,a_i}| = 1/2$. Taking (a, b) at the next iteration, the region for each qubit is the over lap of two hemispheres $R_{i,a_i} \cap R_{i,b_i}$ and by construction, since b_i bisects R_{i,a_i}, the overlaps of the hemispheres also bisect R_{i,a_i} setting $|R_{i,a_i b_i}| = 1/4$. As this pattern continues, each step of the iteration halves the region for qubit i and we are left with regions of size at most ϵ at the end of the branch. If the instance is satisfiable, the state proposed will satisfy each projector up to ϵ resulting in the oracle to return YES. Of course, when one of the strings chosen is bad, the proposal $|\phi_j^{\perp}\rangle$ for some qubit j will end up having a large inner product with the forbidden state $|\psi_j\rangle$ and will result in the oracle returning the ID of the projector involving j. This concludes the proof. ☐

7 Hidden Quantum 2SAT

This section deals with a 2QSAT instance that is hidden and can only be accessed by a worst-violation oracle. We show how learn the underlying local Hamiltonian to precision ϵ by finding 2-local projectors Π'_e such that $\|\Pi_e - \Pi'_e\| \leq \epsilon$ for every projector Π_e. This yields an approximate local Hamiltonian $H' = \sum_e \Pi'_e$ whose ground energy is at most $m\epsilon$ away from the ground energy of the original Hamiltonian $H = \sum_e \Pi_e$. If ϵ is set such that $m\epsilon$ is much smaller than the promise gap of the initial Hamiltonian H (which merely requires $\epsilon < 1/\operatorname{poly}$), then the Hamiltonian H' will have a promise gap as well. This is stated in the theorem below with the proof and algorithm details omitted owing to space constraints.

▶ **Theorem 5.** *Given a* H–2QSAT *problem* $H = \sum_{(u,v)} \Pi_{uv}$ *on* n *qubits, and precision* ϵ*. If the interaction graph for* H *is not Star-like, then there is an* $O(n^4 + n^2 \log\left(\frac{1}{\epsilon}\right))$ *algorithm that can find an approximation* $H' = \sum_{(u,v)} \Pi'_{uv}$ *where* $\forall\ (u,v), \|\Pi'_{uv} - \Pi_{uv}\| \leq \epsilon$

The algorithm proceeds by:

1. Identifying two pairs of qubits $(i,j) \neq (k,\ell)$ on which two projectors are defined, Π_{ij} and $\Pi_{k\ell}$, and finding a constant approximation for these projectors;
2. Improving the constant approximation of the two projectors recursively so that the approximation improves by a factor of 2 in each iteration and
3. Using the ϵ-approximation of a projector to identify the rest of the independent projectors and approximating them to ϵ-precision.

Using the H' output by the above algorithm in a procedure which could find a good approximation to the ground energy of H' would completely solve H–2QSAT. At this time, though, existing 2QSAT algorithms [2, 6, 8] are not robust to such errors and seem to require $\frac{1}{\exp(n)}$ precision. Our algorithm for H–2QSAT does allow one to learn the projectors to exponential precision, since the dependence on ϵ in Theorem 5 is merely logarithmic. However, in this parameter regime our algorithm is somewhat unrealistic, as this would require the oracle to be able to distinguish between values that are exponentially close together[5]. If our oracle were constrained to be implementable in polynomial time by an experimenter, acting on polynomially many copies of the proposed state ρ, then one could only learn the instance up to error $\epsilon = \frac{1}{\text{poly}(n)}$. A natural open question is to determine whether one can still solve 2QSAT when one only knows the individual clauses to inverse polynomial precision; we believe this is a fundamental question about the nature of 2QSAT, which is left for future work.

References

1 Daniel S. Abrams and Seth Lloyd. Nonlinear quantum mechanics implies polynomial-time solution for NP-complete and #P problems. *Phys. Rev. Lett.*, 81:3992–3995, 1998.

2 Itai Arad, Miklos Santha, Aarthi Sundaram, and Shengyu Zhang. Linear time algorithm for quantum 2SAT. *CoRR*, abs/1508.06340, 2015. To appear in the 43rd International Colloquium on Automata, Languages and Programming. URL: `http://arxiv.org/abs/1508.06340`.

3 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. Erratum: Information Processing Letters 14(4): 195 (1982).

4 Xiaohui Bei, Ning Chen, and Shengyu Zhang. On the complexity of trial and error. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 31–40. ACM, 2013. `doi:10.1145/2488608.2488613`.

5 Xiaohui Bei, Ning Chen, and Shengyu Zhang. Solving linear programming with constraints unknown. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 2015. `doi:10.1007/978-3-662-47672-7_11`.

6 Sergey Bravyi. Efficient algorithm for a quantum analogue of 2-SAT. In Kazem Mahdavi, Deborah Koslover, and Leonard L. Brown, editors, *Contemporary Mathematics*, volume

[5] This seems to give the oracle too much power - because having the ability to distinguish exponentially close quantum states would allow one to solve PP-hard problems [1]. In contrast all of the problems considered are in NP due to the presence of classical, poly(n) size witnesses.

536. American Mathematical Society, 2011. URL: http://arxiv.org/abs/quant-ph/0602108.

7 S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium*, pages 151–158, New York, 1971. ACM.

8 Niel de Beaudrap and Sevag Gharibian. A linear time algorithm for quantum 2-SAT. *CoRR*, abs/1508.07338, 2015. To appear in 31st Conference on Computational Complexity. URL: http://arxiv.org/abs/1508.07338.

9 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

10 David Gosset and Daniel Nagaj. Quantum 3-SAT is QMA1-complete. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:756–765, 2013. doi:10.1109/FOCS.2013.86.

11 Gábor Ivanyos, Raghav Kulkarni, Youming Qiao, Miklos Santha, and Aarthi Sundaram. On the complexity of trial and error for constraint satisfaction problems. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP*, volume 8572 of *Lecture Notes in Computer Science*, pages 663–675. Springer, 2014. doi:10.1007/978-3-662-43948-7_55.

12 M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967. doi:10.1002/malq.19670130104.

13 L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

14 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

The Parameterized Complexity of Fixing Number and Vertex Individualization in Graphs[*]

Vikraman Arvind[1], Frank Fuhlbrück[2], Johannes Köbler[3], Sebastian Kuhnert[4], and Gaurav Rattan[5]

1 **The Institute of Mathematical Sciences, Chennai, India**
arvind@imsc.res.in
2 **Institut für Informatik, Humboldt-Universität zu Berlin, Germany**
fuhlbfra@informatik.hu-berlin.de
3 **Institut für Informatik, Humboldt-Universität zu Berlin, Germany**
koebler@informatik.hu-berlin.de
4 **Institut für Informatik, Humboldt-Universität zu Berlin, Germany**
kuhnert@informatik.hu-berlin.de
5 **The Institute of Mathematical Sciences, Chennai, India**
grattan@imsc.res.in

Abstract

In this paper we study the complexity of the following problems:

1. Given a colored graph $X = (V, E, c)$, compute a minimum cardinality set of vertices $S \subset V$ such that no nontrivial automorphism of X *fixes* all vertices in S. A closely related problem is computing a minimum base S for a permutation group $G \le S_n$ given by generators, i.e., a minimum cardinality subset $S \subset [n]$ such that no nontrivial permutation in G fixes all elements of S. Our focus is mainly on the *parameterized complexity* of these problems. We show that when $k = |S|$ is treated as parameter, then both problems are MINI[1]-hard. For the dual problems, where $k = n - |S|$ is the parameter, we give FPT algorithms.
2. A notion closely related to fixing is called individualization. Individualization combined with the Weisfeiler-Leman procedure is a fundamental technique in algorithms for Graph Isomorphism. Motivated by the power of individualization, in the present paper we explore the complexity of individualization: what is the minimum number of vertices we need to individualize in a given graph such that color refinement "succeeds" on it. Here "succeeds" could have different interpretations, and we consider the following: It could mean the individualized graph becomes: (a) discrete, (b) amenable, (c) compact, or (d) refinable. In particular, we study the parameterized versions of these problems where the parameter is the number of vertices individualized. We show a dichotomy: For graphs with color classes of size at most 3 these problems can be solved in polynomial time, while starting from color class size 4 they become W[P]-hard.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity – General

Keywords and phrases Parameterized complexity, graph automorphism, fixing number, base size, individualization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.13

[*] This work was supported by the Alexander von Humboldt Foundation in its research group linkage program. The third and fourth authors are supported by DFG grant KO 1053/7-2.

1 Introduction

A permutation π on the vertex set V of a (vertex) colored graph $X = (V, E, c)$ is an *automorphism* if π preserves edges and colors. Uncolored graphs can be seen as the special case where all vertices have the same color. The automorphisms of X form the group $\mathrm{Aut}(X)$, which is a subgroup of the symmetric group $\mathrm{Sym}(V)$ of all permutations on V.

A *fixing set* for a colored graph $X = (V, E, c)$ is a subset S of vertices such that there is no nontrivial automorphism of X that fixes every vertex in S. The *fixing number* of X is the cardinality of a smallest size fixing set of X. This notion was independently studied in [10, 15, 16]. A nice survey on this and related topics is by Bailey and Cameron [8].

In this paper, one of the problems of interest is the computational complexity of computing the fixing number of graphs:

▶ **Problem 1.1.** k-Rigid

Input: A colored graph X and an integer k
Parameter: k
Question: Is there a subset S of k vertices in V such that there are no nontrivial automorphisms of X that fix each vertex of S?

There is a closely related problem that has received some attention. Let $G \leq S_n$ be a permutation group on $[n]$. A *base* of G is a subset $S \subset [n]$ such that no nontrivial permutation of G fixes each point in S, i.e., the pointwise stabilizer subgroup $G_{[S]} = \{g \in G \mid i^g = i \ \forall\, i \in S\}$ of G is the trivial subgroup $\{1\}$.

▶ **Problem 1.2.** k-Base-Size

Input: A generating set for a permutation group G on $[n]$ and an integer k
Parameter: k
Question: Is there a subset $S \subset [n]$ of size k such that no nontrivial permutation of G fixes each point in S?

Note that a graph X is in k-Rigid if and only if $\mathrm{Aut}(X)$ is in k-Base-Size.

Computing a minimum cardinality base for $G \leq S_n$ given by generators is shown to be NP-hard by Blaha [9]. The same paper also gives a polynomial-time $\log\log n$ factor approximation algorithm for the problem, i.e., the algorithm outputs a base of size bounded by $b(G)\log\log n$, where $b(G)$ denotes the optimal base size. We show that this approximation factor cannot be improved unless P = NP; see Theorem 2.7.

In this paper our focus is on the parameterized complexity of these problems. Arvind has shown that k-Base-Size is in FPT for transitive groups and groups with constant orbit size [2], and raised the question whether this can be extended to more general permutation groups. We show that both k-Rigid and k-Base-Size are MINI[1]-hard, even when the automorphism group of the given graph X (resp., the given group G) is an elementary 2-group; see Section 2.

We also consider the dual problems $(n-k)$-Rigid and $(n-k)$-Base-Size, which ask whether the given graph or group have a fixing set or base that consists of all but k vertices or points and k is the parameter. We show that these problems are fixed parameter tractable. More precisely, we give an $k^{O(k^2)} + n^{O(1)}$ time algorithm for $(n-k)$-Base-Size and an $k^{O(k^2)} n^{O(1)}$ time algorithm for $(n-k)$-Rigid in Section 3.

Color refinement and individualization. A broader question that arises is in the context of the Graph Isomorphism problem: Given two colored graphs $X = (V, E, c)$ and $X' =$

(V', E', c') the problem is to decide if they are *isomorphic*, i.e., whether there is a bijection $\pi\colon V \to V'$ such that for all $x \in V$, $c'(x^\pi) = c(x)$ and for all $x, y \in V$, $(x, y) \in E$ if and only if $(x^\pi, y^\pi) \in E'$. Color refinement is a classical heuristic for Graph Isomorphism, and in combination with other tools (group-theoretic/combinatorial) it has proven successful in Graph Isomorphism algorithms (e.g. in the two most important papers in the area [7, 6]). The basic color refinement procedure works as follows on a given colored graph $X = (V, E, c)$. Initially each vertex has the color given by c. The refinement step is to color each vertex by the tuple of its own color followed by the colors of its neighbors (in color-sorted order). The refinement procedure continues until the color classes become stable. If the multisets of colors are different for two graphs X and X', we can conclude that they are not isomorphic. Otherwise, more processing needs to be done to decide if the input graphs are isomorphic. One important approach in this area is to combine *individualization* of vertices with color refinement: Given a graph $X = (V, E)$ and k vertices $v_1, v_2, \dots, v_k \in V$, first these k vertices are assigned distinct colors $c_1, c_2, \dots, c_k$, respectively. Then, with this as initial coloring, the color refinement procedure is carried out as before. Individualization is used both in the algorithms with the best worst case complexity [7, 6] and in practical isomorphism solvers [21]. Note that individualizing a vertex v results in fixing v, as every automorphism must preserve the unique color of v.

In [5] we have examined several classes of colored graphs in connection with the color refinement procedure. They form a hierarchy:

$$\textsc{Discrete} \subsetneq \textsc{Amenable} \subsetneq \textsc{Compact} \subsetneq \textsc{Refinable} \tag{1}$$

- $X \in \textsc{Discrete}$ if running color refinement on X results in singleton color classes.
- $X \in \textsc{Amenable}$ if for any X' that is non-isomorphic to X, color refinement on X and X' results in different stable colorings [5].
- $X \in \textsc{Compact}$ if every fractional automorphism of X is a convex combination of automorphisms of X [25]. Here, automorphisms are viewed as permutation matrices that commute with the adjacency matrix A of X, and fractional automorphisms are doubly stochastic matrices that commute with A.
- $X \in \textsc{Refinable}$ if two vertices u and v of X receive the same color in the stable coloring if and only if there is an automorphism of X that maps u to v [5].

For these graph classes, various efficient isomorphism and automorphism algorithms are known. Motivated by the power of individualization in relation to color refinement, we consider the following type of problems.

▶ **Problem 1.3.** k-$\mathcal{C}$ (where $\mathcal{C}$ is a class of colored graphs)
Input: A colored graph $X = (V, E, c)$ and an integer k
Parameter: k
Question: Are there k vertices of X so that individualizing them results in a graph in $\mathcal{C}$?

It turns out that for each class $\mathcal{C}$ in the hierarchy (1), the problem k-$\mathcal{C}$ is W[P]-hard, even when the input graph has color class size at most 4. For color class size at most 3 however, the problems become polynomial time solvable. For the class $\textsc{Discrete}[\ell]$ of all colored graphs where ℓ rounds of color refinement turn all color classes into singletons, we show that k-$\textsc{Discrete}[\ell]$ is W[2]-hard. These results are in Section 4.

Additionally, we give an FPT algorithm for the dual problem $(n-k)$-$\textsc{Discrete}$ that asks whether there is a way to individualize all but k vertices so that the input graph becomes discrete; see Section 5.

Color valence. A beautiful observation due to Zemlyachenko [27], that plays a crucial role in [7], concerns the color valence of a graph. Given a colored graph $X = (V, E, c)$, the *color degree* $\deg_C(v)$ of a vertex v in a color class $C = \{v \in V \mid c(v) = c_0\}$ is the number of neighbors of v in C. The *color co-degree* of v in C is $\text{co-deg}_C(v) = |C| - \deg_C(v)$. The *color valence* of X is defined as $\max_{v,C} \min\{\deg_C(v), \text{co-deg}_C(v)\}$. Zemlyachenko has shown [27] that in any n-vertex graph $X = (V, E)$ we can individualize $O(n/d)$ vertices so that the vertex colored graph obtained after color refinement has color valence at most d. This gives rise to the following natural algorithmic problem:

▶ **Problem 1.4.** k-COLOR-VALENCE

Input: A colored graph $X = (V, E, c)$ and two numbers k and d
Parameter: k
Question: Is there a set of k vertices such that when these are individualized, the graph obtained after color refinement has color valence bounded by d?

We show that this problem is W[P]-complete; see Corollary 4.4.

2 The number of fixed vertices as parameter

In this section we show that the parameterized problems k-RIGID and k-BASE-SIZE are both MINI[1]-hard. The class MINI[1] contains all parameterized problems that are FPT-reducible to MINI-3SAT. Both were defined in [12, 14].

▶ **Problem 2.1** ([12, 14])**.** MINI-3SAT

Input: A formula F in 3-CNF of size bounded by $k \log n$ and the number n in unary
Parameter: k
Question: Is there a boolean assignment to the variables that satisfies the formula F?

It turns out that MINI[1] is contained in the class W[1] [14] and has a variety of complete problems in it. Moreover, it has been linked to the exponential time hypothesis.

▶ **Lemma 2.2** ([12, 14])**.** *If* MINI[1] = FPT *then there is a* $2^{o(n)}$ *time algorithm for* 3SAT.

▶ **Theorem 2.3.** *The problem k-BASE-SIZE is* MINI[1]*-hard, even for elementary* 2*-groups.*

Proof. It is easy to see that MINI-3SAT in which each variable occurs at most 3 times is also MINI[1]-complete, by modification of a standard NP-completeness proof. This only increases the size by a constant factor. We will give an FPT many-one reduction from this variant of MINI-3SAT to k-BASE-SIZE. Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, and n in unary, be a MINI-3SAT instance with variable occurrences bounded by 3. Since the size of F is bounded by $k \log n$, we have $m \le k \log n$. Let V denote the set of distinct variables in F. We also have $|V| \le k \log n$. We partition V as $V = \bigsqcup_{i=1}^{k} V_i$, where $|V_i| \le \log n$ for $1 \le i \le k$. For each i, the set $T_i = \{0, 1\}^{V_i}$ consisting of all truth assignments to variables in V_i has size $|T_i| \le n$. Define the universe $U = \{1, 2, \ldots, m, m + 1, \ldots, m + k\}$. For each truth assignment $a \in T_i$ we define the subset $S_{i,a} \subset U$ consisting of $m + i$ along with all j such that a satisfies C_j, i.e.,

$$S_{i,a} = \{m + i\} \cup \{j \mid C_j \text{ contains a literal that is true under } a\}.$$

Clearly, since each variable occurs at most 3 times in F and since $|a| = |V_i| \le \log n$, it follows that $|S_{i,a}| \le 1 + 3 \log n$. The following claim is straightforward.

▶ **Claim 2.4.** The collection of sets $\{S_{i,a} \mid 1 \le i \le k, a \in T_i\}$ with universe U has a set cover of size k if and only if F is satisfiable.

We will now transform this special set cover instance into an instance of k-Base-Size. The group we shall consider is $\mathbb{F}_2^{m+k}$, i.e., the product of $m+k$ copies of the group on $\{0,1\}$ defined by addition modulo 2. Treating each set $S_{i,a}$ as a subset of the coordinates $1, 2, \ldots, m+k$, we can associate a copy of $\mathbb{F}_2^{|S_{i,a}|}$ with it. Consider the set $\Omega = \bigsqcup_{i,a} \mathbb{F}_2^{|S_{i,a}|}$. Note that $|\Omega| = \sum_{i,a} 2^{|S_{i,a}|} \le nk$. The group $\mathbb{F}_2^{m+k}$ acts faithfully on Ω as follows. Given an element $u \in \mathbb{F}_2^{m+k}$ and a point $v \in \mathbb{F}_2^{|S_{i,a}|}$, let $u_{i,a}$ denote the projection of u to the coordinates in $S_{i,a}$. Then u maps v to $v \oplus u_{i,a}$. Thus, $\mathbb{F}_2^{m+k}$ is a permutation group acting on Ω given by the standard basis of $m+k$ unit vectors as generating set. The following straightforward claim completes the reduction.

▶ **Claim 2.5.** The group $\mathbb{F}_2^{m+k}$ acting on Ω, as defined above, has a base of size k if and only if the set cover instance $(U, \{S_{i,a} \mid 1 \le i \le k, a \in T_i\})$ has a set cover of size k.

To see the claim, observe that $V \subseteq \Omega$ is a base if and only if the sets $S_{i,a}$ with $V \cap \mathbb{F}_2^{|S_{i,a}|} \neq \emptyset$ form a set cover for U. Indeed, a point $p \in U$ is covered by these sets if and only if all $u \in \mathbb{F}_2^{m+k}$ with $u_p = 1$ move an element of V. ◀

▶ **Theorem 2.6.** *The problem k-Rigid is* MINI[1]*-hard.*

Proof. It suffices to encode the k-Base-Size instance constructed in the proof of Theorem 2.3 as a k-Rigid instance (X, k) with the following properties. The graph X has $|\Omega| + 2(m+k)$ vertices and at most $|\Omega|(1 + 3\log n)$ edges. Further, the above k-Base-Size instance has a base of size k if and only if the graph X has a fixing set of size k.

We explain the construction of X. Let $l = m+k$. The vertex set of X is $\Omega \cup I_1 \cup \cdots \cup I_l$ where each set $I_j = \{a_j^0, a_j^1\}$ is a distinct color class of size 2. The edge set of X is defined as follows. Let $v = (b_1, \ldots, b_p) \in \mathbb{F}_2^{|S_{i,a}|}$ be a vertex in Ω and let $S_{i,a} = \{i_1, i_2, \ldots, i_p\}$ be the set of coordinates occurring in v. Then we connect v to the vertices $a_{i_q}^{b_q}$, for each $q = 1, \ldots, p$. This finishes the construction of X.

We claim a one-to-one correspondence between the permutation group $\mathbb{F}_2^{m+k}$ acting on Ω and $\mathrm{Aut}(X)$. Indeed, any vector $v = (b_1, \ldots, b_l) \in \mathbb{F}_2^{m+k}$ can be associated with a unique automorphism σ of X as follows. The automorphism σ flips the color class I_j if and only if $b_j = 1$. For a vertex $u \in \Omega$, define $\sigma(u) = v(u)$ using the action of $\mathbb{F}_2^{m+k}$ on Ω. It is easy to check that σ respects the adjacencies inside X. Note that the action of an automorphism of X is determined by its action on $I_1, \ldots, I_l$, so this is a one-to-one correspondence.

Consequently, a set $J \subset \Omega$ is a base for the original k-Base-Size instance if and only if J is a fixing set for the graph X. We observe that we can always avoid fixing a vertex u inside $I_1 \cup \cdots \cup I_l$ by instead fixing some neighbor of $u \in \Omega$. Therefore, the original k-Base-Size instance has a base of size k if and only if the graph X has a fixing set of size k. ◀

We end this section with some consequences of our hardness proofs on the approximability of the minimum base size of a group. There is a $\log\log n$ factor approximation algorithm due to Blaha [9] for the minimum base problem (in fact, a careful analysis yields a $\ln\ln n$-factor approximation). In this connection we have an interesting observation about the set cover problem instances that arise in Theorem 2.3 (Claim 2.4). A more general version is the B-Set-Cover problem: we are given a collection of subsets of size at most B of some universe U and the problem is to find a minimum size set cover. Trevisan [26] has shown that there is no approximation algorithm for this problem with approximation factor smaller than $\ln B - O(\ln\ln B)$ unless $\mathsf{P} = \mathsf{NP}$. This leads us to the following theorem.

▶ **Theorem 2.7.** *The approximation factor of $\ln\ln n$ in Blaha's approximation algorithm for minimum base cannot be improved, even for elementary abelian 2-groups, unless* $\mathsf{P} = \mathsf{NP}$.

Proof. The reduction from $(\log n)$-SET-COVER to the minimum base problem that is explained in the proof of Theorem 2.3 preserves the optimal solution size. Furthermore, it is easy to see that this reduction carries over to all $(\log n)$-SET-COVER instances. Combined with Trevisan's result, this completes the proof. ◀

3 The number of non-fixed vertices as parameter

In this section we show that the problems $(n-k)$-RIGID and $(n-k)$-BASE-SIZE are in FPT with running time $k^{O(k^2)}n^{O(1)}$. We will show this first for $(n-k)$-BASE-SIZE. We need some permutation group theory.

Let $G \leq \mathrm{Sym}(\Omega)$ be a permutation group acting on a set Ω. The *support* of a permutation $g \in G$ is $\mathrm{supp}(g) = \{i \in \Omega \mid i^g \neq i\}$. The *orbit* of a point $i \in \Omega$ is the set $i^G = \{i^g \mid g \in G\}$. The group G is *transitive* if it has a single orbit in Ω. Let $G \leq \mathrm{Sym}(\Omega)$ be transitive. A subset $\Delta \subseteq \Omega$ is a *block* if for every $g \in G$ its image $\Delta^g = \{i^g \mid i \in \Delta\}$ is either $\Delta^g = \Delta$ or $\Delta^g \cap \Delta = \emptyset$. Clearly, Ω and singleton sets are blocks for any G. All other blocks are called nontrivial. A transitive group G is *primitive* if it has no nontrivial blocks.

There are polynomial-time algorithms that take as input a generating set for some $G \leq \mathrm{Sym}(\Omega)$ and compute its orbits and maximal nontrivial blocks [19]. We can test if G is primitive in polynomial time. If G is transitive on Ω we can compute a maximal nontrivial block Δ_1. It is easy to see that Δ_1^g is also a block for each $g \in G$. This yields a partition of Ω into blocks (which are said to constitute a block system for G): $\Omega = \Delta_1 \sqcup \Delta_2 \sqcup \ldots \sqcup \Delta_\ell$. The group G acts transitively on the blocks $\{\Delta_1, \Delta_2, \ldots, \Delta_\ell\}$. Furthermore, since these are maximal blocks, the group action is primitive. The following classic result is useful for our algorithm.

▶ **Lemma 3.1.** [13, Lemma 3.3D] *Suppose $G \leq S_n$ is primitive and G is neither A_n nor S_n itself. If there is an element $g \in G$ such that $|\mathrm{supp}(g)| \leq k$, then $|\Omega| \leq (k-1)^{2k}$.*

Here, $A_n = \mathrm{Alt}([n])$ denotes the subgroup of S_n that consists of those permutations that can be written as the product of an even number of transpositions.

▶ **Theorem 3.2.** *There is a $k^{O(k^2)}+n^{O(1)}$ time algorithm for the $(n-k)$-BASE-SIZE problem.*

Proof. Let $G \leq S_n$ be the input group given by a generating set and let k be the parameter. We call a set $S \subseteq [n]$ a *co-base* for G, if $[n] \setminus S$ is a base for G. The algorithm finds a co-base S of size k if it exists. During its execution, the algorithm may decide to fix some points. Since in this case the actual group G is replaced by the pointwise stabilizer subgroup, there is no need to store these points. The algorithm proceeds as follows.

1. Let $O_1, O_2, \ldots, O_\ell$ be the orbits of the group G. If $\ell \geq k$ then the set S obtained by picking one point from each of the orbits $O_1, O_2, \ldots, O_k$ is a co-base for G.
2. Suppose $\ell < k$, and there is an orbit O_i of size more than k^{2k} on which G's action is not primitive. In this case compute a maximal block system of G in O_i, $O_i = \Delta_{i1} \sqcup \ldots \sqcup \Delta_{ir_i}$, and deal with the following subcases:
   a. If $r_i > k$, then the set S obtained by picking one point from each block $\Delta_{i1}, \ldots, \Delta_{ik}$ is a co-base for G.
   b. If $r_i \leq k$, then each block Δ_{ij} is of size at least k^{2k-1} which is strictly more than k. Thus any $n-k$ sized subset of $[n]$ intersects each block Δ_{ij} and hence the support of any permutation that moves the blocks. Let H be the subgroup of G that setwise stabilizes all blocks Δ_{ij}. The subgroup H can be computed from G in polynomial time

using the Schreier-Sims algorithm [19]. Replace G by H and go to Step 1. This step is invoked at most k times since each invocation increases the number of orbits.

3. Suppose $\ell < k$, and there is an orbit O_i of size more than k^{2k} such that G is primitive on O_i, but different from $\mathrm{Sym}(O_i)$ and $\mathrm{Alt}(O_i)$. Then any k points of O_i form a co-base for G (by Lemma 3.1).
4. Suppose there is an orbit O_i of size more than k^{2k} such that G restricted to O_i is either $\mathrm{Sym}(O_i)$ or $\mathrm{Alt}(O_i)$. Then fix the first $|O_i| - k$ elements of O_i (the choice of the subset of points fixed does not matter as both $\mathrm{Sym}(O_i)$ and $\mathrm{Alt}(O_i)$ are t-transitive for $t \le |O_i| - 2$). Replace G by the subgroup H that fixes the first $|O_i| - k$ elements of O_i and go to Step 1. This step is invoked at most once.
5. This step is only reached if all orbits are of size at most k^{2k}, implying that the entire domain size is at most k^{2k+1}. Hence, the algorithm can find a co-base S of size k by brute-force search in $k^{O(k^2)}$ time if it exists.

The brute-force computation (done in the last step), when the search space is bounded by k^{2k+1}, costs $k^{O(k^2)}$. The rest of the computation uses the standard group-theoretic algorithms [19] whose running time is polynomially bounded by n. Therefore, the overall running time of the algorithm is bounded by $k^{O(k^2)} + k\,n^{O(1)}$. As $k \le n$, the theorem follows.

We note that the algorithm is in fact a kernelization algorithm. It computes in $n^{O(1)}$ time a kernel of size k^{2k+1} (where size refers to the size of the domain on which the group acts). ◀

We now show the main result of this section, i.e., that $(n-k)$-Rigid is in FPT.

▶ **Theorem 3.3.** *There is a $k^{O(k^2)}n^{O(1)}$ time algorithm for the $(n-k)$-*Rigid *problem.*

Proof. Let $X = (V, E, c)$ be a colored n-vertex graph and k as parameter be an instance of $(n-k)$-Rigid. If we can use a subroutine for the Graph Isomorphism problem then we can compute a generating set for the automorphism group $\mathrm{Aut}(X)$ of X with polynomially many calls to this subroutine [20]. With this generating set as input we can then run the algorithm of Theorem 3.2 to compute an $(n-k)$ size fixing set for X, if it exists, in time $k^{O(k^2)}n^{O(1)}$.

However, it turns out that we can avoid using the Graph Isomorphism subroutine and still solve the problem in $k^{O(k^2)}n^{O(1)}$ time with the following observations:

1. We note that any set of size $n-k$ will intersect the support of any element $\sigma \in \mathrm{Aut}(X)$ if $|\mathrm{supp}(\sigma)| > k$. Thus, we only need to collect all elements of support bounded by k.
2. An automorphism $\sigma \in \mathrm{Aut}(X)$ is defined to be a *minimal support* automorphism of X if there is no nontrivial automorphism $\varphi \in \mathrm{Aut}(X)$ such that $\mathrm{supp}(\varphi) \subsetneq \mathrm{supp}(\sigma)$. For any nontrivial automorphism $\pi \in \mathrm{Aut}(X)$ such that $|\mathrm{supp}(\pi)| \le k$, there is a minimal support automorphism $\varphi \in \mathrm{Aut}(X)$ such that $|\mathrm{supp}(\varphi)| \le k$ and $\mathrm{supp}(\varphi) \subseteq \mathrm{supp}(\pi)$.
3. We observe that Schweitzer's algorithm in [24] can be used to compute, in $k^{O(k)}n^{O(1)}$ time, the set M of all minimal support automorphisms $\sigma \in \mathrm{Aut}(X)$ such that $|\mathrm{supp}(\sigma)| \le k$.
4. Let G' be the subgroup of $\mathrm{Aut}(X)$ generated by M. It follows from the above discussion that an $n-k$ sized subset of V is a base for $\mathrm{Aut}(X)$ (and thus a fixing set for X) if and only if it is a base for G'. We can apply the algorithm of Theorem 3.2 to compute such a base if it exists. ◀

4 The number of individualized vertices as parameter

In this section, we show that the problem k-$\mathcal{C}$ is W[P]-hard for all classes $\mathcal{C}$ of the color refinement hierarchy (1). To this end, we give a reduction from Weighted Monotone Circuit Satisfiability, which is known to be W[P]-complete [1].

▶ **Problem 4.1.** WEIGHTED MONOTONE CIRCUIT SATISFIABILITY

Input: A monotone boolean circuit C on n inputs and an integer k
Parameter: k
Question: Is there an assignment $x \in \{0,1\}^n$ of Hamming weight k so that $C(x) = 1$?

▶ **Theorem 4.2.** *For all classes $\mathcal{C}$ of the color refinement hierarchy* (1)*, k-$\mathcal{C}$ is* W[P]*-hard, even for graphs of color class size at most* 4*.*

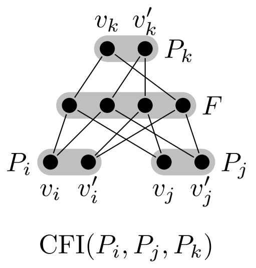
Proof. We will give a parameter-preserving reduction that maps positive instances of WEIGHTED MONOTONE CIRCUIT SATISFIABILITY to positive instances of k-DISCRETE, while negative instances are mapped to negative instances of k-REFINABLE. A similar reduction was used to show that the classes from the color refinement hierarchy (1) are all P-hard [5], which in turn builds on ideas of Grohe [17].

Let $\langle C, k\rangle$ be the given instance of WEIGHTED MONOTONE CIRCUIT SATISFIABILITY, and let n be the number of inputs of the circuit C. We define a graph X_C. For each gate g_k of C (including the input gates), X_C contains a vertex pair $P_k = \{v_k, v'_k\}$, which forms a color class. If a pair corresponds to an input gate, we call it an *input pair.* The intention is that setting an input g_i to 1 corresponds to individualizing the vertex v_i; we will add gadgets to X_C so that after color refinement it holds also for each non-input gate g_k that $g_k = 1$ if and only if v_k and v'_k have different colors.

To achieve this, we use the gadgets given in Figure 1. The basic building block is the gadget CFI(P_i, P_j, P_k) introduced by Cai, Fürer, and Immerman [11]. It connects the three pairs P_i, P_j, and P_k using four additional vertices as depicted. These four vertices form a color class F; each instance of the gadget uses its own copy of F. This gadget has the property that every automorphism flips either none or exactly two of the pairs P_i, P_j and P_k; thus the CFI-gadget implements the XOR function in the sense that any automorphism must flip P_k if and only if it flips exactly one of P_i and P_j. In our case, however, the CFI-gadget implements the AND function: If both P_i and P_j are distinguished (either by direct individualization or in previous rounds of color refinement), the vertices of the inner color class F and consequently P_k will be distinguished in two rounds of color refinement. Conversely, if at most one of the pairs P_i and P_j is distinguished, then the color class F is split into two color classes of size 2 and color refinement stops at this point, leaving the other two pairs non-distinguished. For each AND gate $g_k = g_i \wedge g_j$ in C, we add the gadget CFI(P_i, P_j, P_k) to X_C.

The second gadget we use is IMP(P_i, P_k). It consists of the gadget CFI(F', F'', P_k), where F' and F'' are vertex pairs that form color classes of size two, and perfect matchings that connect these pairs to P_i; see Fig. 1. Again, each instance of this gadget gets its own copy of the color classes F, F' and F''. There is an automorphism of IMP(P_i, P_k) that flips the vertices in P_i, but none that flips the vertices in P_k. In the color refinement setting, this gadget implements the IMPLICATION function: When P_i is distinguished, this will propagate to both F' and F'', and consequently also to F and P_k. Conversely, distinguishing P_k will only split F into two color classes of size 2 before color refinement stops. For each OR gate $g_k = g_i \vee g_j$ in C, we add the gadgets IMP(P_i, P_k) and IMP(P_j, P_k) to X_C. For the output gate g_ℓ of C, we add a second vertex pair Q and the gadget IMP(P_ℓ, Q) to X_C.

Our above analysis of the gadgets ensures that the following invariant holds when running color refinement on X_C after individualizing a subset of its input pairs: For each IMPLICATION gadget IMP(P_i, P_k) in X_C the pair P_k can only be distinguished if P_i is distinguished, and for each AND gadget CFI(P_i, P_j, P_k) the pair P_k can only be distinguished if both P_i and P_j are distinguished. This implies the following.



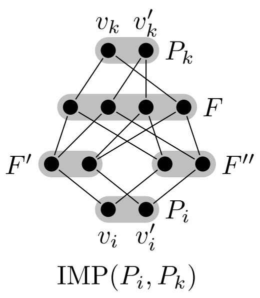


Figure 1 Gadgets used in the reduction of Theorem 4.2.

▶ **Claim 4.3.** Running color refinement on X_C after individualizing some input pairs will distinguish exactly those pairs P_k for which the gate g_k evaluates to 1 under the assignment that sets exactly those input gates to 1 whose corresponding pairs were initially individualized.

Let X'_C be the graph that is obtained from X_C by adding implication gadgets from Q to each pair P_i that corresponds to an input gate g_i. If C has a satisfying assignment $x \in \{0,1\}^n$ of weight k, individualizing the vertices v_i with $x_i = 1$ and subsequently running color refinement will assign distinct colors to all vertices of X_C. Indeed, the gadgets of X_C ensure that the pair Q becomes distinguished, the additional gadgets in X'_C propagate this to all input pairs P_i, and the gates of X_C in turn make sure that all remaining color classes become distinguished. Conversely, if C does not have a weight k satisfying assignment, there is no way to individualize k input vertices such that color refinement distinguishes Q. However, we already noted that there is no automorphism that transposes the output pair of the IMP(P_ℓ, Q) gadget, so no way of individualizing k input vertices makes X'_C refinable.

In X'_C, it always suffices to individualize one vertex from Q to make it discrete. To drop the assumption that each of the k individualized vertices must correspond to an input gate, we construct a graph X''_C. It consists of n input pairs $P_i = \{v_i, v'_i\}$ and n copies of X_C, to which we will refer to as $X_C^{(1)}, \ldots, X_C^{(n)}$. We also add the gadgets IMP$(P_i, P_i^{(j)})$ for all $i, j \in \{1, \ldots, n\}$ and the gadgets IMP$(Q^{(i)}, P_i)$ for all $i \in \{1, \ldots, n\}$. It is not hard to see that $\langle C, k\rangle \mapsto \langle X''_C, k\rangle$ is the desired reduction; see the full version [3] for its correctness. ◀

As a corollary to this proof we can derive the W[P]-hardness of the k-Color-Valence problem.

▶ **Corollary 4.4.** *k-Color-Valence is* W[P]*-hard.*

Proof. In the previous reduction we mapped instances of Weighted Monotone Circuit Satisfiability to instances of k-Discrete such that the given boolean circuit C has a satisfying assignment of weight k if and only if the resulting graph X''_C can be made discrete by individualizing k vertices. Note that individualizing k vertices in X''_C and subsequently running color refinement results in singleton color classes if and only if it brings the color valence down to 0. Thus, k-Color-Valence is W[P]-hard even for $d = 0$. ◀

4.1 Graphs of color class size at most 3

We call a vertex-colored graph *b-bounded* if all its color classes are of size at most b. In this section, we show that for any 3-bounded graph, we can compute in polynomial time the minimum number of vertices that have to be individualized so that the resulting colored graph becomes rigid, discrete, amenable, compact, or refinable. We will use the following two lemmas; their proofs can be found in the full version of this article [3].

▶ **Lemma 4.5.** *Let X be a 3-bounded graph whose color classes are stable. If $\mathrm{Aut}(X)$ restricted to any color class C_i of X is the full symmetric group on C_i, then X is compact.*

▶ **Lemma 4.6.** *Let X be a connected 3-bounded graph whose color classes are stable. If some $\sigma \in \mathrm{Aut}(X)$ is cyclic (i.e., σ acts cyclically on each color class C_i), then X is compact.*

▶ **Theorem 4.7.** *For any 3-bounded graph we can compute in polynomial time a vertex set S of minimum size such that individualizing (or fixing) all the vertices in S makes the graph discrete, amenable, compact, refinable (or rigid).*

Proof. Let $X = (V, E, c)$ be the given 3-bounded graph. We first compute the color partition $\{C_1, \dots, C_m\}$ of the stable coloring of X. We can assume that each induced graph $X_i = X[C_i]$ is empty and each induced bipartite graph $X_{ij} = X[C_i, C_j]$ has at most $|C_i| \cdot |C_j|/2$ edges, as otherwise we can complement these subgraphs. Since the partition $\{C_1, \dots, C_m\}$ is stable and the color classes have size at most 3, it follows that there are no edges between color classes having different sizes, and that between color classes C_i and C_j of the same size we either have a perfect matching or no edges at all.

We say that two color classes C_i and C_j are *linked* if there is a path between some vertex $u \in C_i$ and some vertex $v \in C_j$. Since this is an equivalence relation, it partitions the color classes into equivalence classes. This induces a partition $V = V_1 \sqcup \dots \sqcup V_l$ of the vertices such that each set V_i is a union of linked color classes having the same size and there are no edges between V_i and V_j whenever $i \neq j$. Hence, it suffices to solve the problem separately for each of the induced subgraphs $X[V_i]$.

If all color classes of $X[V_i]$ are of size 2, then $\mathrm{Aut}(X[V_i])$ contains exactly one non-trivial automorphism flipping all the color classes, implying that $X[V_i]$ is compact (see Lemma 4.5). In this case it suffices to individualize (or fix) an arbitrary vertex to make the graph discrete (or rigid). Further, $X[V_i]$ is already amenable if and only if it is a forest [4].

If all color classes of $X[V_i]$ are of size 3, then we compute its connected components as well as $\mathrm{Aut}(X[V_i])$ (which is even possible in logspace [18, 23]) and consider the following subcases.

- If $X[V_i]$ has 6 automorphisms (or, equivalently, consists of three connected components), then $X[V_i]$ is compact (see Lemma 4.5) and it suffices to individualize two vertices inside an arbitrary color class to make the graph discrete. On the other hand, if we individualize only one vertex, then the graph does not become discrete (not even rigid). Further, $X[V_i]$ is amenable if and only if it is a forest [4]. If $X[V_i]$ contains cycles then we need to individualize 2 vertices to make the graph amenable.
- If $X[V_i]$ has 3 automorphisms, then it follows that these automorphisms act cyclically on each color class and $X[V_i]$ is connected as well as compact (see Lemma 4.6). In this case it suffices to individualize an arbitrary vertex to make the graph discrete.
- If $X[V_i]$ has 2 automorphisms (or, equivalently, consists of two connected components), then $X[V_i]$ is not refinable and it suffices to individualize an arbitrary vertex in the larger of the two components to make the graph discrete.
- Finally, if $X[V_i]$ is rigid, then it follows that $X[V_i]$ is connected and not refinable. In this case it suffices to individualize an arbitrary vertex to make the graph discrete. ◀

We can actually strengthen Theorem 4.7 and show that these problems are in logspace. Since the case analysis in the proof can be done in logspace, it suffices to show that the stable color partition of a 3-bounded graph can be computed in logspace. The proof of this result is given in the full version of this article [3].

4.2 Bounded number of refinement steps

In this section, we consider (colored) graphs in which all color classes become singletons after ℓ rounds of color refinement. We denote the class of these graphs by DISCRETE[ℓ].

▶ **Theorem 4.8.** *The k-DISCRETE[ℓ] problem is* W[2]*-hard for any constant $\ell \geq 1$, even for uncolored and for 2-bounded graphs.*

Proof. We prove this by providing a reduction from the W[2]-complete problem DOMINATING SET that is inspired by [22, Theorem 7]. The input to this problem is a graph $X = (V, E)$ and a number k (treated as parameter) and the question is whether there exists a *dominating set* $D \subseteq V$ of size k in X, meaning that each vertex $v \in V \setminus D$ is adjacent to at least one vertex in D. We transform the DOMINATING SET instance (X, k) with $X = (V, E)$ into an equivalent instance (X', k) where $X = (V', E', c')$ for k-DISCRETE[ℓ]. For every $v \in V$, the colored graph X' contains the vertices $v_1, \dots, v_\ell$ and $v'_1, \dots, v'_\ell$ as well as the edges $\{v_i, v_{i+1}\}$ and $\{v'_i, v'_{i+1}\}$ for all i in $\{1, \dots, \ell - 1\}$. Furthermore, we add the edges $\{v_1, u_1\}$ and $\{v'_1, u'_1\}$ for every edge $\{u, v\}$ of X. We choose c' in such a way that for all $v \in V$ the set $\{v_1, v'_1\}$ is a color class and $c'(v_i) = c'(v'_i)$ for all $i \in \{2, \dots, \ell\}$.

Let D be a dominating set in X. Individualizing all the vertices v_1 in X' with $v \in D$ will distinguish the pairs $\{v_1, v'_1\}$ for all $v \in V$ after one round of color refinement. Thus after at most $\ell - 1$ more rounds all color classes of X' will be singletons.

For the converse direction, let I be a set of vertices in X', such that individualizing them and running ℓ rounds of color refinement produces singleton color classes. If I contains vertices v_i or v'_i for $i > 1$, we can replace them by v_1 and this still puts X' in DISCRETE[ℓ]. It is easy to see that this replacement does not decrease the number of color classes that become singletons after ℓ rounds. Hence, we can assume that I only contains vertices of the form v_1, implying that the set $D = \{v \in V \mid v_1 \in I\}$ is a dominating set of size at most $|I|$ in X. To see this it suffices to observe that the vertices u_ℓ and u'_ℓ can only be distinguished by color refinement within ℓ rounds if either u_1 is in I or u has a neighbor v for which v_1 is in I, implying that either u or some neighbor of u is in D.

For uncolored graphs we simulate the colors with degrees using a global gadget which in turn is distinguished from the rest by four special vertices of which two need to be individualized in any case. This increases the parameter from k to $k + 2$. See the full version [3] for details. ◀

5 The number of non-individualized vertices as parameter

In this section, we show that the problem $(n - k)$-DISCRETE is in FPT. In fact, we show a linear kernel and consequently, a $k^{O(k)} n^{O(1)}$ time algorithm for this problem.

▶ **Theorem 5.1.** *There exists a kernel of size $2k$ for $(n - k)$-DISCRETE that can be computed in polynomial time.*

We begin with some notation. Given a colored graph $X = (V, E, c)$, let S be a subset of vertices. Let $\mathcal{C}[S]$ denote the stable partition obtained by individualizing every vertex in $V \setminus S$ and performing color refinement. We denote the number of color classes in $\mathcal{C}[S]$ by $|\mathcal{C}[S]|$. We can partition the vertices u in $V \setminus S$ by their neighborhood $N(u) \cap S$ inside the set S. We denote this partition of $V \setminus S$ by $\mathcal{N}[S]$ and the number of sets in it by $|\mathcal{N}[S]|$. We call two vertices u and v *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. This relation is an equivalence relation and the corresponding equivalence classes are called *twin classes.* A graph is said to be *twin-free* if each twin class is of size 1.

The following lemma shows that sufficiently large twin-free graphs are YES instances of the $(n-k)$-DISCRETE problem.

▶ **Lemma 5.2.** *Let $X = (V, E)$ be a twin-free graph. Suppose $|V| > 2k$. There exists a set $S \subset V$ of size k such that $\mathcal{C}[S]$ is discrete. Moreover, we can compute such a set in $(nk)^{O(1)}$ time.*

Proof. We describe the algorithm for computing S. Initially, we pick an arbitrary subset $T_0 \subset V$ of size k and run color refinement to compute the stable partition $\mathcal{C}[T_0]$. Let $C_1, \dots, C_l$ be the color classes in $\mathcal{C}[T_0]$ that are contained in T_0. If $\mathcal{C}[T_0]$ is already discrete, we output the set $S = T_0$ and stop.

Otherwise we rename the color classes such that $|C_1| \geq |C_i|$ for $i = 2, \dots, l$. Then we compute the partition $\mathcal{N}[S] = \{B_1, \dots, B_m\}$ of $V \setminus S$, where we assume that $|B_1| \geq |B_i|$ for $i = 2, \dots, m$. If $m \geq k$, then we form S by picking an arbitrary vertex from each of the sets $B_1, \dots, B_k$. To see that $\mathcal{C}[S]$ is discrete it suffices to observe that individualizing all the vertices in $V \setminus S$ causes the separation of the sets $B_1, \dots, B_m$ and individualizing all but at most one vertex in each set B_i makes the graph discrete.

It remains to handle the case that $m < k$. We show that in this case it is possible to compute in polynomial time a set T_1 of size k such that $|\mathcal{C}[T_1]| > |\mathcal{C}[T_0]|$. By repeating this procedure $i \leq k-1$ times, we end up with a set T_i for which $\mathcal{C}[T_i]$ is discrete. Let u and v be two vertices inside the color-class C_1. Since X is twin-free, there must be a vertex a witnessing the fact that u and v are not twins. Since u and v have the same color, a cannot be individualized, implying that $a \in T_0$. Let C_j be the color class containing a. Since C_1 and C_j are stable color classes, there must exist a vertex $b \in C_j$ such that $\{u, a\}$ and $\{v, b\}$ are edges and $\{u, b\}$ and $\{v, a\}$ are non-edges. Clearly, individualizing a refines the color class C_1. Therefore, the set $T' = T_0 - \{a\}$ has the desired property $|\mathcal{C}[T']| > |\mathcal{C}[T_0]|$ but is of size $k-1$.

Since $|V| > 2k$ and $m < k$, it follows that $|B_1| \geq 2$. Let x and y be two vertices inside B_1. Since X is twin-free, there must be a vertex z witnessing the fact that x and y are not twins. Since all vertices in T_0 either have both vertices x and y as neighbors or none of them (otherwise, x and y would have different neighborhoods inside T_0, contradicting the fact that $x, y \in B_1$), it follows that $z \notin T_0$. We claim that the set $T_1 = T' \cup \{z\}$ yields the same stable partition as T', i.e., $\mathcal{C}[T_1] = \mathcal{C}[T']$. In fact, color refinement anyway assigns a unique color to z, since it is the only non-individualized vertex that is adjacent to exactly one of the two individualized vertices x and y. This completes the proof of the lemma. ◀

Proof of Theorem 5.1. We outline a simple kernelization algorithm for $(n-k)$-DISCRETE. Let X be the given graph and let k be the given parameter. The algorithm first makes the graph X twin-free by removing all but one vertex from each twin-class.

If the resulting graph X' has at most $2k$ vertices, it outputs the instance (X', k) as the kernel. Since in each twin class of X, all but one vertices have to be individualized to make the graph discrete, the two instances (X, k) and (X', k) are indeed equivalent with respect to the $(n-k)$-DISCRETE problem.

If X' has more than $2k$ vertices, the algorithm computes in polynomial time a set S of size k such that individualizing every vertex outside of S makes the graph X' discrete (see Lemma 5.2). Clearly this set S is also a solution for X, so the kernelization algorithm can output a trivial YES instance. ◀

Acknowledgements. We thank the anonymous referees for their helpful comments.

References

1 Karl A. Abrahamson, Rodney G. Downey, and Michael R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73(3):235–276, 1993. doi:10.1016/0168-0072(94)00034-Z.

2 V. Arvind. The parameterized complexity of fixpoint free elements and bases in permutation groups. In *Proceedings of 8th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 4–15. Springer, 2013. doi:10.1007/978-3-319-03898-8_2.

3 V. Arvind, Frank Fuhlbrück, Johannes Köbler, Sebastian Kuhnert, and Gaurav Rattan. The parameterized complexity of fixing number and vertex individualization in graphs. arXiv:1606.04383, 2016.

4 V. Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. On the power of color refinement. In *Proceedings of 20th International Symposium Fundamentals of Computation Theory (FCT)*, pages 339–350. Springer, 2015. doi:10.1007/978-3-319-22177-9_26.

5 V. Arvind, Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky. On Tinhofer's linear programming approach to isomorphism testing. In *Proceedings of 40th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 26–37. Springer, 2015. doi:10.1007/978-3-662-48054-0_3.

6 László Babai. Graph Isomorphism in quasipolynomial time. arXiv:1512-03547, 2015.

7 László Babai and Eugene M. Luks. Canonical labeling of graphs. In *Proceedings of 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 171–183, 1983. doi:10.1145/800061.808746.

8 Robert F. Bailey and Peter J. Cameron. Base size, metric dimension and other invariants of groups and graphs. *Bulletin of the London Mathematical Society*, 43(2):209–242, 2011. doi:10.1112/blms/bdq096.

9 Kenneth D. Blaha. Minimum bases for permutation groups: The greedy approximation. *Journal of Algorithms*, 13(2):297–306, 1992. doi:10.1016/0196-6774(92)90020-D.

10 Debra L. Boutin. Identifying graph automorphisms using determining sets. *Electronic Journal of Combinatorics*, 13:R78, 2006. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v13i1r78.

11 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.

12 Liming Cai and David Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789–807, 2003. doi:10.1016/S0022-0000(03)00074-6.

13 John D. Dixon and Brian Mortimer. *Permutation groups*. Springer, 1996. doi:10.1007/978-1-4612-0731-3.

14 Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto, and Frances A. Rosamund. Cutting up is hard to do: The parameterised complexity of k-cut and related problems. *Electronic Notes in Theoretical Computer Science*, 78:209–222, 2003. doi:10.1016/S1571-0661(04)81014-4.

15 David Erwin and Frank Harary. Destroying automorphisms by fixing nodes. *Discrete Mathematics*, 306(24):3244–3252, 2006. doi:10.1016/j.disc.2006.06.004.

16 Gašper Fijavž and Bojan Mohar. Rigidity and separation indices of paley graphs. *Discrete Mathematics*, 289(1-3):157–161, 2004. doi:10.1016/j.disc.2004.09.004.

17 Martin Grohe. Equivalence in finite-variable logics is complete for polynomial time. *Combinatorica*, 19(4):507–532, 1999. doi:10.1007/s004939970004.

18 Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, 66(3):549–566, 2003. doi:10.1016/S0022-0000(03)00042-4.

19 Eugene M. Luks. Permutation groups and polynomial-time computation. In *Groups and Computation*, pages 139–175. American Mathematical Society, 1993. URL: http://www.cs.uoregon.edu/~luks/dimacs.pdf.

20 Rudolf Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979. doi:10.1016/0020-0190(79)90004-8.

21 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.

22 Joanna Raczek. Distance paired domination numbers of graphs. *Discrete Mathematics*, 308(12):2473–2483, 2008. doi:10.1016/j.disc.2007.05.018.

23 Omer Reingold. Undirected st-connectivity in log-space. In *Proceedings of 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 376–385, 2005. doi:10.1145/1060590.1060647.

24 Pascal Schweitzer. Isomorphism of (mis)labeled graphs. In *Proceedings of 19th European Symposium on Algorithms (ESA)*, pages 370–381, Berlin, 2011. Springer. doi:10.1007/978-3-642-23719-5_32.

25 Gottfried Tinhofer. A note on compact graphs. *Discrete Applied Mathematics*, 30(2–3):253–264, 1991. doi:10.1016/0166-218X(91)90049-3.

26 Luca Trevisan. Non-approximability results for optimization problems on bounded degree instances. In *Proceedings of 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 453–461. ACM, 2001. doi:10.1145/380752.380839.

27 Viktor N. Zemlyachenko, Nikolay M. Kornienko, and Regina I. Tyshkevich. Graph isomorphism problem. *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta*, 118:83–158, 1982. Russian. Translation to English: [28].

28 Viktor N. Zemlyachenko, Nikolay M. Kornienko, and Regina I. Tyshkevich. Graph isomorphism problem. *Journal of Mathematical Sciences*, 29(4):1426–1481, 1985. English translation of [27]. doi:10.1007/BF02104746.

Real Interactive Proofs for VPSPACE [*]

Martijn Baartse[1] and Klaus Meer[2]

1 Computer Science Institute, BTU Cottbus-Senftenberg
Platz der Deutschen Einheit 1
D-03046 Cottbus, Germany

2 Computer Science Institute, BTU Cottbus-Senftenberg
Platz der Deutschen Einheit 1
D-03046 Cottbus, Germany
meer@b-tu.de

Abstract

We study interactive proofs in the framework of real number complexity as introduced by Blum, Shub, and Smale. The ultimate goal is to give a Shamir like characterization of the real counterpart $\mathrm{IP}_{\mathbb{R}}$ of classical IP. Whereas classically Shamir's result implies $\mathrm{IP} = \mathrm{PSPACE} = \mathrm{PAT} = \mathrm{PAR}$, in our framework a major difficulty arises from the fact that in contrast to Turing complexity theory the real number classes $\mathrm{PAR}_{\mathbb{R}}$ and $\mathrm{PAT}_{\mathbb{R}}$ differ and space resources considered alone are not meaningful. It is not obvious to see whether $\mathrm{IP}_{\mathbb{R}}$ is characterized by one of them - and if so by which.

In recent work the present authors established an upper bound $\mathrm{IP}_{\mathbb{R}} \subseteq \mathrm{MA}\exists\mathbb{R}$, where $\mathrm{MA}\exists\mathbb{R}$ is a complexity class satisfying $\mathrm{PAR}_{\mathbb{R}} \subsetneq \mathrm{MA}\exists\mathbb{R} \subseteq \mathrm{PAT}_{\mathbb{R}}$ and conjectured to be different from $\mathrm{PAT}_{\mathbb{R}}$. The goal of the present paper is to complement this result and to prove interesting lower bounds for $\mathrm{IP}_{\mathbb{R}}$. More precisely, we design interactive real protocols for a large class of functions introduced by Koiran and Perifel and denoted by $\mathrm{UniformVPSPACE}^0$. As consequence, we show $\mathrm{PAR}_{\mathbb{R}} \subseteq \mathrm{IP}_{\mathbb{R}}$, which in particular implies $\text{co-NP}_{\mathbb{R}} \subseteq \mathrm{IP}_{\mathbb{R}}$, and $\mathrm{P}_{\mathbb{R}}^{Res} \subseteq \mathrm{IP}_{\mathbb{R}}$, where *Res* denotes certain multivariate Resultant polynomials.

Our proof techniques are guided by the question in how far Shamir's classical proof can be used as well in the real number setting. Towards this aim results by Koiran and Perifel on $\mathrm{UniformVPSPACE}^0$ are extremely helpful.

1998 ACM Subject Classification F.1.1 Models of Computation; F 1.2 Modes of Computation; F 1.3 Complexity Measures and Classes

Keywords and phrases Interactive Proofs, real number computation, Shamir's theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.14

1 Introduction

Shamir's famous theorem [18] characterizes the set IP of languages that can be verified by an interactive protocol performed between a polynomial time probabilistic verifier and a prover of unlimited power as being equal to PSPACE.

Around the same time of Shamir's result Blum, Shub, and Smale [5] introduced a model of computation over the real numbers (for short: BSS model in the sequel) and a complexity theory for it. Since then, among other things one line of activity in research on the BSS model was to figure out whether and by what reasons important classical results in Turing

[*] Both authors were partially supported under projects ME 1424/7-1 and ME 1424/7-2 by the Deutsche Forschungsgemeinschaft DFG. We gratefully acknowledge the support.

complexity theory hold as well over other computational structures. Following this line, in the present paper we are interested in deriving results about the real class $\mathrm{IP}_{\mathbb{R}}$ of languages verifiable by an interactive protocol over the reals; for precise definitions see next section.

It is well known that over the reals complexity classes that are classically defined or characterized using space resources turn out to have a more subtle relation among each other than they do classically. Taken alone, space resources have no meaning at all; each decision problem can be decided in linear space using an elementary coding trick [15]. As consequence, for many equivalent characterizations especially of the class PSPACE in classical complexity it is unclear what they should become in the real number framework. Recall that PAR, PSPACE, PAT, and IP, denoting the classes of languages acceptable in parallel polynomial time with exponentially many processors, in polynomial space, in polynomial alternating time, and by interactive proofs, respectively, all are the same in Turing complexity; see the textbook [1] for references and proofs. In contrast, over $\mathbb{R}$ it is known that the first three classes mentioned above satisfy $\mathrm{PAR}_{\mathbb{R}} \subsetneq \mathrm{PSPACE}_{\mathbb{R}} \subseteq \mathrm{PAT}_{\mathbb{R}}$, where $\mathrm{PSPACE}_{\mathbb{R}}$ denotes the class of real decision problems decidable by an algorithm using both exponential time and polynomial space and the other two classes are defined by extending the classical definitions straightforwardly, see [7, 4]. As a consequence, if a new class like $\mathrm{IP}_{\mathbb{R}}$ is studied which classically gives yet another characterization of PSPACE via Shamir's result, it is not obvious where it can be located over the reals.

It is straightforward to see from the definitions that $\mathrm{NP}_{\mathbb{R}} \subseteq \mathrm{IP}_{\mathbb{R}}$. But already the inclusion co-$\mathrm{NP}_{\mathbb{R}} \subseteq \mathrm{IP}_{\mathbb{R}}$ is far from being obvious. Shamir's proof designs an interactive protocol for the PSPACE-complete Quantified Boolean Formulas problem QBF roughly as follows: First, the problem is arithmetized in form of giving a short formula representing an algebraic expression with exponentially many terms. This expression replaces Boolean quantifiers in the original formula by sums and products, respectively, in which the quantified variables run through all values in $\{0, 1\}$. The goal of the communication protocol is to evaluate this expression interactively and randomly. Towards this aim, certain canonical univariate polynomials are attached to this expression by eliminating one after the other the leftmost operator $\sum\limits_{x_i=0}^{1}$ or $\prod\limits_{x_i=0}^{1}$ in it. This results in a polynomial in x_i of polynomial degree whose value in a random point is verified interactively. Though a real variant of QBF can easily be defined and is complete for $\mathrm{PAT}_{\mathbb{R}}$, Shamir's proof cannot be transformed. An arithmetization of quantifiers ranging over the reals is not possible in the same way and immediately destroys the hope of following the above approach; however, see [2] for some attempts dealing with more restricted real decision problems.

In this paper we shall therefore follow a different approach. We still are guided by the question how far Shamir's technique might lead. Instead of dealing directly with an arithmetization of computationally hard real number problems we rely on results in BSS theory that figure out how much can be done using certain oracles in real computations. Such results have been obtained by several authors; crucial for us is work by Koiran and Perifel [12]. Therein a relatively huge class $\mathrm{UniformVPSPACE}^0$ of families of polynomial functions is introduced and studied. It is a kind of uniform extension of Valiant's class VNP and covers families of polynomials with exponential degree and integer coefficients computable in PSPACE. Crucial for us will be two observations: Firstly, verifying whether the result $f_n(x)$ of a member f_n of such a family on input x equals a given y can be done within the resources of $\mathrm{IP}_{\mathbb{R}}$. This result is obtained by showing that the discrete techniques used by Shamir are sufficient to deal with $\mathrm{UniformVPSPACE}^0$. This is important in order to circumvent the above mentioned problems. Secondly, as shown in [12] $\mathrm{UniformVPSPACE}^0$ is powerful enough to deal with interesting real number problems in classes like co-$\mathrm{NP}_{\mathbb{R}}$

and larger via polynomial time BSS algorithms that access an oracle for function families in UniformVPSPACE^0. That way, a real verifier can be designed that is able to deal with problems even from class $\mathrm{PAR}_{\mathbb{R}}$. This leads to our main result stating $\mathrm{PAR}_{\mathbb{R}} \subseteq \mathrm{IP}_{\mathbb{R}}$. Taking into account previous results this will locate $\mathrm{IP}_{\mathbb{R}}$ much better within certain real number classes than it has been possible so far. It shows as well that also over the reals $\mathrm{IP}_{\mathbb{R}}$ under standard complexity theoretic assumptions is considerably larger than $\mathrm{NP}_{\mathbb{R}}$.

1.1 Previous results

Before summarizing previous results let us recall the formal definition of algebraic circuits and the class $\mathrm{PAR}_{\mathbb{R}}$. An algebraic circuit is a connected and directed acyclic graph having nodes either of indegree 0 (input nodes) , 1 (test nodes) or 2 (computation nodes). Nodes of indegree 2 are labelled with one of the operations $+, -, \bullet$, nodes of indegree 1 are labelled with $' \geq 0?'$ A circuit has one output node of outdegree 0. The size of a circuit is the number of its nodes, its depth is the length of the longest path from an input node to the output node. A circuit with n input nodes computes in the straightforward manner a function from $\mathbb{R}^n \mapsto \mathbb{R}$; on input $x \in \mathbb{R}^n$ it propagates values along the labels of nodes in the obvious way. The value of a test node is either 1 or 0, depending on whether its incoming value is ≥ 0 or not. We only consider circuits with one output node which is a test node. Thus, our circuits compute characteristic functions.

▶ **Definition 1.** A probem $L \subseteq \mathbb{R}^\infty := \bigsqcup_{i \geq 1} \mathbb{R}^i$ belongs to class $\mathrm{PAR}_{\mathbb{R}}$ iff there exists a family $\{C_n\}_{n \in \mathbb{N}}$ of algebraic circuits of depth polynomially bounded in n, a constant $s \in \mathbb{N}$, and a vector $c \in \mathbb{R}^s$ of real constants such that

(i) each C_n has $n + s$ input nodes;
(ii) for all $n \in \mathbb{N}$ the circuit C_n computes the characteristic function of $L \cap \mathbb{R}^n$, when the last s input nodes are assigned the constant values from c, i.e., $x \in L \cap \mathbb{R}^n \Leftrightarrow C_n(x, c) = 1$;
(iii) the family $\{C_n\}_n$ is PSPACE uniform, i.e., there is a Turing machine working in polynomial space which for each $n \in \mathbb{N}$ computes a description of C_n.

If no constant vector c is involved we obtain the constant free version of $\mathrm{PAR}_{\mathbb{R}}$ denoted by $\mathrm{PAR}^0_{\mathbb{R}}$.

The above definition basically is from [6]. There are equivalent ones explicitly involving BSS machines [4]. The vector c used above then plays the role of the machine constants of such a BSS algorithm.

There has so far not been much work on interactive proofs in the BSS model. It started with a paper by Ivanov and de Rougemont [11] where Shamir's result was shown to hold as well in the additive real number model. In this model, multiplications are not allowed. The interaction was restricted to exchanging bits. One side result in this paper was that the classes $\mathrm{PAR}_{\mathbb{R}}$ and $\mathrm{IP}_{\mathbb{R}}$ are provably different [1].

In [2] the real class $\mathrm{IP}_{\mathbb{R}}$ was introduced. The above mentioned problem in $\mathrm{IP}_{\mathbb{R}} \setminus \mathrm{PAR}_{\mathbb{R}}$ from [11] is one that can be formalized by using polynomially alternating existential quantifications over the reals and arbitrary Boolean quantifiers. It is therefore kind of natural trying to relate $\mathrm{IP}_{\mathbb{R}}$ to another real complexity class $\mathrm{MA}\exists\mathbb{R}$ introduced and studied by Cucker and

[1] in [11] $\mathrm{IP}_{\mathbb{R}}$ is not introduced formally, but it is shown that there exists a problem not in $\mathrm{PAR}_{\mathbb{R}}$ but within a class that easily is seen to be a subclass of $\mathrm{IP}_{\mathbb{R}}$ as defined below. However, this example does NOT show $\mathrm{PAR}_{\mathbb{R}} \subseteq \mathrm{IP}_{\mathbb{R}}$

Briquel in [8]. It is a class which does not make sense over finite alphabets and can be located between $\mathrm{PSPACE}_{\mathbb{R}}$ and $\mathrm{PAT}_{\mathbb{R}}$.

▶ **Definition 2.** ([8]) The class $\mathrm{MA}\exists\mathbb{R}$ consists of all decision problems $A \subseteq \mathbb{R}^{\infty}$ for which there exists a problem $L \in \mathrm{P}_{\mathbb{R}}$ together with a polynomial p such that an $x \in \mathbb{R}^{\infty}$ belongs to A if and only if the following formula holds:

$$\forall_B z_1 \exists_{\mathbb{R}} y_1 \ldots \forall_B z_{p(|x|)} \exists_{\mathbb{R}} y_{p(|x|)} (x, y, z) \in L \ .$$

Here, $y = (y_1, \ldots, y_{p(|x|)})$ and $z = (z_1, \ldots, z_{p(|x|)})$. The subscripts $B, \mathbb{R}$ for the quantifiers indicate whether a quantified variable ranges over $B := \{0, 1\}$ or $\mathbb{R}$, respectively.

Note that $\mathrm{MA}\exists\mathbb{R}$ contains the real polynomial hierarchy $\mathrm{PH}_{\mathbb{R}}$, i.e., problems with a fixed number of both existential and universal real quantifiers and even its supclasses $\mathrm{PAR}_{\mathbb{R}} \subsetneq \mathrm{PSPACE}_{\mathbb{R}}$, see [8]. It is not known to capture $\mathrm{PAT}_{\mathbb{R}}$; however, $\mathrm{MA}\exists\mathbb{R}$ reflects the special structure of quantifiers mentioned above in relation to the problem that witnesses $\mathrm{PAR}_{\mathbb{R}} \neq \mathrm{IP}_{\mathbb{R}}$. In fact, we have

▶ **Theorem 3.** *([2])* $\mathrm{IP}_{\mathbb{R}} \subseteq \mathrm{MA}\exists\mathbb{R}$.

The main purpose of this paper is to complement this upper bound result by obtaining non-trivial lower bounds for $\mathrm{IP}_{\mathbb{R}}$ as well. In Section 2 we introduce the main concepts and define $\mathrm{IP}_{\mathbb{R}}$ and $\mathrm{UniformVPSPACE}^0$. Section 3 gives the result showing that all function families in $\mathrm{UniformVPSPACE}^0$ can be evaluated interactively within $\mathrm{IP}_{\mathbb{R}}$. The final section applies this theorem to prove our main result, namely that some further real complexity classes are included in $\mathrm{IP}_{\mathbb{R}}$, $\mathrm{PAR}_{\mathbb{R}}$ being the most interesting among them.

One remark concerning the contribution of this paper seems in charge. There is not one big new technical result presented here. Different variants of Theorem 10 below have been known before, see [16] and [14]; we present the proof again because of self-containment and because reformulating the results in the cited papers in the way we need them would not save much space. We believe the value of the present paper is the combination of several pieces of previous works in a way that has not been done so far. This in particular refers to using the class $\mathrm{UniformVPSPACE}^0$ in relation with interactive proofs and realizing that discrete techniques are sufficient to deal with it in a certain sense. That way, we obtain the strongest result on real interactive proofs so far. This result in our opinion definitely is interesting by itself and means significant progress concerning the question how a seminal result of classical complexity theory looks like in the real number framework.

2 Basic notions and results

In this section we recall the definitions of the two main complexity classes considered in this paper, namely $\mathrm{IP}_{\mathbb{R}}$ as well as $\mathrm{UniformVPSPACE}^0$. The former was defined in [2], the latter in [12].

2.1 The model for interaction and some variants

As underlying algorithm model we work in the Blum-Shub-Smale BSS model over $\mathbb{R}$ [4, 5]. Decision problems considered in this model are subsets of $\mathbb{R}^{\infty} := \bigsqcup_{i \geq 1} \mathbb{R}^i$. The model allows to perform the basic arithmetic operations $+, -, \bullet$ and test instructions of the form 'is $x \geq 0$?' at unit cost; an $x \in \mathbb{R}^i$ has (algebraic) size i. Below, in addition we allow both the verifier and the prover to exchange real numbers at unit cost.

The prover P is a BSS machine unlimited in computational power. The verifier V is a randomized polynomial time BSS algorithm. It is important to point out that randomization (still) is discrete, i.e., V generates a sequence of random bits $r = (r_1, r_2, \ldots)$ during its computation. The computation proceeds as follows:

- Given an input $x \in \mathbb{R}^n$ of size $|x| = n$ and (some of) the random bits of r the verifier V computes a real $V(x, r) =: w_1 \in \mathbb{R}$ and sends it to P;
- using x and w_1 the prover P sends a real $P(x, w_1) =: p_1 \in \mathbb{R}$ back to V;
- let $(w_1, p_1, w_2, \ldots, p_i)$ denote the information sent forth and back after i rounds, then in round $i + 1$ V computes a real $V(x, r, w_1, p_1, \ldots, p_i) =: w_{i+1}$ and sends it to P; P then computes a real $P(x, w_1, p_1, \ldots, p_i, w_{i+1}) =: p_{i+1}$ and sends it to V;
- the communication halts after a polynomial number $m = poly(|x|)$ of rounds. Then V computes its final result $V(x, r, w_1, \ldots, p_{m-1}) =: w_m \in \{0, 1\}$ representing its decision to reject or accept the input, respectively.

We denote the result of an interaction between V and P on input x and V using r as random string by $(P, V)(x, r)$. All computations by V have to be finished in (real) polynomial time; thus, in particular the number of random bits generated as well as the number of rounds is polynomially bounded in the size $|x|$ of x.

▶ **Definition 4.** a) A language $L \subseteq \mathbb{R}^\infty$ has an interactive protocol if there exists a polynomial time randomized verifier V such that
(i) if $x \in L$ there exists a prover P such that $\Pr\limits_{r \in \{0,1\}^*} \{(P, V)(x, r) = 1\} \geq \frac{2}{3}$ and
(ii) if $x \notin L$, then for all provers P it holds $\Pr\limits_{r \in \{0,1\}^*} \{(P, V)(x, r) = 1\} \leq \frac{1}{3}$.
Above, the length of r can be polynomially bounded in the length of x.
b) The class $\mathrm{IP}_\mathbb{R}$ contains all $L \subseteq \mathbb{R}^\infty$ which have an interactive protocol.

In the above definitions private coins are used, i.e., we do not allow the prover to know the outcome of V's random choices. One could change this requirement and let the verifier only send the random bits; what the verifier computes out of it then could be as well computed by the allmighty prover. Such protocols are called Arthur-Merlin protocols. Another modification uses one-sided instead of two-sided error in the acceptance condition for V. Then, for $x \in L$ there must be a prover such that V accepts with probability 1. For sake of completeness we show below that these modifications do not change the class $\mathrm{IP}_\mathbb{R}$. Both the result and its proof are the same as in the Turing model.

▶ **Definition 5.** The class $\widetilde{\mathrm{IP}_\mathbb{R}}$ is defined similar to $\mathrm{IP}_\mathbb{R}$, but with the following modifications:
(i) The verifier V uses *public coins*, i.e., it only sends the random bits r generated in each round to P.
(ii) The verifier accepts with one-sided error: A language L is in $\widetilde{\mathrm{IP}_\mathbb{R}}$ iff there is a verifier V such that $\forall x \in L$ there exists a prover P such that $\Pr\limits_{r \in \{0,1\}^*} \{(P, V)(x) = 1\} = 1$. And $\forall x \notin L$ $\forall P$ it holds $\Pr\limits_{r \in \{0,1\}^*} \{(P, V)(x) = 1\} \leq \frac{1}{3}$.

▶ Proposition 6. $\mathrm{IP}_\mathbb{R} = \widetilde{\mathrm{IP}_\mathbb{R}}$.

Proof. [2] The inclusion $\widetilde{\mathrm{IP}_\mathbb{R}} \subseteq \mathrm{IP}_\mathbb{R}$ being clear let $L \in \mathrm{IP}_\mathbb{R}$ and let V be a corresponding

[2] The fact that public coins are as powerful as private ones was first shown in [10]. An easier proof that also replaces two-sided by one-sided error was given by J. Kilian. We could not figure out whether the proof was published, it is however refered to in [9]. For sake of completeness we follow this proof below.

verifier accepting L with private coins and two-sided error. Without loss of generality in each communication round V generates one random bit. A new verifier $\tilde{V}$ using public coins and accepting L with one-sided error is obtained as follows. On input x $\tilde{V}$ expects from a prover to provide information about the communication between V and an optimal prover for it. More precisely, define a protocol tree T coding the protocol between V and an optimal prover on x as follows. An edge in T represents one communication round after a coin toss has been made by V. Since one bit is generated in each round T is binary, the outgoing edges of each node represent the communication for results 0 and 1, respectively. For m communication rounds the probability that V accepts x is $1/2^m$ times the number of accepting paths.

In its communication on x with a prover the new verifier $\tilde{V}$ descends a path of T top down as follows. Let r be the current node of T traversed, r_1, r_2 its left and right child, respectively. $\tilde{V}$ asks the prover for the numbers R, R_1, R_2 of accepting paths the communication between an optimal prover and V would generate when starting in r, r_1, and r_2, respectively. If r is the root and the number reported by the prover is $< \frac{2}{3} \cdot 2^m$ the verifier rejects right away. For an arbitrary node r it checks whether $R = R_1 + R_2$ and rejects if the equation is violated. Otherwise, $\tilde{V}$ moves to r_i with probability $\frac{R_i}{R}$ for $i = 1, 2$. The protocol continues until a leaf is reached. If the path traversed is accepting for the protocol followed by V, then $\tilde{V}$ accepts, otherwise it rejects.

$\tilde{V}$ obviously uses public coins; its random decisions are known to the prover because it is informed about the child of r that is picked by $\tilde{V}$. To see that $\tilde{V}$ accepts L with one-sided error first note that for $x \in L$ an optimal prover will always give the correct numbers R, R_1, R_2 and thus $\tilde{V}$ ends with probability 1 in an accepting leaf because there must exist such a leaf in T. Let us then assume $x \notin L$ and let P be an arbitrary prover.

Claim: For each node r in T the following holds: if there are R accepting paths from r on for an optimal prover and V, but the current prover P claims there are $R' > R$ accepting paths, then $\tilde{V}$ will realize an error with probability $\geq 1 - \frac{R}{R'}$.

Proof of claim: By induction on the height h of r. Let $h = 1$ and let r have children r_1, r_2 being leaves. If $R = 0$, then no matter whether $R' = 1$ or $R' = 2$ both paths are rejecting and $\tilde{V}$ realizes it with probability $1 = 1 - \frac{0}{R'}$. If $R = 1$ then $R' = 2$ and $\tilde{V}$ chooses the rejecting path with probability $\frac{1}{2} = 1 - \frac{1}{2}$.

For arbitrary h let the correct number of accepting paths from r, r_1, r_2 on be R, R_1, R_2, respectively. Let R', R'_1, R'_2 denote the (larger) numbers claimed by P. According to the induction hypothesis if the protocol starts in r_i the verifier $\tilde{V}$ realizes an error with probability $\geq (1 - \frac{R_i}{R'_i}), i = 1, 2$. In node r it chooses the left child with probability $\frac{R'_1}{R'}$ and the right one with probability $\frac{R'_2}{R'}$. The error probability thus is $(1 - \frac{R_1}{R'_1}) \cdot \frac{R'_1}{R'} + (1 - \frac{R_2}{R'_2}) \cdot \frac{R'_2}{R'} = 1 - \frac{R}{R'}$.

Finally, each $x \notin L$ is rejected by the original verifier V and any prover with probability $\geq \frac{2}{3}$, i.e., at most $\frac{1}{3}$ of all paths starting at the root of T are accepting. $\tilde{V}$ either rejects directly if the prover claims $R < \frac{2}{3} \cdot 2^m$ accepting paths or it rejects with probability $\geq 1 - \frac{1/3}{2/3} = \frac{1}{2}$ by the claim. Running the protocol for $\tilde{V}$ once more increases this probability to at least $\frac{3}{4} > \frac{2}{3}$ as required. ◻

2.2 UniformVPSPACE^0

The following class of functions was introduced and studied by Koiran and Perifel in [12] and kind of generalizes the famous Valiant class VNP. Informally, it consists of uniform families of polynomials with integer coefficients which depend on polynomially many variables, potentially an exponential degree and whose coefficients can be computed in PSPACE. Though originally defined over arbitrary fields we restrict ourselves to the real numbers.

▶ **Definition 7.** (see [12])

(a) A family $\{f_n\}_{n\in\mathbb{N}}$ of real polynomials belongs to UniformVPSPACE^0 iff the following conditions are satisfied: There exists a polynomial p such that

  (i) each f_n depends on $u(n)$ variables, where $u(n)$ is bounded from above by $p(n)$;

  (ii) the total degree of each f_n is bounded by $2^{p(n)}$;

  (iii) the coefficients of each f_n are integers which are bounded in their bitsize by $2^{p(n)}-1$;

  (iv) the coefficient function a is PSPACE computable. More precisely, a gets as arguments triples (n,α,i), where $n\in\mathbb{N}$ is given in unary, $\alpha=(\alpha_1,\ldots,\alpha_{u(n)})$ is a list of binary numbers representing a monomial $x^\alpha=x_1^{\alpha_1}\cdot x_2^{\alpha_2}\cdot\ldots\cdot x_{u(n)}^{\alpha_{u(n)}}$, and i is a binary number. Then $a(n,\alpha,i)\in\{0,1\}$ gives the i-th bit of the coefficient of monomial x^α in f_n. In particular, the value $a(n,\alpha,0)$ gives the sign of this monomial. [3]

The functions f_n thus have the following representation:

$$f_n(x_1,\ldots,x_{u(n)})=\sum_\alpha\left[(-1)^{a(n,\alpha,0)}\left(\sum_{i=1}^{2^{p(n)}}2^{i-1}a(n,\alpha,i)\right)x^\alpha\right].$$

(b) A family $\{f_n\}_{n\in\mathbb{N}}$ of polynomials belongs to class UniformVPAR iff it can be computed by a PSPACE-uniform family of arithmetic circuits of polynomial depth, compare Definition 1. If the family is constant free we obtain class UniformVPAR^0.

Again, the superscript '0' indicates that a class is defined without involving additional real constants. It is relatively straightforward to see that both notions above characterize the same set of families:

▶ **Lemma 8.** *([12]) It holds* UniformVPSPACE^0 = UniformVPAR^0.

The result implies in particular that if a family of functions $\{f_n\}\in$ UniformVPAR is defined by a family of circuits using a constant vector $c\in\mathbb{R}^s$, then one obtains another family of functions $\{g_n\}\in$ UniformVPSPACE^0 such that for all $x\in\mathbb{R}^{u(n)}$ of suitable input size we have $g_n(x,c)=f_n(x)$. This will be needed below.

3 Lower bound for IP$_\mathbb{R}$

In this section we prove our main technical theorem. Basically it shows that function families in UniformVPSPACE^0 can be represented by certain formulas having a very particular structure. The latter strongly resembles the structure of formulas arising via arithmetization of discrete quantified boolean formulas as outlined in the introduction. Of course, the new kind of formulas involve real variables. The special structure obtained allows to verify the values of such functions in a way similar to Shamir's original interactive proof for the QBF problem. Using additional results about class UniformVPSPACE^0 then makes it possible to derive real interactive proofs for interesting real number problems, foremost for all problems in PAR$_\mathbb{R}$.

We start with the definition of these specially structured formulas.

▶ **Definition 9.** Let $x_1,x_2,\ldots$ be a countable set of variable symbols.

(a) A *binary polynomial formula* over the reals is a formula p which can be built in finitely many steps according to the following rules:

[3] Note that since a only attains values in $\{0,1\}$ it can be seen as decision problem and thus the PSPACE requirement makes sense.

- **(i)** $p = 1$ and $p = x_i$ for $i = 1, 2, \ldots$ are binary polynomial formulas ;
- **(ii)** if p_1, p_2 are binary polynomial formulas, then so are $p_1 + p_2, p_1 - p_2, p_1 \cdot p_2$;
- **(iii)** if p is a binary polynomial formula depending freely on x_i, then both $\sum\limits_{x_i \in \{0,1\}} p(\ldots, x_i, \ldots)$ and $\prod\limits_{x_i \in \{0,1\}} p(\ldots, x_i, \ldots)$ are binary polynomial formulas (with x_i bounded by summation and multiplication, respectively).

All formulas - and no others - that can be obtained in finitely many steps applying the rules i) to iii) are binary polynomials formulas.

(b) The size of a binary polynomial formula is defined as the number of construction steps used in a) to generate it.

(c) A binary polynomial formula p in the canonical way represents a real polynomial function. It depends on the free variables, i.e., on those x_i that have been introduced via rule i) but have not been bound by a Boolean summation or multiplication applying rule iii).

The following theorem shows that families of functions in UniformVPSPACE^0 are basically the same as families of polynomials given via uniform families of binary polynomial formulas. Similar statements in different variants are already in [16] and [14]. We present the proof for sake of self-containment and because reformulating the results of those papers in the way we need them would likely not save much space.

▶ **Theorem 10.** *Let $\{f_n\}_n$ be a family of polynomial functions. Then $\{f_n\}_n$ belongs to class* UniformVPSPACE^0 *if and only if there exists a polynomial time Turing algorithm which on input $n \in \mathbb{N}$ (in unary) computes a binary polynomial formula p_n which represents f_n. By computing p_n we mean that the algorithm computes a scheme how to generate p_n according to the steps defined above.*

Proof. For the if-direction let $\{p_n\}_n$ be a family of binary polynomial formulas which are uniformly generated by a polynomial time Turing machine. Then it is easy to see that p_n can be computed by a PSPACE-uniform family of arithmetic circuits of polynomial depth. Since the formulas only involve the constant 1 the circuits are constant-free as well. The summation and multiplication operators in a formula can be simulated in parallel by the circuit, thus the polynomially many construction steps for the formula result in a polynomial depth for the circuit. It follows that the polynomial family $\{p_n\}_n$ belongs to class UniformVPAR^0. Lemma 8 now implies the 'if'-direction.

For the only-if direction, let a family $\{f_n\}_n \in$ UniformVPSPACE^0 be given and consider one of its members

$$f_n(x_1, \ldots, x_{u(n)}) = \sum_{\alpha} \left[(-1)^{a(n,\alpha,0)} \left(\sum_{i=1}^{2^{p(n)}} 2^{i-1} a(n, \alpha, i) \right) x^{\alpha} \right].$$

Without loss of generality we assume $u = p$. Our task is to show that the different parts in this representation can be rewritten in form of binary polynomial formulas.

Step 1: Let us start with constructing binary polynomial formulas for the numbers 2^{i-1}. To catch the necessary ideas we first give an unsuccessful approach: It is $2^{i-1} = \sum\limits_{j_1=0}^{1} \sum\limits_{j_2=0}^{1} \ldots \sum\limits_{j_{i-1}=0}^{1} 1$, but the length of this binary formula is i. Since parameter i in the above sum for representing f_n is running from 1 to $2^{p(n)}$ the corresponding formula becomes too long. Instead, consider the binary representation of $i =: (i_1, \ldots, i_{p(n)})$. We define a binary polynomial formula for $G_1(i_1, \ldots, i_{p(n)}) := 2^{i-1}$. Its main building block is a formula for the characteristic function

$$F_1(j_1,\ldots,j_{p(n)},i_1,\ldots,i_{p(n)}) = \begin{cases} 1 & \text{if } 0 \neq (j_1,\ldots,j_{p(n)}) < (i_1,\ldots,i_{p(n)}) \\ -1 & \text{if } 0 = i \\ 0 & \text{otherwise} \end{cases} \quad ,$$

where the ordering $<$ is to be understood as ordering of the integers represented in binary by the corresponding tuples. Once a binary polynomial formula for F_1 is available one for G_1 is obtained via

$$G_1(i_1,\ldots,i_{p(n)}) = \prod_{j_1=0}^{1} \cdots \prod_{j_{p(n)}=0}^{1} \left(F_1(j_1,\ldots,j_{p(n)},i_1,\ldots,i_{p(n)})+1\right);$$

this follows from the definition of F_1 since the above product contributes a factor 2 for each $0 \neq j < i$, a factor 0 if $i = 0$ and a factor 1 in the other cases.

Binary polynomial formulas for the cases $i = 0$ and $j = 0$ are easily obtained. The order relation $(j_1,\ldots,j_{p(n)}) < (i_1,\ldots,i_{p(n)})$ can be expressed as

$$\begin{array}{l} j_{p(n)} < i_{p(n)} \;\vee\; \left\{j_{p(n)} = i_{p(n)} \wedge j_{p(n-1)} < i_{p(n-1)}\right\} \;\vee \ldots \\ \left\{j_{p(n)} = i_{p(n)} \wedge \ldots \wedge j_2 = i_2 \wedge j_1 < i_1\right\} \end{array} .$$

A binary polynomial formula for the characteristic function $y < z$ of comparing two single input bits is given by $z \cdot (z - y)$; and a formula for the above Boolean combination is obtained by combining two characteristic functions χ_1, χ_2 via $\chi_1 \cdot \chi_2$ for conjunctions and via $\chi_1 + \chi_2$ for disjunctions (note here that at most one clause becomes true). That way a binary polynomial formula representing G_1 is obtained. Its length clearly is polynomially bounded in n.

Step 2: Next, a binary polynomial formula for the function

$$G_\alpha(x_1,\ldots,x_{p(n)}) := x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_{p(n)}^{\alpha_{p(n)}} = x^\alpha$$

for given α is derived as follows. First, consider a single factor, for example $x_1^{\alpha_1}$, and let the binary representation of α_1 be $(\alpha_{11},\alpha_{12},\ldots,\alpha_{1p(n)})$. Now for $p(n)$ variables $t := (t_1,\ldots,t_{p(n)})$ consider the binary polynomial formula

$$\chi(\alpha_1 = 0) + (1 - \chi(\alpha_1 = 0)) \cdot \left(x_1 \cdot \prod_{t_1=0}^{1} \cdots \prod_{t_{p(n)}=0}^{1} \left(F_1(t_1,\ldots,t_{p(n)},\alpha_{11},\ldots,\alpha_{1p(n)}) \cdot (x_1 - 1) + 1\right) \right).$$

Here, $\chi(\alpha_1 = 0)$ denotes a binary polynomial formula for the characteristic function of the condition $\alpha_1 = 0$. A short moment of reflection now shows that for $\alpha_1 = 0$ the above formula results in $x_1^0 = 1$; if $\alpha_1 > 0$, then for each integer $0 < t < \alpha_1$ a factor x_1 is contributed whereas for $t = 0$ and $t \geq \alpha_1 > 0$ a factor 1 is obtained. Thus, the formula represents $x_1^{\alpha_1}$.

Since each monomial in f_n has $p(n)$ variables, the above construction can be repeated $p(n)$ many times to obtain

$$G_\alpha(x_1,\ldots,x_{p(n)}) = \prod_{j=1}^{p(n)} \Bigg[\chi(\alpha_j = 0) + (1 - \chi(\alpha_j = 0)) \cdot \left(x_j \prod_{t_1=0}^{1} \cdots \prod_{t_{p(n)}=0}^{1} \left(F_1(t_1,\ldots,t_{p(n)},\alpha_{j1},\ldots,\alpha_{jp(n)}) \cdot (x_j - 1) + 1\right) \right) \Bigg],$$

i.e., a binary polynomial formula for x^α of polynomial length. Note that in the above formula the first product results from applying a polynomial number of times construction step a),ii) of Definition 9, whereas the subsequent products result from step iii).

Step 3: The representation of the coefficients $a(n, \alpha, i)$ as binary polynomial formulas is based on PSPACE-completeness of the QBF problem, i.e., the question of deciding whether a quantified Boolean formula is true [19]. By assumption, computing $a(n, \alpha, i)$ can be done in PSPACE. Thus, for each n there exists a Boolean formula $\psi_n(\alpha, i) = \exists x_1 \forall x_2 \dots Q_m x_m \phi(n, \alpha, i)$ where the quantifiers range over $\{0, 1\}$, $Q_m \in \{\exists, \forall\}$, ϕ is quantifier free and $a(n, \alpha, i) = 1$ iff $\psi_n(\alpha, i)$ is true. Moreover, ψ_n can be computed uniformly in polynomial time in n. Next, arithmetize ψ_n in the folklore way (see, for example, [18]): first, compute in polynomial time a polynomial $q(x, \alpha, i), x = (x_1, \dots, x_m)$, that gives the truth value of the quantifier free formula $\phi(x, \alpha, i)$, then replace quantifiers of the form $\exists x_j q(..., x_j, ...)$ by $1 - \prod\limits_{x_j=0}^{1} (1 - q(..., x_j, ...))$ (this guarantees the result to stay in $\{0, 1\}$) and quantifiers of form $\forall x_j q(..., x_j, ...)$ by $\prod\limits_{x_j=0}^{1} q(..., x_j, ...)$. This gives uniformly a binary polynomial formula $G_2(n, \alpha, i)$ computing $a(n, \alpha, i)$.

Step 4: A binary formula for the sign $(-1)^{a(n,\alpha,0)}$ of a monomial x^α is given as $-2 \cdot G_2(n, \alpha, 0) + 1$.

Putting everything together, a binary polynomial formula representing $f_n(x_1, \dots, x_{p(n)})$ results from two further exponential sums, both expressed in our scheme via polynomially many applications of construction rule iii). Identifying as before $i = (i_1, \dots, i_{p(n)}), \alpha_j = (\alpha_{j1}, \dots, \alpha_{jp(n)})$ and $\alpha = (\alpha_1, \dots, \alpha_{p(n)})$ and recalling that $G_1(0) = 0$ this binary polynomial formula is

$$\sum_{j=1}^{p(n)} \sum_{\alpha_{j1}=0}^{1} \dots \sum_{\alpha_{jp(n)}=0}^{1} \left[(-2G_2(n, \alpha, 0) + 1) \cdot \left(\sum_{i_1=0}^{1} \dots \sum_{i_{p(n)}=0}^{1} G_1(i) \cdot G_2(n, \alpha, i) \right) \cdot G_\alpha(x_1, \dots, x_{p(n)}) \right] . \qquad \square$$

The theorem now can easily be applied to prove, maybe a bit surprisingly, that the classical technique by Shamir leads relatively far when designing interactive protocols also in the real number framework. More precisely, we have

▶ **Theorem 11.** *It holds* UniformVPSPACE$^0 \subseteq$ IP$_\mathbb{R}$ *in the following sense: Let* $\{f_n\}_n$ *be a family in* UniformVPSPACE^0 *such that* f_n *depends on* $u(n)$ *variables. Then there exists a real interactive protocol for the language* $\{(n, x, y) \in \mathbb{N} \times \mathbb{R}^{u(n)} \times \mathbb{R} \mid f_n(x) = y\}$.

Proof. The proof is an immediate application of Theorem 10 and the original proof of IP=PSPACE in [18]. Given an instance (n, x, y) the verifier first computes in polynomial time the binary polynomial formula obtained at the end of the proof of Theorem 10 representing $f_n(x)$. Note that it involves real numbers resulting from the input values x_j, has polynomial length and contains a polynomial number of operators of the form $\sum\limits_{t=0}^{1}$ and $\prod\limits_{t=0}^{1}$. This is the decisive observation; it implies that the technique used in Shamir's proof to verify interactively an equation $f_n(x) = y$ can be applied in our setting as well without major modifications: Once again, as briefly outlined in the introduction, the verification of $f_n(x) = y$ can be done by eliminating one after the other the leftmost of the operators. The fact that we deal with binary polynomial formulas of polynomial size guarantees that the univariate polynomials obtained with Shamir's construction have polynomially bounded degree. Therefore, the protocol runs in polynomial time.

□

4 Applications

In view of the difficulties described in the introduction when trying to design an interactive proof for problems in $\mathrm{PAR}_{\mathbb{R}}$ directly, an idea is to study oracle algorithms in the BSS model. More precisely, algorithms that are of interest use as information from an oracle different function evaluations. If f is a member of a family of functions such that for an argument x and a value y the equality $f(x) = y$ can be verified by an interactive protocol, then the outcome of a polynomial time BSS oracle computation having access to an oracle for values of f can be verified interactively as well; for each oracle query the verifier performs an interactive proof with the prover asking the latter to provide proofs of the correct oracle answers. Those are verified by the verifier. If it detects an error in any of the claimed oracle answers it rejects.

In order to obtain an interactive proof for interesting real complexity classes we can therefore consider such oracle computations. A typical classical example along this line is the computation of the permanent polynomial. In [13] an interactive protocol for verifying the value of a permanent of a 0-1-matrix was given (before Shamir's result was known). Together with Toda's theorem that the polynomial hierarchy PH is included in $P^{\#P}$ and the $\#P$-completeness of the permanent computation this implies the existence of an interactive protocol for all problems in the polynomial hierarchy. The protocol for the permanent, as for example described in [1], works as well for real matrices in the BSS model. This implies that real problems that can be decided by a polynomial time BSS algorithm having access to an oracle computing the permanent of real number matrices, i.e., all problems in class $\mathrm{P}_{\mathbb{R}}^{Perm}$, belong to $\mathrm{IP}_{\mathbb{R}}$. However, it is not known whether the permanent plays a similar role for real counting problems as it does in the Turing model. This is an active field of research. Basu and Zell [3] have given a real analogue of Toda's theorem. Instead of the permanent in this approach the computation of so-called Betti numbers of semi-algebraic sets plays a crucial role. The latter express certain topological properties of semi-algebraic sets. But they seem to be even more difficult to handle than permanent computations. And for the permanent itself a real variant of Toda's theorem is currently not known to hold.

In our context, Theorem 11 along the above lines has interesting consequences due to the strong relation the class $\mathrm{PAR}_{\mathbb{R}}$ has to $\mathrm{UniformVPSPACE}^0$. The main result of [12] is a transfer result which roughly states that if families in $\mathrm{UniformVPSPACE}^0$ can be evaluated efficiently, then there is a collapse of $\mathrm{PAR}_{\mathbb{R}}$ to $\mathrm{P}_{\mathbb{R}}$. On the way to prove this result the authors show a result most interesting for us; it witnesses the strength of oracle algorithms that query function evaluations of members of families in $\mathrm{UniformVPSPACE}^0$. We first state this result more precisely, starting with the following definition.

▶ **Definition 12** ([12]). A polynomial-time algorithm with $\mathrm{UniformVPSPACE}^0$-tests is a family $\{f_n(x_1, \ldots, x_{u(n)})\}_n \in \mathrm{UniformVPSPACE}^0$ together with a uniform family $\{C_n\}_n$ of constant-free algebraic circuits of polynomial size. The circuits in addition to their usual gates have special oracle gates of indegree $u(n)$. Those gates on input $x \in \mathbb{R}^{u(n)}$ output 1 if $f_n(x) \leq 0$ and 0 otherwise.

▶ **Theorem 13** ([12]). *For each $A \in \mathrm{PAR}_{\mathbb{R}}^0$ there is a polynomial-time algorithm with* $\mathrm{UniformVPSPACE}^0$*-tests deciding A.*

Given the remark following Lemma 8 the theorem holds analogously for all problems in $\mathrm{PAR}_{\mathbb{R}}$. Together with Theorem 11 we can now prove our main result.

▶ **Theorem 14.** $\mathrm{PAR}_{\mathbb{R}} \subsetneq \mathrm{IP}_{\mathbb{R}}$

Proof. Let $A \in \mathrm{PAR}_{\mathbb{R}}$. Theorem 13 and the subsequent remark imply that there exists a family $\{f_n\}_n \in \mathrm{UniformVPSPACE}^0$ such that membership in A can be decided by a polynomial time BSS algorithm that has access to an oracle answering questions of the form: is $f_n(x) \leq 0$ for certain arguments x computed during the algorithm. Now each time such an oracle question is posed the verifier asks the prover for a $y \leq 0$ (or $y > 0$, respectively, if the answer should be $f_n(x) > 0$). Then, it applies the algorithm behind the proof of Theorem 11 to verify the result and to continue with the correct oracle answer. If no error occurs the given input is accepted to belong to A, otherwise it is rejected. Given the arguments at the beginning of this section the statement follows. ◻

Applying the same line of arguments and picking up the above discussion it also follows that $\mathrm{P}_{\mathbb{R}}^{Res} \in \mathrm{IP}_{\mathbb{R}}$, where $Res = \{Res_n\}_n$ denotes the family of resultant polynomials of $n+1$ homogeneous polynomials in $n+1$ variables. This follows from [12] because there it is shown that $\mathrm{Res} \in \mathrm{UniformVPSPACE}^0$.

Problem 1. How large is the class $\mathrm{P}_{\mathbb{R}}^{\mathrm{UniformVPSPACE}^0}$?

In this paper we have derived a first significant lower bound for the class $\mathrm{IP}_{\mathbb{R}}$. Summarizing the results already mentioned the current picture is $\mathrm{PAR}_{\mathbb{R}} \subsetneq \mathrm{IP}_{\mathbb{R}} \subseteq \mathrm{MA}\exists\mathbb{R} \subseteq \mathrm{PAT}_{\mathbb{R}}$. There are some further immediate questions resulting from our lower bound. Given Shamir's characterization of classical IP it follows that IP is closed under complementation. However, without Shamir's result there seems no obvious way to prove this. Thus, in the real number setting we currently do not know whether the analogue statement holds.

Problem 2. Is it true that $\mathrm{IP}_{\mathbb{R}} = \text{co-}\mathrm{IP}_{\mathbb{R}}$?

Of course, we are still missing a characterization of $\mathrm{IP}_{\mathbb{R}}$. The work in [8] gives rise to conjecture $\mathrm{MA}\exists\mathbb{R} \subsetneq \mathrm{PAT}_{\mathbb{R}}$ which would imply that $\mathrm{IP}_{\mathbb{R}}$ is neither characterized by $\mathrm{PAR}_{\mathbb{R}}$ nor by $\mathrm{PAT}_{\mathbb{R}}$. Comparing our results with the different discrete characterizations of IP there seems to be only one more natural class left as a candidate, namely the class $\mathrm{PSPACE}_{\mathbb{R}}$ of problems being decidable by an algorithm using both exponential time and polynomially many registers. Note that requiring both conditions at the same time makes the coding argument from [15] not working. As mentioned above it is known that $\mathrm{PAR}_{\mathbb{R}} \subsetneq \mathrm{PSPACE}_{\mathbb{R}} \subseteq \mathrm{MA}\exists\mathbb{R} \subseteq \mathrm{PAT}_{\mathbb{R}}$. For establishing $\mathrm{PSPACE}_{\mathbb{R}}$ as a lower bound for $\mathrm{IP}_{\mathbb{R}}$ using the above techniques we should first get a similar result to Theorem 11 for a class like $\mathrm{UniformVPSPACE}^0$ such that using this class in oracle computations will cover $\mathrm{PSPACE}_{\mathbb{R}}$. We do not not know whether $\mathrm{UniformVPSPACE}^0$ itself or another similar class satisfies this. The upper bound $\mathrm{MA}\exists\mathbb{R}$ should then also be replaced by $\mathrm{PSPACE}_{\mathbb{R}}$.

Problem 3. What is the relation between $\mathrm{PSPACE}_{\mathbb{R}}$ and $\mathrm{IP}_{\mathbb{R}}$?

References

1 S. Arora, B. Barak: Computational Complexity: A Modern Approach. Cambridge University Press, 2009.

2 M. Baartse, K. Meer: Some results on interactive proofs for real computations. Extended abstract in: Proc. 11th conference Computability in Europe CiE 2015, Bucharest, A. Beckmann, V. Mitrana, M. Soskova (eds.), Springer LNCS 9136, 107–116, 2015.

3 S. Basu, T. Zell: Polynomial hierarchy, Betti numbers, and a real analogue of Toda's theorem. Foundations of Computational Mathematics, 10(4), 429–454, 2010.

4 L. Blum, F. Cucker, M. Shub, S. Smale: Complexity and Real Computation. Springer, 1998.

5 L. Blum, M. Shub, S. Smale: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. Amer. Math. Soc., vol. 21, 1–46, 1989.

6 O. Chapuis, P. Koiran: Saturation and Stability in the Theory of Computation over the Reals. Ann. Pure Appl. Logic 99 (1-3), 1–49, 1999.

7 F. Cucker: On the complexity of quantifier elimination: The structural approach. The Computer Journal, vol. 36 No. 5, 400–408, 1993.

8 F. Cucker, I. Briquel: A note on parallel and alternating time. Journal of Complexity, vol. 23, 594–602, 2007.

9 S. Goldwasser: Interactive Proof Systems. In: Computational Complexity Theory, J. Hartmanis (ed.), Proc. of Symposia in Applied Mathematics, Vol. 38, 108–128, 1989.

10 S. Goldwasser, M. Sipser: Private coins versus public coins in interactive proof systems. In Proc. of the 18th Symposium on Theory of Computing STOC, 59–68, 1986.

11 S. Ivanov, M. de Rougemont: Interactive Protocols on the reals. Computational Complexity 8, 330–345, 1999.

12 P. Koiran, S. Perifel: VPSPACE and a transfer theorem over the reals. Computational Complexity 18 (4), 551–575, 2009.

13 C. Lund, L. Fortnow, H. Karloff, N. Nisan: Algebraic methods for interactive proof systems. Journal of the ACM 39 (4), 859–868, 1992.

14 G. Malod: Succinct Algebraic Branching Programs Characterizing Non-Uniform Complexity Classes. Extended abstract in: Proc. 18th International Symposium on Fundamentals of Computation Theory FCT 2011, Oslo, Lecture Notes in Computer Science 6914, 205–216, 2011.

15 C. Michaux: Une remarque à propos des machines sur $\mathbb{R}$ introduites par Blum, Shub et Smale. C.R. Acad. Sci. Paris, t. 309, série I, 435–437, 1989.

16 B. Poizat: Â la recherche de la définition de la complexité d'espace pour le calcul des polynômes à la manière de Valiant. Journal of Symbolic Logic, 73:4, 1179–1201, 2008.

17 J. Renegar: On the computational Complexity and Geometry of the first-order Theory of the Reals , I - III. Journal of Symbolic Computation, 13, 255–352, 1992.

18 A. Shamir: IP = PSPACE. Journal of the ACM, vol. 39(4), 869–877, 1992.

19 L.J. Stockmeyer, A.R. Meyer: Word problems requiring exponential time. In: Proceedings STOC, ACM, 1–9, 1973.

Synchronizing Data Words for Register Automata*

Parvaneh Babari[1], Karin Quaas[2], and Mahsa Shirmohammadi[3]

1 Universität Leipzig, Germany
2 Universität Leipzig, Germany
3 University of Oxford, United Kingdom

Abstract

Register automata (RAs) are finite automata extended with a finite set of registers to store and compare data. We study the concept of synchronizing data words in RAs: Does there exist a data word that sends all states of the RA to a single state?

For deterministic RAs with k registers (k-DRAs), we prove that inputting data words with $2k+1$ distinct data, from the infinite data domain, is sufficient to synchronize. We show that the synchronizing problem for DRAs is in general PSPACE-complete, and is NLOGSPACE-complete for 1-DRAs. For nondeterministic RAs (NRAs), we show that Ackermann(n) distinct data (where n is the size of RA) might be necessary to synchronize. The synchronizing problem for NRAs is in general undecidable, however, we establish Ackermann-completeness of the problem for 1-NRAs. Our most substantial achievement is proving NEXPTIME-completeness of the length-bounded synchronizing problem in NRAs (length encoded in binary). A variant of this last construction allows to prove that the bounded universality problem in NRAs is co-NEXPTIME-complete.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Data Words, Register Automata, Synchronizing Problem, Ackermann-completeness, Bounded Universality, Regular-like expressions with squaring

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.15

1 Introduction

Synchronizing words for finite automata have been studied since the 70's, see [8, 26, 32, 24]; such a word w drives the automaton from an unknown or unobservable state to a specific state q_w that only depends on w. The famous Černý conjecture on synchronizing words is a long-standing open problem in automata theory. The conjecture claims that the length of a shortest synchronizing data word for a deterministic finite automaton (DFA) with n states is at most $(n-1)^2$. There exists a family of DFAs, where the length of the shortest synchronizing word is exactly $(n-1)^2$, which attains the exact claimed bound in the conjecture. Despite all received attention in last decades, this conjecture has not been proved or disproved.

Synchronizing words have applications in planning, control of discrete event systems, biocomputing, and robotics [3, 32, 16]. Over the past few years, this classical notion has sparked renewed interest thanks to its generalization to games on transition systems [22, 29, 21], and to infinite-state systems [15, 10], which are more relevant for modelling complex systems such as distributed data networks or real-time embedded systems. These studies have inspired an elegant extension of temporal logics to capture synchronizing properties [9]; the proposed logic is more expressive than classical computation tree logic.

* This work was partially supported by Deutsche Forschungsgemeinschaft (DFG), GRK 1763 (QuantLA) and project QU 316/1-2.

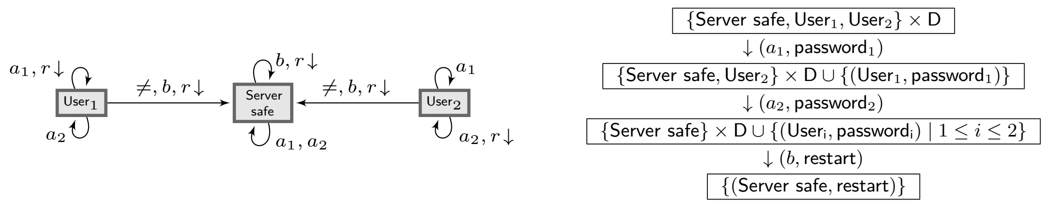

$a_1, r\downarrow$
User$_1$
a_2
$\neq, b, r\downarrow$
$b, r\downarrow$
Server safe
a_1, a_2
$\neq, b, r\downarrow$
a_1
User$_2$
$a_2, r\downarrow$
{Server safe, User$_1$, User$_2$} $\times$ D
$\downarrow (a_1$, password$_1)$
{Server safe, User$_2$} $\times$ D $\cup$ {(User$_1$, password$_1$)}
$\downarrow (a_2$, password$_2)$
{Server safe} $\times$ D $\cup$ {(User$_i$, password$_i$) $\mid 1 \leq i \leq 2$}
$\downarrow (b$, restart)
{(Server safe, restart)}

Figure 1 An RA $\mathcal{R}$ with single register r that models the interactive interfaces between a server and two users on the web. An *update*, denoted by $r\downarrow$, stores the input datum into r. Transitions labelled with $\neq$ are only taken if the datum of the current position of the input word and datum in register r are different. The data word $w = (a_1, \mathsf{password}_1)(a_2, \mathsf{password}_2)(b, \mathsf{restart})$ with the distinct datum restart is synchronizing; the set of successors after reading each input of w is shown on the right, where D is the infinite data domain. Observe that $\mathcal{R}$ is synchronized in (Server safe, restart).

In this paper, we are interested in *synchronizing data words* for *register automata. Data words* are sequences of pairs where the first element is taken from a finite alphabet and the second element is taken from *an infinite data domain* such as natural numbers or ASCII strings. In recent years, this structure has become an active subject of research thanks to applications in querying and reasoning about data models with complex structural properties, in XML, and lately also in graph databases [17, 2, 1, 5]. For reasoning about data words, various formalisms have been considered, ranging over first-order logic for data words [4, 6], extensions of linear temporal logic [23, 13, 12, 14], data automata [4, 7], register automata [20, 27, 25, 12] and extensions thereof, e.g. [31, 18, 11].

Register automata (RAs) are a natural generalization of finite automata over data words, and are equipped with a finite set of registers. When processing a data word, the automaton may store the data value of the current position in one or more registers. It may also test the data value of the current position for equality with the values stored in the registers, where the result of this test determines how the RA evolves. This allows for handling parameters like user names, passwords, identifiers of connections, sessions, etc., in a fashion similar to, and more expressive than, the class of data-independent systems. RAs come in different variants, e.g., one-way vs. two-way, deterministic vs. non-deterministic, alternating vs. non-alternating. For alternating RAs, classical decision problems like non-emptiness, universality and language inclusion are undecidable. We focus on the class of one-way RAs without alternation: They have a decidable non-emptiness problem [20], and the subclass of nondeterministic RAs with a single register has a decidable non-universality problem [12].

Semantically, an RA defines an infinite-state system, due to the unbounded domain for data stored in registers. Synchronizing words were introduced for infinite-state systems with infinite branching in [15, 29]; in particular, the notion of synchronizing words is motivated and studied for weighted automata and timed automata. In some infinite-state settings such as nested word automata (or equivalently visibly pushdown automata), finding the right definition of synchronizing words is however more challenging [10]. We define the synchronizing problem for RAs along the suggested framework in [15, 29]: Given an RA $\mathcal{R}$, does there exist a data word w that brings each of the infinitely many states of $\mathcal{R}$ to some specific state (depending only on w)? Such a data word is called a *synchronizing data word.*

Figure 1 depicts a web interface modelled by an RA $\mathcal{R}$ with register r. The RA models communications between a server and two users over an interactive interface. The server execute commands a_1, a_2 or b, and users locally attach private information as data to the input. The register r in each user's interface can be used to store local information such as the password, which implies the server has only partial information about the current state of the users' interfaces. When the server detects that an attacker is eavesdropping on the communication, it guides the system to a *safe* state. The data word $w = (a_1, \mathsf{password}_1)(a_2, \mathsf{password}_2)(b, \mathsf{restart})$ with the distinct datum restart, is synchronizing for the RA. We display the successive states after reading each input of w in Figure 1. The computation starts in the infinite set of all states in which the server and users might be; registers may have stored any datum from the data domain D, ranging over *infinitely many possible data values* (e.g. ASCII strings or numbers). The input $(a_1, \mathsf{password}_1)$ updates r in interface of the user 1 which synchronizes the infinite set of states of that user in the state $(\mathsf{User}_1, \mathsf{password}_1)$. However, no update has taken place in interface of the user 2. In fact, the register of that interface may still store any datum from D; this changes after inputting $(a_2, \mathsf{password}_2)$. Using the last input $(b, \mathsf{restart})$, the server accomplishes synchronizing $\mathcal{R}$ into $(\mathsf{Server\ safe}, \mathsf{restart})$. Now, the users can renew their passwords to prevent the attacker from future eavesdropping.

Contribution. The problem of finding synchronizing data words for RAs imposes new challenges in the area of synchronization. It is natural to ask how many distinct data are necessary and sufficient to synchronize an RA, which we refer to by the notion of *data efficiency* of synchronizing data words. We show this data efficiency to be polynomial in the number of registers for deterministic RAs (DRAs), and $\mathsf{Ackermann}(n)$ for nondeterministic RAs (NRAs), where n is the number of states. Remarkably, data efficiency is tightly related to the complexity of deciding the existence of a synchronizing data word.

For DRAs, we prove that for all automata $\mathcal{R}$ with k registers, if $\mathcal{R}$ has a synchronizing data word, then it also has one with data efficiency *at most* $2k + 1$. We provide a family $(\mathcal{R}_k)_{k\in\mathbb{N}}$ with k registers, for which indeed a polynomial data efficiency (in the size of k) is necessary to synchronize. This bound is the base of an (N)PSPACE-algorithm for DRAs; we prove a matching PSPACE lower bound by ideas carried over from timed settings [15]. We argue that, the synchronizing problems in DRAs with a single register (1-DRAs) and DFAs are NLOGSPACE-interreducible, implying that the problem is NLOGSPACE-complete for 1-DRAs.

For NRAs, a reduction from the non-universality problem yields the undecidability of the synchronization problem. For single-register NRAs (1-NRAs), we prove Ackermann-completeness of the problem by a novel construction proving that the synchronizing problem and the non-universality problem in 1-NRAs are polynomial-time interreducible. We believe that this technique is useful in studying synchronization in all nondeterministic settings, requiring careful analysis of the size of the construction.

Our most substantial achievement is proving NEXPTIME-completeness of the *length-bounded synchronizing problem* in NRAs: Does there exist a synchronizing data word with at most a given length (encoded in binary)?

For the lower bound, we present a non-trivial reduction from the bounded non-universality problem for *regular-like expressions with squaring*, which is known to be NEXPTIME-complete [30]. The crucial ingredient in this reduction is a family of RAs implementing binary counters. A variant of our construction yields a proof for co-NEXPTIME-completeness of the *bounded universality problem* in NRAs; the bounded universality problem asks whether all data words with at most a given length (encoded in binary) are in the language of the automaton.

2 Preliminaries

Deterministic finite-state automata (DFAs) are tuples $\mathcal{A} = \langle Q, \Sigma, \Delta\rangle$ where Q is a finite set of states, Σ is a finite alphabet and the transition function $\Delta : Q \times \Sigma \to Q$ is totally defined. The function Δ extends to finite words in a natural way: $\Delta(q, wa) = \Delta(\Delta(q, w), a)$ for all words $w \in \Sigma^*$ and letters $a \in \Sigma$; and it extends to all sets S by $\Delta(S, w) = \bigcup_{q\in S} \Delta(q, w)$.

Data Words and Register automata. Given an infinite data domain D, *data words* are finite words over $\Sigma \times \mathsf{D}$. For a data word $w = (a_1, d_1)(a_2, d_2)\ldots(a_n, d_n)$, the length of w is $|w| = n$. We use $\mathsf{data}(w) = \{d_1, \ldots, d_n\} \subseteq \mathsf{D}$ to refer to the set of data values occurring in w, and we say that the *data efficiency of* w is $|\mathsf{data}(w)|$.

Let reg be a finite set of *register variables*. We define *register constraints* ϕ over reg by the grammar $\phi ::= \mathsf{true} \mid \ =r \mid \phi \wedge \phi \mid \neg\phi$, where $r \in \mathsf{reg}$. We simply use $\neq r$ for the inequality constraint $\neg(=r)$; we denote by $\Phi(\mathsf{reg})$ the set of all register constraints over reg. A *register valuation* is a mapping $\nu : \mathsf{reg} \to \mathsf{D}$ that assigns a data value to each register; by a slight abuse of notation, we sometimes consider $\nu = \begin{pmatrix}\nu(r_1)\\ \cdots \\ \nu(r_k)\end{pmatrix} \in \mathsf{D}^k$ where $\mathsf{reg} = \{r_1, \cdots, r_k\}$. The satisfaction relation of register constraints is defined on $\mathsf{D}^k \times \mathsf{D}$ as follows: (ν, d) satisfies the constraint $= r$ if $\nu(r) = d$; the other cases follow. For example, $(\begin{pmatrix}d_1\\ d_2\\ d_1\end{pmatrix}, d_2)$ satisfies $((= r_1) \wedge (= r_2)) \vee (\neq r_3))$ where $d_1 \neq d_2$. For the set $\mathsf{up} \subseteq \mathsf{reg}$, we define the *update* $\nu[\mathsf{up} := d]$ of valuation ν by $\nu[\mathsf{up} := d](r) = d$ if $r \in \mathsf{up}$, and $\nu[\mathsf{up} := d](r) = \nu(r)$ otherwise.

Register automata (RAs) over infinite data domains D are tuples $\mathcal{R} = \langle\mathcal{L}, \mathsf{reg}, \Sigma, T\rangle$ where $\mathcal{L}$ is a finite set of locations, reg is a finite set of registers, Σ is a finite alphabet and $T \subseteq \mathcal{L} \times \Sigma \times \Phi(\mathsf{reg}) \times 2^{\mathsf{reg}} \times \mathcal{L}$ is a transition relation. We use $\ell \xrightarrow{\phi\ a\ \mathsf{up}\downarrow} \ell'$ to show transitions $(\ell, a, \phi, \mathsf{up}, \ell') \in T$. We call $\xrightarrow{\phi\ a\ \mathsf{up}\downarrow}$ an a-transition and ϕ the *guard*. We may omit ϕ when $\phi = \mathsf{true}$, and omit up if $\mathsf{up} = \emptyset$. We write $r\downarrow$ when $\mathsf{up} = \{r\}$ is singleton.

The *states* of $\mathcal{R}$ are pairs $(\ell, \nu) \in \mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$ of locations ℓ and register valuations ν; since the data domains for registers are infinite, RAs are infinite-state transitions systems. We describe the behaviour of $\mathcal{R}$ as follows: Given that $\mathcal{R}$ is in state $q = (\ell, \nu)$, on inputting the letter a and datum d, an a-transition $\ell \xrightarrow{\phi\ a\ \mathsf{up}\downarrow} \ell'$ may be fired if (ν, d) satisfies the constraint ϕ; then $\mathcal{R}$ starts in *successor* state $q' = (\ell', \nu')$ where $\nu' = \nu[\mathsf{up} := d]$ is the update on registers. By $\mathsf{post}(q, (a, d))$, we denote all successor states q' of q, on inputting letter a and datum d. A run of $\mathcal{R}$ over the data word $w = (a_1, d_1)(a_2, d_2)\cdots(a_n, d_n)$ is a sequence of states $q_0q_1 \ldots q_n$ where $q_i \in \mathsf{post}(q_{i-1}, (a_i, d_i))$ for all $1 \leq i \leq n$.

We extend post to sets S of states by $\mathsf{post}(S, (a, d)) = \bigcup_{q\in S} \mathsf{post}(q, (a, d))$; and we extend post to words by $\mathsf{post}(S, w \cdot (a, d)) = \mathsf{post}(\mathsf{post}(S, w), (a, d))$ for all words $w \in (\Sigma \times \mathsf{D})^*$, letters $a \in \Sigma$ and datum $d \in \mathsf{D}$.

In the rest of paper, we consider *complete* RAs, meaning that for all states $q \in \mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$ and all inputs $(a, d) \in \Sigma \times \mathsf{D}$, there is at least one successor: $|\mathsf{post}(q, (a, d))| \geq 1$. We also classify the RAs into *deterministic* (DRAs) and *nondeterministic* (NRAs), where an RA is deterministic if $|\mathsf{post}(q, (a, d))| \leq 1$ for all states q and all inputs (a, d).

Synchronizing words and synchronizing data words. The *synchronizing* words are a well-studied concept for DFAs; see [32]. Informally, a synchronizing word leads the automaton from every state to the same state: the word $w \in \Sigma^*$ is synchronizing for $\mathcal{A} = \langle Q, \Sigma, \Delta\rangle$ if there exists some state $\bar{q} \in Q$ such that $\Delta(Q, w) = \{\bar{q}\}$. The *synchronizing problem* in DFAs asks, given a DFA $\mathcal{A}$, whether there exists some synchronizing word for $\mathcal{A}$.

We introduce synchronizing data words for RAs: for an RA $\mathcal{R} = \langle \mathcal{L}, \mathsf{reg}, \Sigma, T \rangle$ over a data domain D, a data word $w \in (\Sigma \times \mathsf{D})^+$ is *synchronizing* if there exists some state $(\bar{\ell}, \bar{\nu})$ such that $\mathsf{post}(\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}, w) = \{(\bar{\ell}, \bar{\nu})\}$. The *synchronizing problem* asks, given an RA $\mathcal{R}$ over a data domain D, whether $\mathcal{R}$ has some synchronizing data word. The *bounded synchronizing problem* decides, given an RA $\mathcal{R}$ and $\mathsf{length} \in \mathbb{N}$ encoded in binary, whether $\mathcal{R}$ has such synchronizing data word w with $|w| \leq \mathsf{length}$.

3 Synchronizing data words for DRAs

In this section, we first show that the synchronizing problems in 1-DRAs and DFAs are NLOGSPACE-interreducible, implying that the problem is NLOGSPACE-complete for 1-DRAs. Next, we prove that the problem for k-DRAs, in general, can be decided in PSPACE; a reduction similar to the timed settings, as in [15], provides the matching lower bound. To obtain the complexity upper bounds, we prove that inputting words with data efficiency $2|\mathsf{reg}| + 1$ is sufficient to synchronize a DRA.

The concept of synchronization requires that all runs of RAs, whatever the initial state (initial location and register valuations), end in the same state $(\ell_{\mathsf{synch}}, \nu_{\mathsf{synch}})$ that only depends on the data word w_{synch}: $\mathsf{post}(\mathcal{L} \times \mathsf{D}, w_{\mathsf{synch}}) = \{(\ell_{\mathsf{synch}}, \nu_{\mathsf{synch}})\}$. While processing a synchronizing data word, the infinite set of states in RAs must necessarily shrink to a finite set of states. The RA $\mathcal{R}$ with 3 registers depicted in Figure 2 illustrates this phenomenon. Considering the set $\{x_1, x_2, x_3\} \subseteq \mathsf{D}$ of distinct data values; starting from states in $\{\mathsf{init}\} \times \mathsf{D}^3$, the infinite set of runs of $\mathcal{R}$ over the data word $(a, x_1)(a, x_2)(a, x_3)$ is merged into the finite set $\{(\ell_3, \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}), (\ell_3', \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix})\}$. We use this observation to provide a linear bound on the sufficient number of required distinct data while synchronizing RAs.

In Lemma 1, we prove that data words over only $|\mathsf{reg}|$ distinct data values are sufficient to shrink states of RAs to a finite set. We establish this result based on the following two key facts: (1) to shrink the set $\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$, one can find a word w_ℓ that brings the RA from $\{\ell\} \times \mathsf{D}^{|\mathsf{reg}|}$ to some finite set for every $\ell \in \mathcal{L}$. Thanks to being deterministic, appending some prefix or suffix to w_ℓ would achieve the same objective; so the successor set of $\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}$ and $(w_\ell)_{\ell \in \mathcal{L}}$ is a finite set. Moreover (2), when processing a synchronizing data word w_{synch} from a state (ℓ, ν) with $\nu(r) \notin \mathsf{data}(w_{\mathsf{synch}})$ for some $r \in \mathsf{reg}$, the register r must be updated. Observe that such updates must happen at inequality-guarded transitions, which themselves must be accessible by inequality-guarded transitions (possibly with no update).

For the RA $\mathcal{R}$ in Figure 2, assume that $d_1, d_2 \notin \mathsf{data}(w_{\mathsf{synch}})$. The two runs of $\mathcal{R}$ starting from $(\mathsf{init}, \begin{pmatrix} d_1 \\ d_1 \\ d_1 \end{pmatrix})$ and $(\mathsf{init}, \begin{pmatrix} d_2 \\ d_2 \\ d_2 \end{pmatrix})$ first take the transition $\mathsf{init} \xrightarrow{\neq r_1 \ a \ r_1\downarrow} \ell_1'$ updating register r_1. Next, the two runs must take $\ell_1' \xrightarrow{\mathsf{else} \ a \ r_2\downarrow} \ell_2'$ to update r_2 and $\ell_2' \xrightarrow{\mathsf{else} \ a \ r_3\downarrow} \ell_3'$ to update r_3; otherwise these two runs would never synchronize in a single state.

▶ **Lemma 1.** *For all DRAs for which there exist synchronizing data words, there exists some data word w with data efficiency $|\mathsf{reg}|$ such that* $\mathsf{post}(\mathcal{L} \times \mathsf{D}^{|\mathsf{reg}|}, w) \subseteq \mathcal{L} \times (\mathsf{data}(w))^{|\mathsf{reg}|}$.

After reading some word that shrinks the infinite set of states in RAs to a finite set S, one can apply the *pairwise synchronization* technique to synchronize states in S. This technique is the core to decide the synchronizing problem in DFAs in NLOGSPACE: Given a DFA $\mathcal{A} = \langle Q, \Sigma, \Delta \rangle$, it is known that it has a synchronizing word if and only if for all pairs of states $q, q' \in Q$, there exists a word v such that $\Delta(q, v) = \Delta(q', v)$ (see [32] for more details). The pairwise synchronization sets $S_{|Q|} = Q$, and for all $i = |Q| - 1, \cdots, 1$ repeats the following: find a word v_i such that $\Delta(q, v_i) = \Delta(q', v_i)$ for some pair $q, q' \in S_{i+1}$ and let

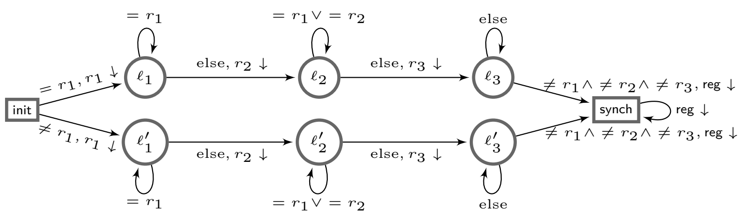


Figure 2 A DRA with three registers r_1, r_2, r_3 and single letter a (omitted from transitions) that can be synchronized in the state (synch, x_4) by the data word $w_{\mathsf{synch}} = (a, x_1)(a, x_2)(a, x_3)(a, x_4)$ if $\{x_1, x_2, x_3, x_4\} \subseteq \mathsf{D}$ is a set of 4 distinct data.

$S_i = \Delta(S_{i+1}, v_i)$. The word $w = v_{n-1} \cdots v_2 \cdot v_1$ is synchronizing for the DFA. We generalize the pairwise synchronization technique for DRAs to establish the following Lemma.

▶ **Lemma 2.** *For all DRAs for which there exist synchronizing data words, there exists a synchronizing data word with data efficiency* $2|\mathsf{reg}| + 1$.

Given a 1-DRA $\mathcal{R}$, the synchronizing problem can be solved by (1) ensuring that from each location ℓ an update on the single register is achieved by going through inequality-guarded transitions, which can be done in NLOGSPACE. Lemma 1 suggests that feeding $\mathcal{R}$ consecutively with a single datum $x \in \mathsf{D}$ is sufficient for this phase and the set of successors of $\mathcal{L} \times D$ would be a subset of $\mathcal{L} \times \{x\}$. Next (2) picking an arbitrary set $\{x, y, z\}$ of data including x, by Lemma 2 and the pairwise synchronization technique, the problem reduces to the synchronizing problem in DFAs where data in registers and input data extend locations and the alphabet: $Q = \mathcal{L} \times \{x, y, z\}$ and $\Sigma \times \{x, y, z\}$. Since a 1-DRA, where all transitions update the register and are guarded with true, models a DFA, we obtain the following result.

▶ **Theorem 3.** *The synchronization problem for* 1*-DRAs is in* NLOGSPACE*-complete.*

We provide a family of DRAs, for which a linear bound on the data efficiency of synchronizing data words, depending on the number of registers, is necessary. This necessary and sufficient bound is crucial to establish membership of synchronizing DRAs in PSPACE.

▶ **Lemma 4.** *There is a family of single-letter DRAs* $(\mathcal{R}_n)_{n \in \mathbb{N}}$*, with* $n = |\mathsf{reg}|$ *registers and* $\mathcal{O}(n)$ *locations, such that all synchronizing data words have data efficiency* $\mathcal{O}(n)$.

The synchronization problem for k-DRA is in PSPACE using the following co-(N)PSPACE algorithm: (1) picking a set $X = \{x_1, x_2, \cdots, x_{2k+1}\}$ of distinct data values, and (2) guessing some location $\ell \in \mathcal{L}$ and checking if there is no word $w \in (\Sigma \times \{x_1, x_2, \cdots, x_k\})^*$ with length $|w| \leq 2^{k|\mathcal{L}||\Sigma|}$ such that along firing inequality-guarded (on all k registers) transitions, some registers are not updated. Next (3) guessing two states $q_1, q_2 \in \mathcal{L} \times X^k$ such that there is no word $w \in (\Sigma \times X)^*$ with length $|w| \leq 2^{(2k+1)|\mathcal{L}||\Sigma|}$ such that $|\mathsf{post}(\{q_1, q_2\}, w)| = 1$.

▶ **Theorem 5.** *The synchronizing problem for k-DRAs is* PSPACE*-complete.*

4 Synchronizing data words for NRAs

In this section, we study the synchronizing problems for NRAs. We slightly update a result in [15] to present a general reduction from the *non-universality* problem to the synchronizing problem in NRAs. This reduction proves the *undecidability* result for the synchronizing problem in k-NRAs, and Ackermann-hardness in 1-NRAs. We then prove that in 1-NRA, the

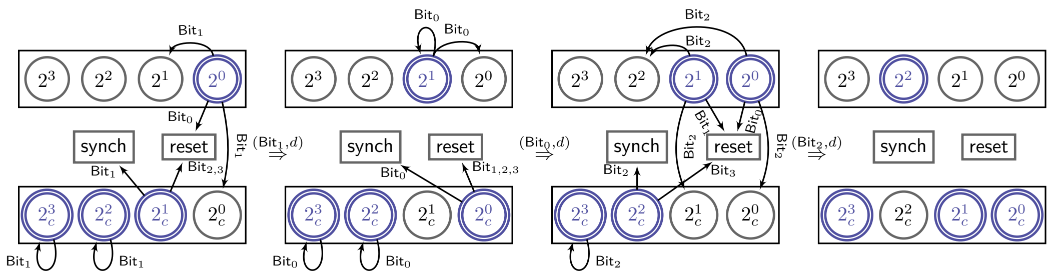


Figure 3 A partial illustration of the incrementing process of the 1-NRA $\mathcal{R}_{\mathsf{counter}}$ of Fig. 4. All Bit_i-transitions are equipped with equality guards. There is an x-token in all doubled transitions.

synchronizing and non-universality problems are indeed interreducible, which completes the picture by Ackermann-completeness of the synchronizing problem in 1-NRAs.

In nondeterministic settings, we present two kinds of counting features while synchronizing. A family of 1-NRAs (with $\mathcal{O}(n)$ locations) where $\mathsf{Ackermann}(n)$ distinct data must be read and another family where an input datum $x \in \mathsf{D}$ must be read 2^n times to achieve synchronization. The second family can be captured by NFAs if the shortest length to synchronize is of interest. To give the intuition behind the constructions, we say an x-token is in location ℓ of an RA after reading a data word v if $(\ell, x) \in \mathsf{post}(\mathcal{L} \times \mathsf{D}, v)$.

The 1-NRA $\mathcal{R}_{\mathsf{counter}}$ shown in Figure 4 encodes a binary counter: In every synchronizing data word w, some datum $x \in \mathsf{data}(w)$ must appear at least 2^3 times. The location synch has self-loops on all letters, thus, $\mathcal{R}_{\mathsf{counter}}$ is only synchronized in synch. Generally speaking, the counting involves *resetting* and *incrementing*. The resetting places an x-token in the location zero by an unavoidable $\star$-transition (tokens in reset can only move out by $\star$-transitions). The numbers $m \le 2^3$ are represented by placing x-tokens in the locations corresponding to binary representation of m. An x-token in location 2^i (and in 2^i_c) means that the i-th least significant bit in binary representation is set to $\mathsf{1}$ (to $\mathsf{0}$). First, by resetting, a Bit_0-transition places x-tokens in $\{2^3_c, 2^2_c, 2^1_c, 2^0\}$ to represent $\mathsf{0001}$. Next, an incrementing process can be set off by inputting the datum x via equality guards. In each increment step the tokens are replaced by firing specific Bit_i-transitions $(0 \le i \le 3)$, following the standard procedure of binary addition. Figure 3 shows the three increment steps. At the end, $\#$-transitions takes the token in 2^3 to location synch and finally synchronize $\mathcal{R}_{\mathsf{counter}}$.

▶ **Lemma 6.** *There is a family of 1-NRAs $(\mathcal{R}_{\mathsf{counter}(n)})_{n \in \mathbb{N}}$ with $\mathcal{O}(n)$ locations, such that for all synchronizing data words w, some datum $d \in \mathsf{data}(w)$ appears in w at least 2^n times.*

We next remark that the data efficiency while synchronizing 1-NRAs can be a function in the *fast growing hierarchy* [28]. Recall that $\mathsf{tower} : \mathbb{N} \to \mathbb{N}$ is defined inductively by $\mathsf{tower}(0) = 1$ and $\mathsf{tower}(n+1) = 2^{\mathsf{tower}(n)}$.

Figure 4 shows the 1-NRA $\mathcal{R}_{\mathsf{tower}}$ over the data domain $\mathbb{N}$. We indicate that $|\mathsf{data}(w)| \in \mathcal{O}(\mathsf{tower}(3))$ for all synchronizing data words w. As in $\mathcal{R}_{\mathsf{counter}}$, synch is the location where the RA must be synchronized in, and an initial *reset* is enforced to reach the location Data_1. The main issue is that while synchronizing $\mathcal{R}_{\mathsf{tower}}$, some inequality-guarded transitions are unavoidable, which are the ones that may *replicate* the tokens. For example, if one token in Data_1, firing two transitions $\mathsf{Data}_1 \xrightarrow{\neq r \ \ \mathsf{rep}} \mathsf{Data}_{1,2}$ and $\mathsf{Data}_1 \xrightarrow{\neq r \ \ \mathsf{rep} \ \ r\downarrow} \mathsf{Data}_{1,2}$ replicates it to two tokens in $\mathsf{Data}_{1,2}$.

Since the question is the required data efficiency of synchronizing words, we always start from datum 1 and feed $\mathcal{R}_{\mathsf{tower}}$ with the smallest number i which contributes to synchronization.

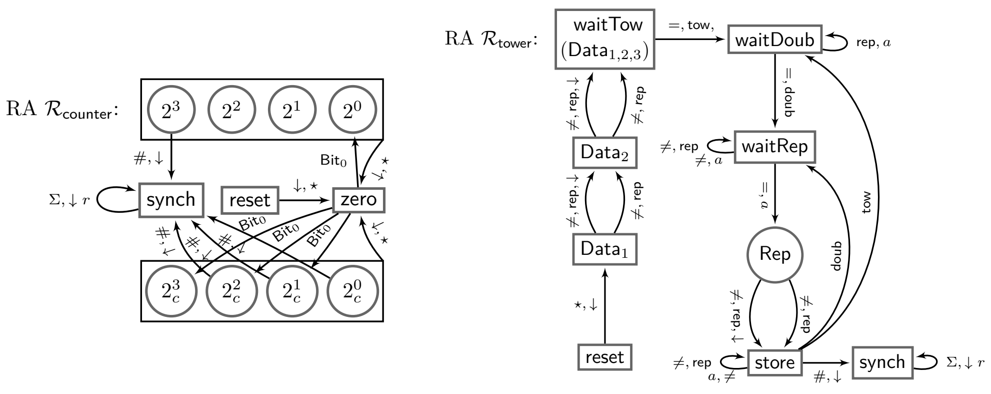


Figure 4 Bit$_i$-transitions in $\mathcal{R}_{\text{counter}}$ have equality guards. Most of the Bit$_i$-transitions are omitted; see Figure 3 for partial illustration of such transitions. Not-drawn $\star$-transitions activate a reset to zero in $\mathcal{R}_{\text{counter}}$, resp. to Data$_1$ in $\mathcal{R}_{\text{tower}}$. All inconsistent and inefficient transitions are omitted.

Moreover, when *resetting* we read datum 1. To synchronize $\mathcal{R}_{\text{tower}}$ with the least data efficiency, we go through the following steps:

▷ **resetting to** Data$_1$: the $\star$-transitions reset and place one token in Data$_1$ by $\ell \xrightarrow{\star\ r\downarrow}$ Data$_1$ for all $\ell \in \mathcal{L} \setminus \{$synch$\}$. Reading $\star$ is necessary for synchronizing since tokens in reset only move out by a $\star$-transition. Since another $\star$ eliminates all tokens and places one token in Data$_1$ again, resetting is inefficient; we call all transitions directed to reset *inefficient*.

▷ **replicating towering tokens**: after a reset with $(\star, 1)$ and having a 1-token in Data$_1$, the only efficient transitions are on $(\text{rep}, 2)(\text{rep}, 3)$, which results in replicating the 1-token in 3 tokens (shown as $\{1, 2, 3\}$-tokens) and placing them in waitTow.

▷ **towering the waiting i-token**: intuitively, the i-token in waitTow is waiting to trigger the tower(i)-process, right after the process of tower$(i-1)$ is accomplished. After the tower(i)-process, we see that $\{1, 2, \cdots, \text{tower}(i)\}$-tokens are in store. Next, if no more token is waiting in waitTow, the $\#$-transition synchronizes the RA into synch; otherwise, the inefficient $\#$-transition in waitTow resets. Below, we argue how, given a 3-token waiting in waitTow and $\{1, 2, \cdots, \text{tower}(2)\}$-tokens in store, the tower(3)-process proceeds. The first efficient transition is on $(\text{tow}, 3)$, which moves all those tokens to waitDoub. Recall that tower$(3) = 2^{\text{tower}(2)}$, simply doubling 1 for tower$(2) = 4$ times. Each i-token waiting in waitDoub (each in $\{1, 2, 3, 4\}$-tokens) is aimed to trigger a doubling,

▷ **1-token**: the only efficient transitions are on $(\text{doub}, 1)(a, 1)(\text{rep}, 2)$ which result in replicating $\{1, 2\}$-tokens in store.

▷ **2-token**: inputting $(\text{doub}, 2)$, which fires the only efficient transition, moves all the tokens obtained in the previous doubling process into waitRep. Then, both $\{1, 2\}$-tokens in waitRep will be replicated individually: note that while replicating, if a locally fresh datum from all data in waitRep, Rep and store is not read, an inefficient transition will be fired. After the second doubling by $(a, 1)(\text{rep}, 3)(a, 2)(\text{rep}, 4)$, the $\{1, 2, 3, 4\}$-tokens are produced in store.

▷ **3-token**: inputting $(\text{doub}, 3)$ moves $\{1, 2, 3, 4\}$-tokens into waitRep, which are indeed the tokens obtained in previous doubling process. For all $1 \leq i \leq 4$, the i-token is replicated into $\{i, 4+i\}$-tokens by $(a, i)(\text{rep}, 4+i)$. This results in storing $\{1, \cdots, 8\}$-tokens in store.

▷ **4-token**: it doubles the number of tokens in store for the 4-th time: $\{1, \cdots, 16\}$-tokens. So, tower$(3) = 2^{\text{tower}(2)} = 16$ distinct data are needed to synchronize $\mathcal{R}_{\text{tower}}$.

▶ **Lemma 7.** *There is a family of* 1*-NRAs* $(\mathcal{R}_{\mathsf{tower}(n)})_{n\in\mathbb{N}}$ *with* $O(n)$ *locations, such that* $|\mathsf{data}(w)| \in \mathcal{O}(\mathsf{tower}(n))$ *for all synchronizing data words* w*.*

We recall, from [28], that tower is at level 3 of the Ackermann-hierarchy. Using similar ideas as in Lemma 7, we can define a family of 1-NRAs $\mathcal{R}_n^m$ $(n, m \in \mathbb{N})$ such that all synchronizing data words have data efficiency at least $\mathsf{ack}_n(m)$, where ack_n is at level n of the Ackermann-hierarchy.

To define the language of a given RA $\mathcal{R}$, we equip it with an initial location ℓ_i and a set $\mathcal{L}_f$ of accepting locations, where, without loss of generality, we assume that all outgoing transitions from ℓ_i update all registers. The language $L(\mathcal{R})$ is the set of all data words $w \in (\Sigma \times \mathsf{D})^+$, for which there is a run from (ℓ_i, ν_i) to (ℓ_f, ν_f) such that $\ell_f \in \mathcal{L}_f$ and $\nu_i, \nu_f \in \mathsf{D}^{|\mathsf{reg}|}$. The universality problem asks, given an RA, whether $L(\mathcal{R}) = (\Sigma \times \mathsf{D})^+$. We adopt an established reduction in [15] to provide the following Lemma.

▶ **Lemma 8.** *The non-universality problem is reducible to the synchronizing problem for NRAs.*

As an immediate result of Lemma 8 and the undecidability of the non-universality problem for k-NRAs (Theorems 2.7 and 5.4 in [12]), we obtain the following theorem.

▶ **Theorem 9.** *The synchronizing problem for* k*-NRAs is undecidable.*

We present a reduction showing that, for 1-NRAs, the synchronizing problem is reducible to the non-universality problem, providing the tight complexity bounds for the synchronizing problem. We observe that Lemma 1 holds for 1-NRAs, meaning that for all 1-NRAs with some synchronizing data word, there exists some data word w with data efficiency 1 (for example, $\mathsf{data}(w) = \{x\}$) such that $\mathsf{post}(\mathcal{L} \times \mathsf{D}, w) \subseteq \mathcal{L} \times \mathsf{data}(w)$. Considering this fact as the skeleton, we define a language lang such that data words in this language are encodings of the synchronizing process. Let $\mathcal{L} = \{\ell_1, \ell_2, \cdots, \ell_n\}$ be the set of locations and x, y two distinct data. Each data word in lang, if there exists any, consists of

▷ **an initial block**: a delimiter $(\star, y)$ with distinct datum, the sequence $(\ell_1, x), (\ell_2, x)$, $\cdots, (\ell_n, x)$ and an input $(a, d) \in \Sigma \times \mathsf{D}$ as the first input of a synchronizing word. The initial block is followed by

▷ **a sequence of normal blocks**: the delimiter $(\star, y)$, successors reached from states and input in the previous block, and the next input of the synchronizing word. Finally, the data word ends with

▷ **a final block**: the delimiter $(\star, y)$, a single successor reached from states and input in the previous block and the delimiter $(\star, y)$.

We consider some further *membership conditions* for lang, which guarantee the correct semantics of the encoding of runs of $\mathcal{R}$. For instance, we impose the condition that for all (ℓ, d) and (a, d') with $d \neq d'$ in one block, if there exists a transition $\ell \xrightarrow{\neq r \;\; a \;\; r\downarrow} \ell'$, then (ℓ', d') must be in the next block.

We then construct a 1-NRA $\mathcal{R}_{\mathsf{comp}}$ that accepts the complement of lang; thus $\mathcal{R}$ has some synchronizing data word if, and only if, the language of $\mathcal{R}_{\mathsf{comp}}$ is not universal. The 1-NRA $\mathcal{R}_{\mathsf{comp}}$ is a finite union of smaller 1-NRAs, each of them violating one of the membership conditions for lang. For instance, the membership condition stated above is violated by the following 1-NRA.

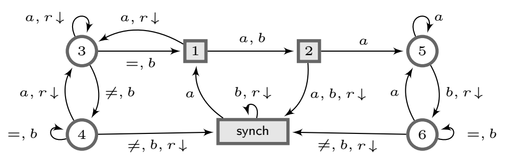
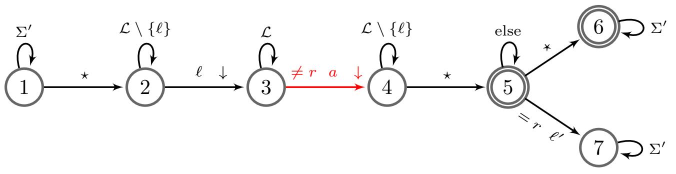


Figure 5 An RA where all synchronizing data words with length at most 3 require data efficiency 3 to shrink the infinite set of states to a finite subset.



▶ **Lemma 10.** *The synchronizing problem is reducible to the non-universality problem for* 1*-NRAs.*

By Lemmas 8 and 10 and Ackermann-completeness of the non-universality problem for 1-NRA, which follows from Theorem 2.7 and the proof of Theorem 5.2 in [12], and the result for counter automata with incrementing errors in [19], we obtain the following theorem.

▶ **Theorem 11.** *The synchronizing problem for* 1*-NRAs is* Ackermann*-complete.*

5 Bounded synchronizing data words for NRAs

The synchronizing problem for NRAs is undecidable in general, due to the unbounded length of synchronizing data words; In the following, we study, for NRAs, the bounded synchronizing problem, that requires the synchronizing data words to have at most a given length.

To decide the synchronizing problem in 1-RAs, in both deterministic and nondeterministic settings, we hugely rely on Lemma 1. We thus assume that the RA inputs the same datum x (chosen arbitrary) as many times as necessary to have the successor set included in $\mathcal{L} \times \{x\}$; next, we synchronize this successor set in a singleton. The RA $\mathcal{R}$ shown in Figure 5 shows that this approach is not useful when the length of synchronizing words are asked to not exceed a given bound. Observe that the data word $(a, x)(b, y)(b, z)$ is synchronizing with length 3 (not exceeding the bound 3). All synchronizing data words that repeat a datum such as x, to first bring the RA to a finite set, have length at least 5.

We first present a NEXPTIME-hardness result based on the binary counting feature in NRAs. The proof is by a reduction from the bounded non-universality problem for *regular-like expressions*. A regular-like expression over an alphabet Σ is a well-parenthesized expression built by constants $a \in \Sigma$, two binary operations $\cdot$ (concatenation) and $+$ (union), and a unary operation ^{2} (squaring). The language $L(\mathsf{expr})$ of such expressions expr is defined inductively as in regular expressions, where $L(\mathsf{expr}^2) = L(\mathsf{expr}) \cdot L(\mathsf{expr})$. The *bounded universality problem* asks, given a regular-like expression expr and length $N \in \mathbb{N}$ written in binary, whether $L(\mathsf{expr})$ includes all strings with length at most N; in other words, if $\Sigma^{\leq N} \subseteq L(\mathsf{expr})$.

▶ Remark. The bounded universality problem of regular-like expressions is co-NEXPTIME-complete, where the membership in co-NEXPTIME comes by guessing a witness string u with length at most N, and checking in EXPTIME that $u \notin L(\mathsf{expr})$. We observe that the reduction presented in [30], for the inequivalence between two regular-like expressions, establishes the co-NEXPTIME-hardness for the bounded universality problem, even if $|\Sigma| = 2$.

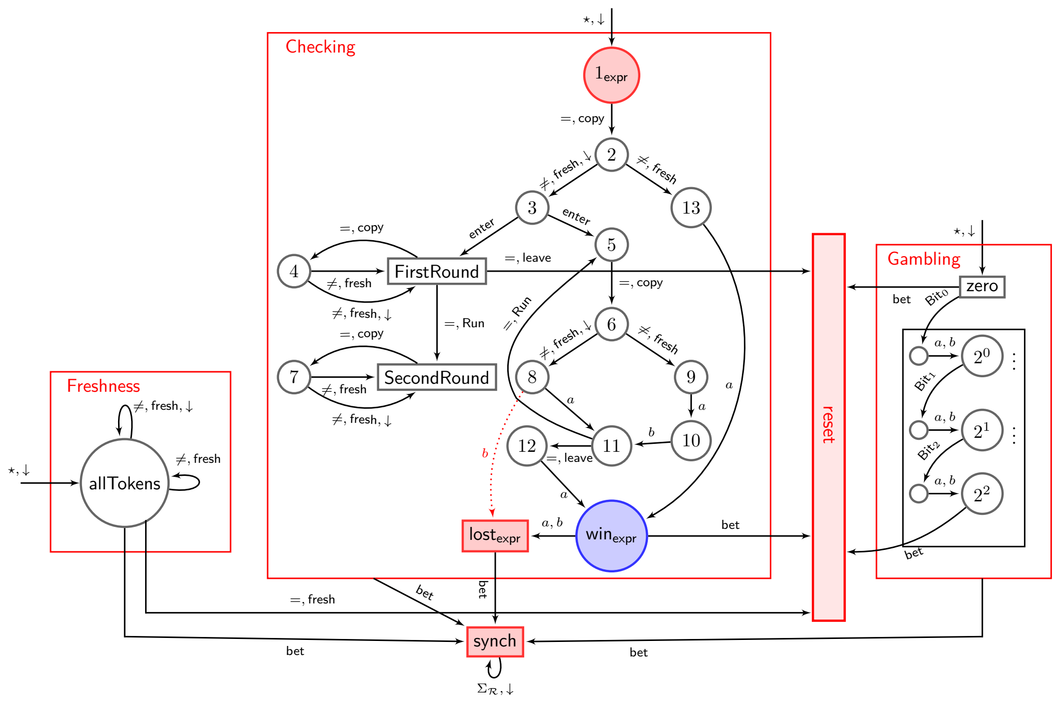


Figure 6 The $\star$-transitions reset $\mathcal{R}$, and all not-drawn a, b-transitions are inconsistent (except in allTokens). Other not-drawn transitions are self-loops.

Given a regular-like expression expr and length $N \in \mathbb{N}$, we construct a 1-NRA $\mathcal{R}$ and length $\in \mathbb{N}$, such that the language of expr is bounded universal if and only if $\mathcal{R}$ has no synchronizing data word with length at most length. The RA $\mathcal{R}$ consists of two distinguished locations reset, synch and three main gadgets: *Gambling*, *Freshness* and *Checking* gadget.

The RA $\mathcal{R}$ relies on the instincts of a *gambler* to synchronize. When feeding $\mathcal{R}$ with a data word w, we say that there is an x-token in location ℓ if $(\ell, x) \in \mathsf{post}(\mathcal{L} \times \mathsf{D}, w)$. Intuitively, whenever a token is in location reset, the gambler must restart; and $\mathcal{R}$ can only synchronize in synch. The reduction, roughly speaking, is such that the gambler guesses a string $u \in (a+b)^+$, letter-by-letter, and at some point places a *bet* that u is the witness for bounded non-universality. Gambling gadget discretely checks whether the bet makes sense: $|u| \leq N$. If *yes*, all tokens in Gambling gadget move to synch; otherwise, all tokens move to reset to give another chance to the gambler. On the other hand, meanwhile the gambler is hesitating to place the bet, Checking gadget tries to counter-attack the gambler by proving that expr generates u. To this aim, Checking gadget always follows all possible sub-expressions of expr which may produce u. This happens by replicating tokens and letting run computations for each sub-expression in parallel. As soon as one sub-expression fails in producing u, its token moves to lost$_{\mathsf{expr}}$ (of Checking gadget); and conversely, if a sub-expression definitely generates u, then its token moves to win$_{\mathsf{expr}}$ (of Checking gadget). The sub-expressions that have a string with prefix u keep their tokens in Checking gadget to follow the next computations (hoping that the gambler will not bet on u and continue guessing more letters). When a bet happens, all tokens in Checking gadget, except tokens in win$_{\mathsf{expr}}$, move to synch. In this way, $\mathcal{R}$ synchronizes in synch if $|u| \leq N$ and $u \notin L(\mathsf{expr})$.

Figure 6 depicts the constructed $\mathcal{R}$ for expr $= (a+ab)^2a + a$ and $N = 3$. Below, we give more intuitive explanations:

Gambler resets the guess: an initial *reset* is enforced while synchronizing since tokens in reset only move out by a $\star$-transition. When a reset happens, the gambler has the chance to change the guessed string u and to restart. Resetting eliminates all tokens in $\mathcal{R}$ and places tokens only to synch and the initial locations of all gadgets: zero, allTokens and 1_{expr}.

Gambler must only bet on $|u| \leq N$: after a reset, the sequence of read a, b is the guessed u by the gambler. Gambling gadget counts all a, b inputs to check whether $|u| \leq N$. This gadget is a chain of (modified) counting RAs $\mathcal{R}_{\mathsf{counter}(i)}$ described in Lemma 6, where $\mathcal{R}_{\mathsf{counter}(i)}$ counts until 2^i. We modify $\mathcal{R}_{\mathsf{counter}(i)}$ such that the *increment process*, triggered by Bit_i-transitions is executed after each occurrence of a or b. Gambling gadget in Figure 6 must count up to $N+1 = 2^2$ that is achieved by calling $\mathcal{R}_{\mathsf{counter}(2)}$.

Freshness gadget: after a reset, Checking gadget starts with a single token in 1_{expr}, say an x-token. This token moves along the gadget by reading u letter-by-letter and checking if the input prefix of u is in expr. For all unions, such as $a + ab$, the token replicates: x-token checks if a, and fresh y-token checks if ab contribute in generating u. Such tokens must move around individually, and thus must be distinctive. Freshness gadget guarantees the global freshness of such tokens: When replicating tokens by fresh-transitions, if the read datum is not fresh, the inconsistent transition $\mathsf{allTokens} \xrightarrow{=r\ \mathsf{fresh}} \mathsf{reset}$ happens.

Checking gadget: The checking is the gadget for expr that is built inductively from gadgets a, b, ab, $a+ab$, $(a+ab)^2$ and $(a+ab)^2a$. After a reset, it starts with a single token in 1_{expr}, if $u \in L(\mathsf{expr})$, then some token moves to $\mathsf{win}_{\mathsf{expr}}$ spoiling the gambler's plan in synchronizing. We explain the core of the sub-gadgets by following the scenario for $\mathcal{R}$ of $\mathsf{expr} = (a+ab)^2a+a$:

$\triangleright$ **When gambler bets on a wrong witness** $u \in L(\mathsf{expr})$, such as aaa. After a reset, assuming that an x-token is in 1_{expr}, it replicates by $(\mathsf{copy}, x)(\mathsf{fresh}, y)$ with $x \neq y$ to $\{x, y\}$-tokens. The x-token moves to 13 entering the a-gadget, and y-token to 3 entering the $(a+ab)^2a$-gadget. The only consistent transition is enter, the initial transition in the squaring. It makes a copy of the entering token in FirstRound to enforce the token to go through the gadget under squaring, two times. After (enter, y), there are y-tokens in FirstRound and in 5 as the initial location of the $(a+ab)$-gadget. For the union $a+ab$, inputting $(\mathsf{copy}, y)(\mathsf{fresh}, z)$ replicates the y-token in 5 to $\{y, z\}$-tokens where Freshness gadget guarantees that z is globally fresh. The z-token in 8 starts the a-gadget and y-token in 9 the ab-gadget. It is crucial that when union replicates tokens under squaring, their copy in FirstRound (and in SecondRound) must be replicated too: so $(\mathsf{copy}, y)(\mathsf{fresh}, z)$ replicates the y-token in FirstRound to $\{y, z\}$-tokens. Next a-transitions are consistent; observe that three tokens $\{x, y, z\}$ check if a is generated: as in $(a+ab)^2a + a$, the first produced a may be the result of three expressions: lonely a or a, ab under squaring.
The x-token from 13 moves to $\mathsf{win}_{\mathsf{expr}}$ meaning that $a \in L(\mathsf{expr})$; however, the gambler is betting on aaa, and the second a wastes this (fake) win by moving the token to $\mathsf{lost}_{\mathsf{expr}}$.
The y-token now must start the second round of squaring: inputting (run, y) brings back the y-token to 5, the initial of squaring, and also free the y-token in FirstRound to SecondRound (as a flag that y-token is ready to *leave* the squaring gadget). Due to the union again, the y-token, individually from z-token, must be replicated. By $(\mathsf{copy}, y)(\mathsf{fresh}, d)$ $(a, y)(\mathsf{leave}, y)(a, y)$, the y-token arrives in $\mathsf{win}_{\mathsf{expr}}$. The gambler places the bet with no more a, b, meaning that the y-token in $\mathsf{win}_{\mathsf{expr}}$ has no way to get synchronized, as it moves to reset by the bet-transition.
$\triangleright$ **When gambler bets on a right witness** $u \notin L(\mathsf{expr})$, such as bb. Observe that $(\star, x)(\mathsf{copy}, x)(\mathsf{fresh}, y)(\mathsf{enter}, x)(\mathsf{copy}, y)(\mathsf{fresh}, z)(b, x)(b, x)(\mathsf{bet}, x)$ synchronizes $\mathcal{R}$ into synch.

▷ **When gambler cheats** by betting on strings longer than N, such as *abbb*. The issue is when $abbb \notin L(\mathsf{expr})$, in these cases data words such as $(\star, x)(\mathsf{copy}, x)(\mathsf{fresh}, y)(\mathsf{enter}, x)$ $(\mathsf{copy}, y)(\mathsf{fresh}, z)(a, x)(\mathsf{run}, y)(\mathsf{copy}, y)(\mathsf{fresh}, d)(b, x)(\mathsf{run}, z)(\mathsf{copy}, z)(\mathsf{fresh}, m)(b, x)(b, x)$ would place all tokens of Checking gadget in $\mathsf{lost}_{\mathsf{expr}}$. Now, bet-transitions would move all tokens from Checking gadget to synch. However, Gambling gadget has counted 4, and thus location 2^2 has a token which goes to reset by placing the bet. This spoils synchronizing $\mathcal{R}$ when the gambler cheats by exceeding the bound $N = 3$. Note that tokens in zero, by bet-transitions, move to reset to forbid that the gambler cheats by the empty word too.

Note that $\mathsf{length} = 14$ of the synchronizing data word is computed inductively: here, $+1$ for resetting $\mathcal{R}$, $+2$ for the first union, $+(2 \cdot (2) + 3)$ for the squaring and union under it, $+1$ for the bet and $+N$ for Gambling gadget.

▶ **Lemma 12.** *The bounded synchronization problem for NRAs is* NEXPTIME*-hard.*

Guessing a data word w with $|(|w) \leq \mathsf{length}$ and checking in EXPTIME whether w is synchronizing yields NEXPTIME-membership. Altogether we obtain the following result:

▶ **Theorem 13.** *The bounded synchronization problem for NRAs is* NEXPTIME*-complete.*

The *bounded universality problem* asks, given an RA and $\mathsf{length} \in \mathbb{N}$ encoded in binary, whether all data words w with $|w| \leq \mathsf{length}$ are in the language of the automaton. We state that the bounded universality problem in NRAs is co-NEXPTIME-complete. The membership in co-NEXPTIME follows by guessing a witness w letter-by-letter; and checking if the successor states after reading w are all non-accepting. A variant of the presented reduction allows to prove that the bounded universality problem in NRAs is co-NEXPTIME-hard: equip $\mathcal{R}$ with the initial location reset and set $\mathcal{L}_f$ of accepting locations including all locations but synch.

▶ **Theorem 14.** *The bounded universality problem for NRAs is* co-NEXPTIME*-complete.*

Acknowledgements. We thank Sylvain Schmitz for helpful discussions on well-structured systems and nonelementary complexity classes.

References

1 Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008. doi:10.1145/1322432.1322433.

2 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4):31:1–31:46, December 2012. doi:10.1145/2389241.2389250.

3 Yaakov Benenson, Rivka Adar, Tamar Paz-Elizur, Zvi Livneh, and Ehud Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proc. National Acad. Sci. USA*, 100:2191–2196, 2003. doi:10.1073/pnas.0535624100.

4 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 7–16. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.51.

5 Mikolaj Bojanczyk and Pawel Parys. Xpath evaluation in linear time. *J. ACM*, 58(4):17:1–17:33, July 2011. doi:10.1145/1989727.1989731.

6 Ahmed Bouajjani, Peter Habermehl, Yan Jurski, and Mihaela Sighireanu. Rewriting systems with data. In Erzsébet Csuhaj-Varjú and Zoltán Ésik, editors, *Fundamentals of Computation Theory, 16th International Symposium, FCT 2007, Budapest, Hungary, August 27-30, 2007, Proceedings*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007. doi:10.1007/978-3-540-74240-1_1.

7 Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003. doi:10.1016/S0890-5401(03)00038-5.

8 Ján Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964.

9 Krishnendu Chatterjee and Laurent Doyen. Computation tree logic for synchronization properties. In *To be appear in 43rd Internation Colloquim on Automata, Languages, and programming, ICALP 2016*, 2016.

10 Dmitry Chistikov, Pavel Martyugin, and Mahsa Shirmohammadi. Synchronizing automata over nested words. In *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 252–268. Springer, 2016.

11 Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 738–749. IEEE, 2015. doi:10.1109/LICS.2015.73.

12 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009. doi:10.1145/1507244.1507246.

13 Stéphane Demri, Ranko Lazic, and David Nowak. On the freeze quantifier in constraint LTL: decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007. doi:10.1016/j.ic.2006.08.003.

14 Stéphane Demri, Ranko Lazic, and Arnaud Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010. doi:10.1016/j.tcs.2010.02.021.

15 Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing words for weighted and timed automata. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.121.

16 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Infinite synchronizing words for probabilistic automata. In *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 278–289. Springer, 2011.

17 Diego Figueira. Satisfiability of downward xpath with data equality tests. In *Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 197–206, New York, NY, USA, 2009. ACM. doi:10.1145/1559795.1559827.

18 Diego Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:22)2012.

19 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 269–278. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.39.

20 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

21 Jan Kretínský, Kim Guldstrand Larsen, Simon Laursen, and Jirí Srba. Polynomial time decidability of weighted synchronization under partial observability. In *26th International*

Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015, volume 42 of *LIPIcs*, pages 142–154. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

22 Kim Guldstrand Larsen, Simon Laursen, and Jirí Srba. Synchronizing strategies under partial observability. In *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2014.

23 Alexei Lisitsa and Igor Potapov. Temporal logic with predicate lambda-abstraction. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 147–155. IEEE Computer Society, 2005. `doi:10.1109/TIME.2005.34`.

24 Pavel V. Martyugin. Complexity of problems concerning carefully synchronizing words for PFA and directing words for NFA. In *Computer Science - Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010. Proceedings*, volume 6072 of *Lecture Notes in Computer Science*, pages 288–302. Springer, 2010.

25 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004. `doi:10.1145/1013560.1013562`.

26 Jean-Eric Pin. Sur les mots synthronisants dans un automate fini. *Elektronische Informationsverarbeitung und Kybernetik*, 14(6):297–303, 1978.

27 Hiroshi Sakamoto and Daisuke Ikeda. Intractability of decision problems for finite-memory automata. *Theor. Comput. Sci.*, 231(2):297–308, 2000. `doi:10.1016/S0304-3975(99)00105-X`.

28 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016.

29 Mahsa Shirmohammadi. Phd thesis: Qualitative analysis of probabilistic synchronizing systems. 2014.

30 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973. `doi:10.1145/800125.804029`.

31 Nikos Tzevelekos. Fresh-register automata. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 295–306. ACM, 2011. `doi:10.1145/1926385.1926420`.

32 Mikhail V. Volkov. Synchronizing automata and the cerny conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008. `doi:10.1007/978-3-540-88282-4_4`.

On the Sensitivity Conjecture for Read-k Formulas

Mitali Bafna*[1], Satyanarayana V. Lokam[2], Sébastien Tavenas[3], and Ameya Velingker*[4]

1 Indian Institute of Technology, Madras, Chennai, India
mitali.bafna@gmail.com
2 Microsoft Research, Bangalore, India
satya@microsoft.com
3 Microsoft Research, Bangalore, India
t-sebat@microsoft.com
4 Carnegie Mellon University, Pittsburgh, USA
avelingk@cs.cmu.edu

Abstract

Various combinatorial/algebraic parameters are used to quantify the complexity of a Boolean function. Among them, *sensitivity* is one of the simplest and *block sensitivity* is one of the most useful. Nisan (1989) and Nisan and Szegedy (1991) showed that block sensitivity and several other parameters, such as certificate complexity, decision tree depth, and degree over $\mathbb{R}$, are all polynomially related to one another. The sensitivity conjecture states that there is also a polynomial relationship between sensitivity and block sensitivity, thus supplying the "missing link".

Since its introduction in 1991, the sensitivity conjecture has remained a challenging open question in the study of Boolean functions. One natural approach is to prove it for special classes of functions. For instance, the conjecture is known to be true for monotone functions, symmetric functions, and functions describing graph properties.

In this paper, we consider the conjecture for Boolean functions computable by read-k formulas. A read-k formula is a tree in which each variable appears at most k times among the leaves and has Boolean gates at its internal nodes. We show that the sensitivity conjecture holds for read-once formulas with gates computing symmetric functions. We next consider *regular* formulas with OR and AND gates. A formula is regular if it is a leveled tree with all gates at a given level having the same fan-in and computing the same function. We prove the sensitivity conjecture for constant depth regular read-k formulas for constant k.

1998 ACM Subject Classification F.1.3 Relations among complexity measures

Keywords and phrases sensitivity conjecture, read-k formulas, analysis of Boolean functions

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.16

1 Introduction

Sensitivity and block sensitivity are two important complexity parameters of Boolean functions. The sensitivity conjecture states that these two parameters are polynomially related. A long-standing open question is to prove (or disprove) this conjecture. In this paper, we prove the conjecture for several subclasses of functions computable by read-k formulas.

The sensitivity $\mathrm{s}(f)$ of a Boolean function f is the maximum (over all inputs) number of coordinate dimensions along which the value of the function changes. This notion was

* This work was done while the author was an intern at Microsoft Research, Bangalore, India.

first introduced by Cook et al. [9] to prove lower bounds on the parallel complexity (in the CREW PRAM model) of Boolean functions. Nisan [16] introduced the more general definition of *block sensitivity.* The block sensitivity $\mathrm{bs}(f)$ of a Boolean function f is the maximum (again, over all inputs) number of disjoint subsets of coordinate dimensions such that flipping all values of a given input in any of these subsets results in flipping the value of the function. Nisan proved that block sensitivity asymptotically captures the CREW PRAM complexity of all Boolean functions. Remarkably, Nisan also showed that several other complexity parameters of Boolean functions such as certificate complexity, decision tree depth, and randomized decision tree depth are polynomially related to block sensitivity. Subsequently, Nisan and Szegedy [17] showed that block sensitivity and degree of polynomials (approximately) representing a Boolean function over $\mathbb{R}$ are polynomially related.

Hence, a number of combinatorial/algebraic parameters describing complexity of Boolean functions are all polynomially related to each other, but sensitivity has so far resisted such a polynomial equivalence with any of these other parameters. In fact, Nisan and Szegedy posed this as the *sensitivity vs. block sensitivity* question and since then, this question has come to be known as the "sensitivity conjecture". More than two decades later, proving (or disproving) this conjecture still remains a foundational challenge in the study of Boolean functions. In recent times, this quest has become even more intriguing as other complexity parameters such as quantum query complexity (both exact and two-sided error versions) have been shown to be polynomially related to block sensitivity [5, 7]. At the same time, the sensitivity conjecture has been shown to be related to a number of other conjectures and open questions in Boolean function complexity, as illustrated in the survey [13].

The best known universal (applicable to all functions) upper bound on block sensitivity remains exponential in sensitivity [19] (see [14], [3], [21] for more refined upper bounds). In the other direction, Rubinstein [18] gives an example function where the gap between sensitivity and block sensitivity is quadratic (see [4] and references therein for improvements in constants). Thus the challenge is to close this gap between quadratic and exponential relations between block sensitivity and sensitivity.

Several approaches have been proposed in the literature to attack the sensitivity conjecture. Gotsman and Linial [12] showed that the degree vs. sensitivity problem is equivalent to a combinatorial problem on the maximum degree of induced subgraphs of the Boolean cube. Aaronson [1] (see also [6]) stated a problem about certain two-colorings of the integer lattice whose solution would imply the sensitivity conjecture. Recently, Gilmer et al. [10] formulated an approach to the degree vs. sensitivity problem using lower bounds on a two-party communication game. Even more recently, Gopalan et al. [11] prove an ℓ_2-approximate version of the degree vs. sensitivity conjecture (the original one needs an ℓ_∞-approximation). They also formulate the notion of tree sensitivity and a robust analog of the degree vs. sensitivity conjecture.

To make progress on our understanding of this problem, researchers also studied the conjecture on special classes of Boolean functions. It is trivial to see that the conjecture holds for monotone functions and symmetric functions. A natural question, then, is if the sensitivity conjecture holds when the function is invariant under other groups of symmetries. Turán [22] proved that for Boolean functions that describe graph properties (edges are the Boolean variables) sensitivity is $\Omega(\sqrt{n})$ and hence the conjecture holds for graph properties. Chakraborty [8] studied minterm-transitive Boolean functions and showed that for such functions sensitivity is $\Omega(n^{1/3})$, thus showing the conjecture for this class of functions. Sun [20] studied block sensitivity for Boolean functions invariant under any transitive permutation group and showed that such functions must have block sensitivity $\Omega(n^{1/3})$.

Our Results: We prove the sensitivity conjecture for another restricted class of Boolean functions, namely certain functions computed by read-k formulas. A *read-k formula* is a tree whose internal nodes are Boolean gates, e.g., AND and OR, and leaves are literals of input variables with the restriction that each variable (as negated or non-negated literal) appears at most k times among the leaves. Such a formula computes a Boolean function in a natural way from the leaves to the root. A formula is called *regular* if all gates at a given depth are the same type and have the same fan-in.

In what follows, we will mainly focus on formulas composed of OR and AND gates. In particular we show that the sensitivity conjecture is true for read-$\log n$ regular formulas whose bottom fanins are sufficiently large.

▶ **Theorem 1. Regular read-$\log n$ with large bottom fanin.** *Let f be a Boolean function, dependent on n variables, computed by a regular read-$(\log n)$ formula with bottom fan-in at least $\log^2 n$. Then $\mathrm{s}(f) \geq \tilde{\Omega}\left(\mathrm{bs}(f)^{1/4}\right)$, where the $\tilde{\Omega}$ notation hides some logarithmic terms.*

We would like to remove the condition on the bottom fanin. We succeed in doing so when the read and depth of the formula are constants.

▶ **Theorem 2. Regular read-constant and constant depth.** *Let f be computed by a regular read-k formula of depth-d for constants k and d such that all internal gates compute non-constant AND-OR functions. Then $\mathrm{s}(f) = \Omega_{k,d}(\sqrt{\mathrm{bs}(f)})$, where the hidden constant is a (rapidly decreasing) function of k and d.*

We present our main results (Theorem 1 and 2) on regular read-k formulas with AND and OR gates in Section 4. A crucial ingredient of our proofs is an application of the *Lovász Local Lemma* (LLL) to show that some literals can be assumed to occur in their positive form in such a formula without increasing the function's sensitivity and ensuring that any satisfying assignment of such a formula must have a large Hamming weight. However, in order to apply LLL, we need the bottom fan-in of such formulas to be large enough. So, we first prove the conjecture for formulas with large bottom fan-in. We then remove the restriction on the bottom fan-in by switching AND's of OR's to OR's of AND's (or vice versa). The idea is that if the formula is sufficiently large and the depth small, there has to be a layer L with large fanin. Then, by switching, we expand the layers under L and put L close to the bottom.

When specialized to read-once formulas with symmetric gates or to read-k DNF's our lower bounds on regular read-k formulas yield better dependence on k.

▶ **Theorem 3. Read-once with symmetric gates.** *Let f be a Boolean function dependent on n variables and computed by a read-once formula with symmetric gates. Then, $\mathrm{s}(f) \geq \sqrt{n}$.*

We note that Hiroki Morizumi [15] proved a similar lower bound for read-once AND-OR formulas.

▶ **Theorem 4. Read-k DNF.** *Let f be a Boolean formula dependent on n variables and computed by a read-k DNF. Then $\mathrm{s}(f) \geq n^{1/3}/(k+2)$. In particular, if $k \leq n^{\frac{1}{3}-\varepsilon} - 2$, then $\mathrm{s}(f) \geq n^{\varepsilon} \geq \mathrm{bs}(f)^{\varepsilon}$.*

Our proof of the conjecture for read-once formulas with symmetric gates appears in Section 3. The results on DNF's appear in Section 5.

2 Notations and Preliminaries

- In this paper, log will always denote the logarithm to base two.
- We will always assume that f is a Boolean function on n variables and moreover that it depends on all its variables.

2.1 Measures on Boolean functions

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. For $x \in \{0,1\}^n$ and $S \subseteq [n]$, we denote by x^S the vector obtained by flipping all the coordinates on x in S. For $x \in \{0,1\}^n$ and $z \in \{0,1\}$, we denote by $|x|_z$ the number of coordinates of x with the value z.

▶ **Definition 5. Sensitivity:**

- The *sensitivity* of f at x is defined as the number of coordinates of x, which when flipped, will flip the value of f: $\mathrm{s}(f,x) := |\{i \in [n] : f(x) \neq f(x^i)\}|$.
- For $z \in \{0,1\}$, the *z-sensitivity* of f is defined as the maximum sensitivity of f at an input in $f^{-1}(z)$: $\mathrm{s}_z(f) := \max\{\mathrm{s}(f,x) : f(x) = z\}$.
- Finally, the *sensitivity of f* is the maximum sensitivity of f among all inputs: $\mathrm{s}(f) := \max\{\mathrm{s}(f,x) : x \in \{0,1\}^n\} = \max\{\mathrm{s}_0(f), \mathrm{s}_1(f)\}$.

▶ **Definition 6.** The *block sensitivity of f at x*, denoted $\mathrm{bs}(f,x)$ is the maximum number of *disjoint* subsets $S_1, \ldots, S_b$ of $[n]$ such that for every i, $f(x) \neq f(x^{S_i})$. The z-block sensitivity and block sensitivity of f are defined similar to the case of sensitivity. In particular, $\mathrm{bs}(f) := \max\{\mathrm{bs}(f,x) : x \in \{0,1\}^n\}$.

▶ **Definition 7.** A *certificate of f on x* is a subset $S \subseteq [n]$ such that $f(y) = f(x)$ whenever $y_i = x_i, \forall i \in S$. The size of the certificate S is $|S|$.

The *certificate complexity of f on x* denoted by $\mathrm{C}(f,x)$ is the size of a smallest certificate of f on x. The *certificate complexity of f* denoted by $\mathrm{C}(f)$ is $\max_x \mathrm{C}(f,x)$. For $z \in \{0,1\}$, the z-certificate complexity of f denoted by $\mathrm{C}_z(f)$, is $\max_{x \in f^{-1}(z)} \mathrm{C}(f,x)$.

We will use the following known results.

▶ **Lemma 8.** *For any Boolean function f and $z \in \{0,1\}$, $\mathrm{C}_z(f) \geq \mathrm{bs}_z(f) \geq \mathrm{s}_z(f)$.*

The first inequality above is from [16] and the second inequality is obvious from definitions.

▶ **Theorem 9** ([4]). *For any Boolean function f and $z \in \{0,1\}$, $\mathrm{C}_z(f) \geq \frac{3\mathrm{bs}_{1-z}(f)}{2\mathrm{s}_{1-z}(f)} - \frac{1}{2}$.*

2.2 Formulas

▶ **Definition 10. Regular Read-k Formulas:**

- A formula C is said to be *$(a_1, \ldots, a_d)$-regular* if it is a layered tree of depth d whose leaves are input variables or their negations and all internal nodes at a given layer i, $1 \leq i \leq d$, are gates of the same kind and the same fanin a_i. The layers are numbered 1 through $d+1$ from the root (output) to the leaves (inputs). We will often denote the gates at the layer d by *bottom gates*. In this paper, we only consider both formulas of alternating layers of AND and OR gates (we could start at the root with either gate and then alternate) and formulas with symmetric gates.
- A formula is *read-k* if each variable (either in its negated or non-negated form) appears at most k times among its leaves.
  One can argue that by replicating the arguments, we can always assume that the formula is in regular form. However, this idea does not work here because by doing this transformation, we would increase the read-multiplicity of the formula.

2.3 Lovász local lemma

We will make use of the Lovász Local Lemmas:

▶ **Lemma 11.** *[The Lovász Local Lemma: Symmetric Case] Let $A_1, \ldots, A_n$ be events in an arbitrary probability space. Suppose that each event A_i is mutually independent of a set of all the other events A_j but at most d and that $\Pr[A_i] \leq p$ for all $1 \leq i \leq n$.*

If $ep(d+1) < 1$, then $\Pr\left[\bigcap \overline{A_i}\right] > 0$.

We will also use the general version of this lemma. Both versions can be found, e.g., in [2].

3 Read-once formulas with symmetric gates

In this section, we prove the sensitivity conjecture for read-once formulas with *symmetric* gates. The read-multiplicity is more restrictive than the model we will consider later but the gates we allow are more powerful.

▶ **Definition 12.** Let g be a non-constant symmetric function on m inputs. We define $\tau(g)$ to be the minimal weight of an input $x \in \{0,1\}^m$ such that $g(x) \neq g(\vec{0})$:

$$\tau(g) := \min\left\{i \mid \ |x|_1 = i \implies g(x) \neq g(\vec{0})\right\}.$$

▶ **Theorem 13.** *Let f be a Boolean function computed by a read-once formula $\mathtt{C}$ with symmetric gates. Then, $\mathrm{s}_0(f)\mathrm{s}_1(f) \geq n$.*

Proof. We prove it by induction on the depth of the formula $\mathtt{C}$. If the depth of the formula is 1, then f is a symmetric function on n variables.

Let $z = f(\vec{0})$ and $t := \tau(f)$. By Definition 12, when $|x|_1 = t-1$, $f(x) = z$ and when $|y|_1 = t$, $f(y) = 1-z$. It follows immediately that $s_z(f,x) \geq n-t+1$ and $s_{1-z}(f,y) \geq t$. So $\mathrm{s}_0(f)\mathrm{s}_1(f) \geq t(n-t+1) \geq n$.

Now assume that the theorem is true for all depths $\leq d$. We prove it for depth $d+1$.

So $f = h(g_1, \ldots, g_m)$, where h is symmetric and each g_i is computed by a read-once formula with symmetric gates, of depth at most d. Let every g_i be a function on n_i variables with $a_i = \mathrm{s}_0(g_i)$ and $b_i = \mathrm{s}_1(g_i)$. By the inductive hypothesis, we know that $a_ib_i \geq n_i$. Since $n = \sum_{i=1}^m n_i$, we have that, $\sum_{i=1}^m a_ib_i \geq n$. Without loss of generality, we may assume that $a_1 \geq a_2 \geq \ldots \geq a_m$ and $b_{\pi(1)} \geq b_{\pi(2)} \geq \ldots \geq b_{\pi(m)}$ for a suitable permutation π of $[m]$. Let $A_j := \sum_{i=1}^j a_i$ and $B_j := \sum_{i=1}^j b_{\pi(i)}$. Let $t := \tau(h)$ so $h(x) = z$ for all x with $|x|_1 = t-1$ and $h(y) = 1-z$ for all y with $|y|_1 = t$.

Since the formula is read-once, the g_i depend on disjoint sets of variables, and so it is easy to see that for all S with $|S| = t-1$, we can find an assignment σ to all the variables of f such that (i) $g_i(\sigma_i) = 1$ for exactly those $i \in S$ and (ii) for $i \notin S$, $g_i(\sigma_i) = 0$ and g_i has $a_i = \mathrm{s}_0(g_i, \sigma_i)$ sensitive inputs.

It follows that $s_z(f) \geq \max_{\substack{S \subseteq [n] \\ |S| = m-t+1}} \{\sum_{i \in S} a_i\} = A_{m-t+1}$.

Similarly, $s_{1-z}(f) \geq \max_{\substack{S \subseteq [n] \\ |S| = t}} \{\sum_{i \in S} b_i\} = B_t$.

So, $\mathrm{s}_0(f)\mathrm{s}_1(f) \geq A_{m-t+1}B_t = (a_1 + \ldots + a_{m-t+1})(b_{\pi(1)} + \ldots + b_{\pi(t)})$.

Our proof is completed by the following claim whose proof is given in the full version.

▶ **Claim 14.** *For any t, $1 \leq t \leq m$, $A_{m-t+1}B_t \geq \sum_{i=1}^m a_ib_i$.*

We therefore conclude that $\mathrm{s}_0(f)\mathrm{s}_1(f) \geq A_{m-t+1}B_t \geq \sum_{i=1}^m a_ib_i \geq n$. ◀

▶ **Corollary 15.** *Let f be a Boolean function computed by a read once formula C with symmetric gates. Then, $s(f) \geq \sqrt{n} \geq \sqrt{bs(f)}$.*

Furthermore, this bound is tight whenever n is a perfect square. To see the tightness of the bound, consider an OR of fan-in $\sqrt{n}$ over $\sqrt{n}$ disjoint AND's on $\sqrt{n}$ variables each. It is easy to see that both 0-sensitivity and 1-sensitivity of this function are exactly $\sqrt{n}$.

4 Read-k formulas

In the following, we will only consider AND-OR formulas (with positive and negative literals). In this section, we prove the sensitivity conjecture for read-k formulas with certain restrictions.

▶ **Theorem 16.** *Let f be computed by a regular read-k formula of depth d with constants k and d such that any internal gate computes a non-constant function. Then, $\mathrm{s}(f) = \Omega_{k,d}(\sqrt{\mathrm{bs}(f)})$, where the hidden constant is a (rapidly decreasing) function of k and d.*

We prove this theorem in two stages:

- In Section 4.1, we first prove a lower bound for $\mathrm{s}(f)$ in terms of $\mathrm{bs}(f)$ when f is computed by a read-k regular formula with large bottom fanin.
- Then, in Section 4.2, we remove the condition on the bottom fanin by defining a normal form for formulas and then reducing a formula with small bottom fanin to one in the normal form where the previous step applies.

Notation: When $\mathtt{C}$ is an $(a_1, \ldots, a_d)$-regular formula with AND-OR gates we will use $A(\mathtt{C}, j)$ to denote the product,

$$A(\mathtt{C}, j) = \prod_{\substack{l \in [j] \\ l \text{ is a } \wedge\text{-gates level}}} a_l .$$

As most of the times, the function A will be used on the parameters $\mathtt{C}$ and $j = d - 2$, we will denote $A(\mathtt{C}, d-2)$ by A.

4.1 Large bottom fan-in

In this section, we give a lower bound for sensitivity in terms of block sensitivity for read-k regular formulas with large bottom fanin.

4.1.1 1-Sensitivity when bottom gates are AND gates

We will first prove a lower bound on the 1-sensitivity of such formulas. We will show that given a formula $\mathtt{C}$ it is possible to get an equivalent formula $\mathtt{C}'$ which has certain nice properties. Specifically, all inputs on which $\mathtt{C}'$ evaluates to 1 have large Hamming weight, which directly implies that the 1-sensitivity for this function is large.

▶ **Definition 17.** A parse tree P of a formula $\mathtt{C}$ computing f is a subcircuit which is recursively defined as follows:

- The output gate of $\mathtt{C}$ is in P.
- If an $\wedge$-gate belongs to P then all its children are also in P.
- If an $\vee$-gate belongs to P then exactly one of its children is in P.

It is easy to see that f evaluates to 1 on an input x if and only if $\mathtt{C}$ contains a parse tree all of whose gates evaluate to 1. A simple induction also shows that every parse tree of a regular formula has $A(\mathtt{C}, d-1)$ bottom gates.

▶ **Definition 18.** The parse-read of $\mathtt{C}$ is the maximum number of times any variable appears in any parse tree.

We will now consider two models. The first model is a (natural) restriction of our model of regular formulas: a variable can appear at most once under the same bottom gate. The second model is the general one without this restriction.

▶ **Lemma 19.** *Let $(a_1, \ldots, a_d) \in (\mathbb{N} \setminus \{0\})^d$ with $a_d \geq 2\log 4k$. Let f be a non-constant function computed by an $(a_1, \ldots, a_d)$-regular read-k and parse-read p formula such that the bottom gates are $\wedge$-gates and such that each variable appears at most once under any bottom gate. Then*

$$\mathrm{s}_1(f) \geq \left(\frac{a_d - 2\log 4k + 1}{2p\log 4k}\right) A.$$

Proof. By regularity, any bottom gate of $\mathtt{C}$ is the parent of a_d literals. Let us group these literals into groups of size α whose value will be chosen later. The last group will be of size a_d modulo α. So we get $\lfloor a_d/\alpha \rfloor$ groups of α literals under every bottom gate. We want to modify $\mathtt{C}$ to $\mathtt{C}'$ such that each group contains at least one positive literal.

Let us randomly negate each variable. Each variable is independently chosen as positive or negative with probability $\frac{1}{2}$. Let A_i be the event that the i^{th} group has no positive literals (where the i^{th} group is taken over all groups under all bottom gates). So $\Pr[A_i] = \frac{1}{2^\alpha}$. Every event A_i is dependent on at most $k\alpha$ other A_j's. Using the symmetric version of the Lovász Local Lemma we get that, if $e(k\alpha+1) \leq 2^\alpha$ then $\Pr\left[\bigcap \overline{A_i}\right] > 0$.

Notice that $\alpha = \lfloor 2\log(4k) \rfloor$ satisfies the previous inequality for all positive integers k. So there exists a new formula $\mathtt{C}'$ such that every group will have at least one positive literal. Let g be the function computed by $\mathtt{C}'$. Note that we now have a fixed σ such that for all x, $f(x \oplus \sigma) = g(x)$.

On any input $x \in g^{-1}(1)$ we get at least one parse tree in $\mathtt{C}'$ all of whose gates evaluate to 1. Consequently, on any input x in $g^{-1}(1)$, there are at least A bottom $\wedge$-gates of $\mathtt{C}'$ which evaluate to 1. As each variable can appear at most p times in any parse tree, we have that $\forall x \in g^{-1}(1)$, $|x|_1 \geq \left\lceil \frac{A}{p} \left\lfloor \frac{a_d}{\alpha} \right\rfloor \right\rceil \geq \frac{A(a_d - 2\log 4k + 1)}{2p\log 4k}$.

Taking the input $x \in g^{-1}(1)$ with least Hamming weight we get that,

$$\mathrm{s}_1(f, x \oplus \sigma) = \mathrm{s}_1(g, x) \geq |x|_1 \geq \left(\frac{a_d - 2\log 4k + 1}{2p\log 4k}\right) A.$$

◀

It is interesting to notice that the proof can be turned into an algorithm for finding an input which has high sensitivity given any 1-input x. Namely, one just have to run the algorithmic version of Lovász Local Lemma to get the above bijection $\oplus\sigma$. Then find (by flipping the 1's from x) a locally minimal weight (under $\oplus\sigma$) assignment that still gives the output 1.

We will now remove the condition that every variable can occur at most once under any bottom gate. In doing so we will lose a factor of k in the lower bound while also demanding a stronger constraint on the bottom fanin. This time around we use the general version of the Lovász Local Lemma to transform $\mathtt{C}$ to $\mathtt{C}'$. The rest of the proof then follows along similar lines to the proof of Lemma 19. The proof can be found in the full version.

▶ **Lemma 20.** *Let $(a_1, \ldots, a_d) \in (\mathbb{N} \setminus \{0\})^d$ with $a_d \geq k\log(3k)$. Let f be a non-constant function computed by an $(a_1, \ldots, a_d)$-regular read-k and parse-read p formula such that the bottom gates are $\wedge$-gates. Then*

$$\mathrm{s}_1(f) \geq \left(\frac{a_d - k\log(3k) + 1}{kp\log(3k)}\right) A.$$

4.1.2 The Sensitivity Conjecture for large bottom fan-in case

We will now combine previously known results with the statements proved in the section above to obtain some relations between sensitivity and block sensitivity.

The next lemma will help us relate the bound obtained for $\mathrm{s}_1(f)$ to $\mathrm{C}_1(f)$ of read-k regular formulas. The proof follows by induction and can be found in the full version.

▶ **Lemma 21.** *Let f be a Boolean function computed by an $(a_1, \ldots, a_d)$-regular formula $\mathtt{C}$. Then, $\mathrm{C}_1(f) \leq A(\mathtt{C}, d)$.*

▶ **Theorem 22.** *Let f be a non-constant Boolean formula computed by an $(a_1, \ldots, a_d)$-regular read-k formula with parse-read p such that its bottom fanin a_d is larger or equal to $(3\log 4k)$ and such that any variable appears at most one time under each bottom gate. Then*

$$\mathrm{s}(f) \geq \sqrt{\frac{\mathrm{bs}(f)}{10p\log 4k}}.$$

Moreover, when a variable can occur multiple times under each bottom gate and the bottom fan-in $a_d \geq 2k\log 3k$, we have

$$\mathrm{s}(f) \geq \sqrt{\frac{3\mathrm{bs}(f)}{10kp\log 3k}}.$$

Proof. Let us start by the first point of the theorem. By considering f or $\neg f$, we can assume that the bottom layer is composed of $\wedge$-gates. By Lemma 19, we have that,

$$\mathrm{s}_1(f) \geq \left(\frac{a_d - 2\log 4k + 1}{2p\log 4k}\right) A.$$

From Lemma 21 we have $C_1(f) \leq a_d A$. Since $a_d \geq 3\log 4k$, $a_d - 2\log 4k \geq a_d/3$,

$$\mathrm{s}_1(f) \geq \frac{3A + \mathrm{C}_1(f)}{6p\log 4k} \geq \frac{\mathrm{C}_1(f) + 1/2}{6p\log 4k}.$$

Using Lemma 8 we get, $\mathrm{s}(f) \geq \mathrm{s}_1(f) \geq \frac{\mathrm{bs}_1(f)}{6p\log 4k}$. We also get by Theorem 9,

$$\mathrm{s}(f)^2 \geq \mathrm{s}_1(f) \cdot \mathrm{s}_0(f) \geq \frac{\mathrm{bs}_0(f)}{4p\log 4k}.$$

Since $\mathrm{bs}(f) = \max(\mathrm{bs}_1(f), \mathrm{bs}_0(f))$, $5\mathrm{s}^2 \geq 2\mathrm{s}^2 + 3\mathrm{s} \geq \dfrac{\mathrm{bs}_0(f)}{2p\log 4k} + \dfrac{\mathrm{bs}_1(f)}{2p\log 4k} \geq \dfrac{\mathrm{bs}(f)}{2p\log 4k}$.

Consequently, $\mathrm{s} \geq \sqrt{\dfrac{\mathrm{bs}(f)}{10p\log 4k}}$, proving the first part. The second part of the theorem follows analogously using Lemma 20. ◀

The following corollary follows from the lower bound for sensitivity proved in [19]. A detailed proof can be found in the full version of the paper.

▶ **Corollary 23.** *Let f be a non-constant Boolean formula computed by an $(a_1, \ldots, a_d)$-regular read-$(\log n)$ formula with bottom fan-in at least $\log^2 n$. Then $\mathrm{s}(f) \geq \tilde{\Omega}\left(\mathrm{bs}(f)^{1/4}\right)$ where the $\tilde{\Omega}$ notation hides some logarithmic terms.*

4.2 Removing the condition on the bottom fan-in

In this section, we complete the proof of Theorem 16. We note that when the depth is constant but the size of the formula is large enough, there has to be a level at which the fanin is sufficiently large. If one of the last two fanins is large, we can apply an argument quite similar to the one in the previous section. Otherwise, we can switch these two layers while incurring a significant blow-up (but still only as a function of depth and read-multiplicity) in certain circuit parameters, while reducing the depth of the circuit. We continue switching the last two layers until one of their fanins is sufficiently large, which is ensured because the circuit is of constant depth.

4.2.1 Normal form by switching:

For notational convenience, we number the layers of a depth-d circuit as $L_1, \ldots, L_d$ with L_1 being just the root (output) gate and L_d the bottom layer (with inputs feeding into them) of gates. Also, we define the following function over $\mathbb{N}$ for later reference:

$$H(x) := 24 \cdot (3x)^{2x} x^4 \log 3x. \qquad (1)$$

As mentioned above, we will transform our formula into an equivalent formula where the fanin in the last or the last but one layer is sufficiently large. Such a representation for Boolean functions will be called a normal form:

▶ **Definition 24.** A formula is in $(k; a_1, \ldots, a_d)$-normal form if the following properties hold:

1. the formula is alternating and $(a_1, \ldots, a_d)$-regular, i.e., fanin of all gates in L_i is a_i,
2. the formula is read-k,
3. the bottom layer L_d is composed of $\wedge$-gates,
4. at least one of the two following conditions on the fanins of the two bottom layers L_{d-1} and L_d is true:
   - $a_d \geq 2k \log 3k$,
   - under each $\vee$-gate in L_{d-1}, i.e., one layer above the bottom layer, there are at least $H(k)$ non-constant $\wedge$-bottom gates.

As we will switch adjacent layers of the formula, let us start by bounding the increase we get by such a procedure. Let the size and width of a DNF (respectively CNF) be the fanin of its first layer and second layer respectively.

▶ **Lemma 25.** *If f is a function computed by a read-k regular DNF (respectively CNF) of size (top fanin) a and width b, then it is also computed by a read-$(kb^{(a-1)})$ CNF (respectively DNF) of size b^a and width a.*

Now we will focus on the last two layers we get after some number of switches in the formula. We will recursively define certain functions T_i below. Intuitively, T_1 is the fanin of the bottom layer without any switches and T_{i+2} is the fanin of the layer just above the bottom layer after i switching steps. Note that a depth d circuit becomes a depth $d - i$ circuit after i switches and merges of adjacent layers (after switching) of gates of the same type. Thus T_{i+2} is the fanin of layer L_{d-i-1} in the transformed circuit after i applications of switching and merging.

Formally, the family of functions $T_i : \mathbb{N}^i \to \mathbb{N}$, where i is a positive integer, is defined as

$$\begin{cases} T_0 = 1 \\ T_1(a) = a \\ T_p(a_1, \ldots, a_p) = a_1 \cdot (T_{p-2}(a_3, \ldots, a_p))^{T_{p-1}(a_2, \ldots, a_p)} & \text{if } p \geq 2. \end{cases}$$

In what follows, the function T_i will almost always be evaluated on the fanins of the last i layers of the formula. So, we will sometimes use the shorter notation $T_i(\mathbf{a})$ to designate $T_i(a_{d-i+1}, \ldots, a_d)$.

Observe that most of the non-regular formulas can be converted into a regular one by inserting gates or subtrees of gates that compute identically constant functions. Since we want to avoid this, we will define *purely regular formulas* as regular formulas in which each internal gate computes a non-constant Boolean function.

In the next claim, we compute the parameters of our new formula after several switches. The proof of the claim is by induction on the number of switchings i and the details can be found in the full version of the paper.

▶ **Claim 26.** *Suppose f is computed by a purely $(a_1, \ldots, a_d)$-regular read-k formula. Then for all integers $i \in [0, d-2]$, f is computable by an $(a_1, \ldots, a_{d-i-2}, u, v)$-regular read-$\Big(kuv/(\prod_{j=d-i-1}^{d} a_j)\Big)$ formula where*

$$u = T_{i+2}(a_{d-i-1}, \ldots, a_d) \quad \text{and} \quad v = T_{i+1}(a_{d-i}, \ldots, a_d)$$

such that under any gate in layer L_{d-i-1}, i.e., one layer above the bottom layer of gates, there are at least a_{d-i-1} non-constant bottom gates.

Recall the function $H(x)$ from (1). We inductively define $R_i(k)$ as

$$\begin{cases} R_0(k) = R_1(k) = k \\ R_p(k) = k \prod_{j=1}^{p-1} T_j(H(R_{j-1}(k)), \ldots, H(R_0(k)))^{T_{j+1}(H(R_j(k)), \ldots, H(R_0(k))) - 1} & \text{if } p \geq 2. \end{cases}$$

Intuitively, the $R_i(k)$'s bound the read value of the formula after $i-1$ switches of the bottom layers. As the functions R_p will always be used on the parameter k (the read value of the original formula), we will usually denote $R_p(k)$ by the simpler notation R_p.

We are now ready to prove that we can transform a sufficiently large regular formula into a formula in normal form. Proof of the following lemma appears in the full version.

▶ **Lemma 27.** *If f is computed by a purely $(a_1, \ldots, a_d)$-regular read-k formula with size larger than $H(R_d)$ then there exists $i \in [0, d-2]$ such that either f or $\neg f$ can be computed by a formula in $(R_{i+1}; a_1, \ldots, a_{d-i-2}, u, v)$-normal form with*

$$u = T_{i+2}(a_{d-i-1}, \ldots, a_d) \text{ and } v = T_{i+1}(a_{d-i}, \ldots, a_d).$$

Moreover,

- *the index i is such that for any $p \geq d-i$ we have $a_p \leq H(R_{d-p})$, and*
- *under each gate in one layer above the bottom one, i.e., L_{d-i-1}, there are at least a_{d-i-1} non-constant gates, where $a_{d-i-1} \geq H(R_{i+1})$.*

Now since our new formula's last or last but one fanin is sufficiently large, we can prove a lower bound on the sensitivity as was done in Theorem 22. The sketch of the proof is similar to the one of Theorem 22, but the fact that we now consider the last two layers (instead of the last layer only) makes details a bit more complicated.

▶ **Theorem 28.** *If f is computed by a purely $(a_1, \ldots, a_d)$-regular read-k formula with size larger than $H(R_d(k))$, then*

$$\mathrm{s}(f) \geq \sqrt{\frac{3\mathrm{bs}(f)}{5R_{d-1}(k)H(R_{d-1}(k))^{(d+1)/2}}} = \Omega_{k,d}(\sqrt{\mathrm{bs}(f)}).$$

Since the function $R_{d-1}(k)$ only depends on d and k, Theorem 16 immediately follows. One can notice that the hidden constant in this theorem is approximatively the inverse of the tetration ${}^{2d-2}k = \underbrace{k^{k^{\cdot^{\cdot^{k}}}}}_{2d-2}$.

Proof of Theorem 28. By Lemma 27, we know that f (or $\neg f$) can be computed by a $(k', a_1, \ldots, a_{d'-2}, u, v)$-regular formula in normal form where $k' = R_{d-d'+1}(k)$. Here d' is the depth of the new (equivalent) formula after applying $d - d'$ switches and merges. If the bottom fanin is larger than $2k' \log(3k')$ (the first condition for the fanins in the normal form) then using Theorem 22 we get that,

$$\mathrm{s}(f) \geq \frac{1}{k'}\sqrt{\frac{3\mathrm{bs}(f)}{10\log 3k'}} \geq \sqrt{\frac{3\mathrm{bs}(f)}{5R_{d-1}(k)H(R_{d-1}(k))^{(d+1)/2}}}.$$

Otherwise we have that under each gate in $L_{d'-1}$, there are at least $a_{d'-1} \geq H(R_{d-d'+1})$ non-constant bottom gates.

In this case, we want to give a similar argument as in proof of Lemma 19 for the last but one layer instead of the last layer. Hence, we would like to have $\wedge$-gates at the last but one layer. So we will consider $(\neg f)$ if necessary. By Lemma 27, such a bottom $\wedge$-gate of $\mathtt{C}$ is the parent of at least $a_{d'-1}$ non-constant bottom $\vee$-gates. Let us group these non-constant $\vee$-gates into groups of size $\alpha = \lfloor H(k')/2 \rfloor$. We now get $\mathtt{C}'$ from $\mathtt{C}$ so that each group contains at least one $\vee$-gate which has only positive literals under it. Let g be the function computed by $\mathtt{C}'$.

Using a similar argument as in proof of Lemma 19 (proved in the full version),

▶ **Claim 29.** *For all x in $g^{-1}(1)$*

$$|x|_1 \geq \left\lceil \frac{A'}{k'} \left\lfloor \frac{a_{d'-1}}{\alpha} \right\rfloor \right\rceil \geq \frac{A'(2a_{d'-1} - H(k') + 1)}{k'H(k')} \quad \textit{with } A' = A(\mathtt{C}, d'-2).$$

Taking the input $x \in g^{-1}(1)$ with least Hamming weight we get that,

$$\mathrm{s}_1(g, x) \geq |x|_1 \geq \frac{A'(2a_{d'-1} - H(k') + 1)}{k'H(k')} \geq \frac{A'a_{d'-1} + 1}{k'H(k')} \geq \frac{A(\mathtt{C}, d) + 1}{k'H(k')H(R_{d-3})^{(d-d'+1)/2}}$$

since for any $p \geq d' + 1$ we have that $a_p \leq H(R_{d-p}) \leq H(R_{d-3})$ and we only need to consider alternate layers in the definitions of A and A'.

Since the circuit is in normal form we know that $k' \leq R_{d-d'+1}(k) \leq R_{d-1}(k)$. Using a proof similar to Lemma 19 we get that, $s(f) \geq \sqrt{\frac{3\mathrm{bs}(f)}{5R_{d-1}(k)H(R_{d-1}(k))^{(d+1)/2}}}$. ◀

5 Sensitivity Lower Bounds for DNFs

In this section, we get sensitivity lower bounds for functions computed by read-restricted DNFs. A DNF is said to be *minimal* if no proper sub-formula of such a DNF computes the same function.

Notation: For a DNF $\mathtt{C}$ let a_1 denote its top fanin and $a_{21}, \ldots, a_{2a_1}$ its bottom fanins, with $a_2 = a_{21} \geq a_{22} \geq \ldots \geq a_{2a_1}$.

5.1 Regular read-k DNFs of large width

We can adapt Corollary 23 in the case where the DNF in question is regular and its width is sufficiently large:

▶ **Corollary 30.** *Let f be a Boolean function computed by a minimal regular DNF of size n^c, for some $c > 0$ with width larger than or equal to $6 + 3c \log n$. Then,* $\mathrm{s}(f) \geq \dfrac{\mathrm{bs}(f)^{1/3}}{2\sqrt{5}\max(2,c)}$.

5.2 Read-k DNFs of small size

In this section we will remove the constraints of regularity and large width for DNFs, thus proving the sensitivity conjecture for all functions computed by read-k DNFs.

The first lemma ensures a lower bound on $\mathrm{s}_0(f)$ for functions computed by read-k DNFs. The proof can be found in the full version.

▶ **Lemma 31.** *Let f be a Boolean formula computed by a minimal read-k DNF* $\mathtt{C}$. *Then* $\mathrm{s}_0(f) \geq \frac{a_1}{ka_2}$.

The second lemma states that the sensitivity of a read-k DNF is lower bounded by a function of its maximum bottom fanin.

▶ **Lemma 32.** *Let f be a Boolean function computed by a minimal read-k DNF* $\mathtt{C}$.
Then $\mathrm{s}_1(f) + (1+k)\mathrm{s}_0(f) \geq a_2$.

Proof. Let the bottom $\wedge-$gates be $W_1, \ldots, W_{a_1}$ with fanins $a_2 = a_{21} \geq \ldots \geq a_{2a_1}$ respectively. Let the variables under W_i be $x_{i1}, \ldots, x_{ia_{2i}}$.

Let us define two sets:
- $z \in P_1$ if and only if $W_1(z) = 1$ and for all $j > 1$, $W_j(z) = 0$,
- $y \in P_2$ if and only if $W_1(y) = 1$ and y is sensitive on the variable x_{11}.

By minimality of $\mathtt{C}$, we can find an input
- z_0 in P_1, otherwise removing the gate W_1 would not modify the function,
- y_0 in P_2, otherwise we can remove the leaf corresponding to x_{11} from W_1.

In fact it would be great to find an input which belongs to both P_1 and P_2, but unfortunately, it is not always possible. However, we show we can find such a pair (z, y) such that the Hamming distance between them is small. The next two claims are proved in the full version.

▶ **Claim 33.** *There exists a pair of inputs $(z_1, y_1) \in P_1 \times P_2$ such that the Hamming distance between z_1 and y_1 is at most $\mathrm{s}_0(f) - 1$.*

Let $J \subseteq [2, a_2]$ be the variables which appear under W_1 and which are sensitive on z_1, so $\mathrm{s}_1(f) \geq |J|$. Let $\bar{J} = [2, a_2] \setminus J$. Hence, it is sufficient to show:

▶ **Claim 34.** $\mathrm{s}_0 \geq |\bar{J}| - k\mathrm{s}_0 + 1$.

◀

▶ **Theorem 35.** *Let f be a Boolean formula computed by a read-k DNF.*
Then $(k+2)\mathrm{s}(f) \geq n^{1/3}$. In particular, if $k \leq n^{\frac{1}{3}-\varepsilon} - 2$, we get $\mathrm{s}(f) \geq n^{\varepsilon} \geq \mathrm{bs}(f)^{\varepsilon}$.

Proof. Using Lemma 32 we get that, $(k+2)\mathrm{s}(f) \geq \mathrm{s}_1(f) + (1+k)\mathrm{s}_0(f) \geq a_2$. By Lemma 31 we know that, $\mathrm{s}_0(f) \geq \frac{a_1}{ka_2} \geq \frac{n}{ka_2^2}$. Combining these two inequalities we get,

$$\mathrm{s}^3(f) \geq \left(\frac{a_2}{k+2}\right)^2 \frac{n}{ka_2^2} \geq \frac{n}{k(k+2)^2}$$

and so $(k+2)\mathrm{s}(f) \geq n^{1/3}$. In particular, when $k+2 \leq n^{\frac{1}{3}-\varepsilon}$, we get $\mathrm{s}(f) \geq n^{\varepsilon} \geq \mathrm{bs}(f)^{\varepsilon}$. ◀

References

1 Scott Aaronson. The "sensitivity" of 2-colorings of the d-dimensional integer lattice. *blog-post*, July 2010. URL: `http://mathoverflow.net/questions/31482/`.

2 Noga Alon and Joel H Spencer. *The Probabilistic Method*. John Wiley and Sons, 1994.

3 Andris Ambainis, Mohammad Bavarian, Yihan Gao, Jieming Mao, Xiaoming Sun, and Song Zuo. Tighter relations between sensitivity and other complexity measures. In *International Colloquium on Automata, Languages and Programming (ICALP) 2014, Part I*, pages 101–113, 2014. `doi:10.1007/978-3-662-43948-7_9`.

4 Andris Ambainis and Krisjanis Prusis. A tight lower bound on certificate complexity in terms of block sensitivity and sensitivity. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 33–44, 2014. `doi:10.1007/978-3-662-44465-8_4`.

5 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 352–361, 1998. `doi:10.1109/SFCS.1998.743485`.

6 Meena Boppana. Lattice variant of the sensitivity conjecture. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:89, 2012.

7 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002. `doi:10.1016/S0304-3975(01)00144-X`.

8 Sourav Chakraborty. On the sensitivity of cyclically-invariant boolean functions. In *Conference on Computational Complexity (CCC)*, pages 163–167, 2005.

9 Stephen Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, 1986.

10 Justin Gilmer, Michal Koucký, and Michael E. Saks. A new approach to the sensitivity conjecture. In *Innovations in Theoretical Computer Science, ITCS*, pages 247–254, 2015.

11 Parikshit Gopalan, Rocco A. Servedio, Avishay Tal, and Avi Wigderson. Degree and sensitivity: tails of two distributions. *to appear CCC 2016*, 2016.

12 Craig Gotsman and Nathan Linial. The equivalence of two problems on the cube. *Journal of Combinatorial Theory, Series A*, 61(1):142–146, 1992.

13 Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the sensitivity conjecture. *Theory of Computing, Graduate Surveys*, 4:1–27, 2011.

14 Claire Kenyon and Samuel Kutin. Sensitivity, block sensitivity, and l-block sensitivity of Boolean functions. *Information and Computation*, 189(1):43–53, 2004.

15 Hiroki Morizumi. Sensitivity, block sensitivity, and certificate complexity of unate functions and read-once functions. *IFIP TCS*, 2014.

16 Noam Nisan. CREW PRAMs and decision trees. In *Symposium on Theory of Computing*, pages 327–335, 1989.

17 Noam Nisan and Mario Szegedy. On the degree of Boolean functions as real polynomial. In *Symposium on Theory of Computing*, pages 462–467, 1992.

18 David Rubinstein. Sensitivity vs. block sensitivity of boolean functions. *Combinatorica*, 15(2):297–299, 1995.

19 Hans-Ulrich Simon. A tight $\Omega(\log\log n)$-bound on the time for parallel RAM's to compute nondegenerated Boolean functions. *Foundations of Computation Theory, Lecture Notes in Computer Science*, 158(1):439–444, 1983.

20 Xiaoming Sun. Block sensitivity of weakly symmetric functions. *Theor. Comput. Sci.*, 384(1):87–91, 2007.

21 Avishay Tal. On the sensitivity conjecture. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2016.

22 György Turán. The critical complexity of graph properties. *Inf. Process. Lett.*, 18(3):151–153, 1984.

Graph Properties in Node-Query Setting: Effect of Breaking Symmetry

Nikhil Balaji[1], Samir Datta[2], Raghav Kulkarni[3], and Supartha Podder[4]

1 **Chennai Mathematical Institute, India &**
Indian Institute of Technology Bombay, India
nikhil@cmi.ac.in

2 **Chennai Mathematical Institute, India**
sdatta@cmi.ac.in

3 **Chennai Mathematical Institute, India**
kulraghav@gmail.com

4 **Centre for Quantum Technologies,**
National University of Singapore, Singapore
supartha@gmail.com

Abstract

The query complexity of graph properties is well-studied when queries are on the edges. We investigate the same when queries are on the nodes. In this setting a graph $G = (V, E)$ on n vertices and a property $\mathcal{P}$ are given. A black-box access to an unknown subset $S \subseteq V$ is provided via queries of the form "Does i belong to S?". We are interested in the minimum number of queries needed in the worst case in order to determine whether $G[S]$ – the subgraph of G induced on S – satisfies $\mathcal{P}$.

Our primary motivation to study this model comes from the fact that it allows us to initiate a systematic study of breaking symmetry in the context of query complexity of graph properties. In particular, we focus on the hereditary graph properties – properties that are closed under deletion of vertices as well as edges. The famous Evasiveness Conjecture asserts that even with a minimal symmetry assumption on G, namely that of vertex-transitivity, the query complexity for any hereditary graph property in our setting is the worst possible, i.e., n.

We show that in the absence of any symmetry on G it can fall as low as $O(n^{1/(d+1)})$ where d denotes the minimum possible degree of a minimal forbidden sub-graph for $\mathcal{P}$. In particular, every hereditary property benefits at least quadratically. The main question left open is: Can it go exponentially low for some hereditary property? We show that the answer is no for any hereditary property with finitely many forbidden subgraphs by exhibiting a bound of $\Omega(n^{1/k})$ for a constant k depending only on the property. For general ones we rule out the possibility of the query complexity falling down to constant by showing $\Omega(\log n/ \log\log n)$ bound. Interestingly, our lower bound proofs rely on the famous Sunflower Lemma due to Erdös and Rado.

1998 ACM Subject Classification F. Theory of Computation, F.1 Computation by Abstract Devices, F.1.1 Models of Computation, Bounded-action devices, G. Mathematics of Computing, G.2 Discrete Mathematics, G.2.2 Graph Theory

Keywords and phrases Query Complexity, Graph Properties, Symmetry and Computation, Forbidden Subgraph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.17

1 Introduction

1.1 The query model

The decision tree model (aka query model) has been extensively studied in the past and still remains a rich source of many fascinating questions. In this paper, we focus on Boolean functions, i.e., functions of the form $f : \{0,1\}^n \to \{0,1\}$ and their decision tree complexity. A deterministic decision tree D_f for f takes $x = (x_1, \dots, x_n)$ as an input and determines the value of $f(x_1, \dots, x_n)$ using queries of the form "is $x_i = 1$?". Let $C(D_f, x)$ denote the cost of the computation, that is the number of queries made by D_f on input x. The *deterministic decision tree complexity* (aka deterministic query complexity) of f is defined as

$$D(f) = \min_{D_f} \max_{x} C(D_f, x)$$

Randomized and the Quantum variants [6] of decision trees have also been extensively studied in the past. Several different variants such as parity decision trees have been studied in connection to communication complexity, learning, and property testing [25, 20, 4]. We refer the interested reader to the excellent survey by Buhrman and de Wolf [6] for more background on decision tree complexity.

Importance of query models

Variants of the decision tree model are fundamental for several reasons: Firstly, they occur naturally in connection to the other models of computation such as communication complexity [25], property testing [4], learning [20], circuit complexity [13] etc. Secondly, decision tree models are much simpler to analyse as compared to other models such as circuits. Thus one can actually hope to use them as a tool in the study of other models. Thirdly, these models are mathematically rich and beautiful – several connections to algebra, combinatorics, topology, Fourier analysis, and number theory [22, 2] make the decision tree models interesting in their own right. Finally, there remain some fascinating open questions [17] in query complexity that have attracted the attention of generations of researchers over the last few decades by their sheer elegance and notoriety.

1.2 Graph properties in node-query setting

In this paper , we investigate the query complexity of graph properties. In particular, we focus on the following setting: A graph $G = (V, E)$ and a property $\mathcal{P}$ are fixed. We have access to $S \subseteq V$ via queries of the form "Does i belong to S?". We are interested in the minimum number of queries needed in the worst case in order to determine whether $G[S]$ – the subgraph of G induced on S – satisfies $\mathcal{P}$, which we denote by $cost(\mathcal{P}, G)$. One may define a similar notion of cost for randomized and quantum models.

We call G the base graph for $\mathcal{P}$. We say that a vertex i of G is relevant for $\mathcal{P}$ if there exists some S containing i such that exactly one of $G[S]$ and $G[S - \{i\}]$ satisfies $\mathcal{P}$. We say that G is relevant for $\mathcal{P}$ if all its vertices are relevant for $\mathcal{P}$. The minimum possible cost of $\mathcal{P}$, denoted by[1] *min-cost*$(\mathcal{P})$, is defined as follows:

$$\textit{min-cost}_n(\mathcal{P}) = \min_{G} \{ cost(\mathcal{P}, G) \mid G \text{ is relevant for } \mathcal{P} \ \& \ |V(G)| = n \}.$$

[1] We slightly abuse this notation by omitting the subscript n.

Note that in the node-query settings the notion of *relevance of a graph G for the property* $\mathcal{P}$ is important because if any vertex $v \in G$ is *not* relevant then v cannot possibly influence the output of the function and hence any query algorithm does not need to query it.

Similarly one can define *max-cost*$(\mathcal{P})$, which is a more natural notion of complexity when one is interested in studying the universal upper bounds. Investigating the *max-cost* in our setting can indeed be a topic of an independent interest. However, for the purpose of this paper, the notion of *min-cost* will be more relevant as we are interested in finding how low can the universal lower bound on query complexity go under broken symmetry (Refer to Section 1.3 for more on symmetry). It turns out that in the presence of symmetry this bound is $\Omega(n)$ for most of the properties and it is conjectured to be $\Omega(n)$ for any hereditary property in our setting. Recall that a hereditary property is a property of graphs, which is closed under deletion of vertices as well as edges. For instance acyclicity, bipartiteness, planarity, and triangle-freeness are hereditary properties whereas connectedness and containing a perfect matching are not. Every hereditary property can be described by a (not necessarily finite) collection of its forbidden subgraphs.[2] [3]

It appears that the node-query setting is a natural abstraction of scenarios where one is interested in the properties of the subgraph induced by active nodes in a network. We discuss three such examples in the Appendix of the full version of this paper. To the best of our knowledge, no systematic study of node-query setting has been yet undertaken. Here we initiate such a line of inquiry for graph properties. In particular, we focus on the role of presence and absence of *symmetry*.

1.3 Effect of breaking symmetry

The primary reason why we are interested in the node-query model is that it allows us to study the effect of breaking symmetry on query complexities of graph properties. In particular, our setting provides a platform to compare the complexity of $\mathcal{P}$ when the base graph G has certain amount of symmetry with the complexity of $\mathcal{P}$ when G has no symmetry whatsoever. To formalize this, we define the notion of $\mathcal{G}$-*min-cost*$(\mathcal{P})$ for a class of graphs $\mathcal{G}$ by restricting ourselves only to graphs in $\mathcal{G}$.

$$\mathcal{G}\text{-}\mathit{min\text{-}cost}_n(\mathcal{P}) = \min_{G \in \mathcal{G}} \{\mathit{cost}(\mathcal{P}, G) \mid G \text{ is relevant for } \mathcal{P} \text{ \& } |V(G)| = n\}.$$

When $\mathcal{G}$ has the highest amount of symmetry, i.e., when $\mathcal{G}$ is the class of complete graphs, then it is easy to see that for every hereditary $\mathcal{P}$, $\mathcal{G}$-*min-cost*$(\mathcal{P})$ is nearly the worst possible, i.e., $\Omega(n)$. It turns out that one does not require the whole symmetry of the complete graph to guarantee the $\Omega(n)$ bound. Even weaker symmetry assumptions on graphs in $\mathcal{G}$, for instance being Cayley graphs of some group, indeed suffices. Thus it is natural to ask how much symmetry is required to guarantee the $\Omega(n)$ bound. In fact, the famous Evasiveness Conjecture implies that even under the weakest form of symmetry on $\mathcal{G}$, i.e., when $\mathcal{G}$ is the class of transitive graphs, for any hereditary property $\mathcal{P}$ the $\mathcal{G}$-*min-cost*$(\mathcal{P})$ would remain the highest possible, i.e., n. So for the complexity to fall down substantially we might have to let go of the transitivity of $\mathcal{G}$. This is exactly what we do. In particular we take $\mathcal{G}$ to be the class of all graphs, i.e., we assume no symmetry whatsoever. Note that in this case

[2] In our setting, every hereditary property is a monotone Boolean function.

[3] We would like to highlight that although we didn't explicitly define *min-cost*$(\mathcal{P})$ or *max-cost*$(\mathcal{P})$ for randomized query model, all our lower bound proofs are based on sensitivity arguments and hence work even for randomized case.

$\mathcal{G}$-*min-cost*$(\mathcal{P})$ = *min-cost*$(\mathcal{P})$ that we defined earlier. Now a natural question is how low can *min-cost*$(\mathcal{P})$ go in the absence of any symmetry? This is the main question addressed by our paper. In particular, we show that for any hereditary property $\mathcal{P}$, the *min-cost*$(\mathcal{P})$ falls down at least quadratically, i.e, to $O(\sqrt{n})$. For some properties, it can go even further below (polynomially down) with polynomials of arbitrary constant degree, i.e. to $O(n^{1/k})$ where k is a constant depending only on the property. The main question left open by our work is: does there exist a hereditary property $\mathcal{P}$ for which *min-cost*$(\mathcal{P})$ is exponentially low? In other words:

▶ Question 1. Is it true that for every hereditary property $\mathcal{P}$ there exists an integer $k_{\mathcal{P}} > 0$ such that

$$\textit{min-cost}(\mathcal{P}) = \Omega(n^{1/k_{\mathcal{P}}})?$$

1.4 Related work

Understanding the effect of symmetry on computation is a very well-studied theme in the past. Perhaps its roots can also be traced back to the non-solvability of quintic equations by radicals – the legendary work of Galois [1]. In the context of query complexity, again there has been a substantial amount of effort invested in understanding the role of symmetry. A recurrent theme here is to exploit the symmetry and some other structure [19] of the underlying functions to prove good lower bounds on their query complexity. For instance the famous Andera-Rosenberge-Karp Conjecture [15] asserts that every non-trivial monotone graph property of n vertex graphs (in the edge-query model) must be evasive, i.e., its query complexity is $\binom{n}{2}$. While a weaker bound of $\Omega(n^2)$ is known, the conjecture remains widely open to this date. Several special cases of the conjecture have also been studied [7]. The randomized query complexity of monotone graph properties is also conjectured to be $\Omega(n^2)$ [10]. The generalizations of these conjectures for arbitrary transitive Boolean functions are also studied: In particular, recently Kulkarni [16] has formulated the Weak-Evasiveness Conjecture for monotone transitive functions, which vastly generalize monotone graph properties. In the past, Lovász had conjectured [14] the evasiveness of checking independence of S exactly in our setting. Sun,Yao, and Zhang [24] study query complexity of graph properties and several transitive functions including the circulant ones. Their motivation was to investigate how low can the query complexity go if one drops the assumption of monotonicity or lower the amount of symmetry. In this paper, we follow their footsteps and ask the same question under no symmetry assumption whatsoever. The main difference between the past works and this one is that most of the previous work exploit the symmetry to prove (or to conjecture) a good lower bound, whereas we investigate the consequences of breaking the symmetry for the query complexity.

1.5 Our main results

In this section we summarize our main results. Let $\mathcal{P}$ be a hereditary graph property and $d_{\mathcal{P}}$ denote the minimum possible degree of a minimal forbidden subgraph for $\mathcal{P}$.

▶ **Theorem 2.** *For any hereditary graph property $\mathcal{P}$:*

$$\textit{min-cost}(\mathcal{P}) = O(n^{1/(d_{\mathcal{P}}+1)}).$$

Table 1 Summary of Results for Finite/Infinite Forbidden Subgraphs.

	Properties		With Symmetry[4]	Without Symmetry
Finite	Independence/Emptiness	[Full Version]	$\Theta(n)$	$\Theta(\sqrt{n})$
	Bounded Degree	[Full Version]	$\Theta(n)$	$\Theta(\sqrt{n})$
	Triangle-freeness	[Full Version]	$\Theta(n)$	$\Theta(n^{1/3})$
	Containing K_t	[Thm. 2][Thm. 4]	$\Theta(n)$	$\Theta(n^{1/t})$
	Containing P_t	[Thm. 2][Thm. 4]	$\Theta(n)$	$O(\sqrt{n})$,$\Omega(n^{1/t})$
	Containing C_t	[Thm. 2][Thm. 4]	$\Theta(n)$	$O(n^{1/3})$, $\Omega(n^{1/t})$
	Containing H: $V(H) = k$	[Thm. 13][Thm. 2][Thm. 4]	$\Theta(n)$	$O(n^{1/(d_{min}+1)})$, $\Omega(n^{1/k})$
Infinite	Acyclicity	[Thm. 15]	$\Theta(n)$	$O(n^{1/3})$
	Bi-partiteness	[Thm. 2]	Open	$O(n^{1/3})$
	3-colorability	[Thm. 2]	Open	$O(n^{1/4})$
	Planarity	[Thm. 17]	$\Theta(n)$[5]	$O(n^{1/4})$

▶ **Corollary 3.** *For any hereditary graph property $\mathcal{P}$:*

$$\textit{min-cost}(\mathcal{P}) = O(\sqrt{n}).$$

Theorem 2 and Corollary 3 show that in the absence of any symmetry on the graph G the query complexity can fall as low as $O(n^{1/(d+1)})$ where d denotes the minimum possible degree of a minimal forbidden sub-graph for $\mathcal{P}$. In particular, every hereditary property benefits at least quadratically.

We note that the above upper bound does not hold for general graph properties. For instance Connectivity has *min-cost* $\Theta(n)$, so does containment of a Perfect Matching, which are both non-hereditary properties (See Appendix of the full version of this paper).

As a partial answer to Question 1 we prove the following theorem.

▶ **Theorem 4.** *Let H be a fixed graph on k vertices and let $\mathcal{P}_H$ denote the property of containing H as a subgraph. Then,*

$$\textit{min-cost}(\mathcal{P}_H) = \Omega(n^{1/k}).$$

Interestingly our proof of Theorem 4 uses the famous Sunflower Lemma due to Erdös and Rado [9]. Moreover it generalizes to any fixed number of forbidden subgraphs each on at most k vertices. This implies that any hereditary property with *finitely many forbidden subgraphs* has a lower bound of $\Omega(n^{1/k})$, for a constant k depending only on the property.

We note that both Theorem 2 and Theorem 4 are not tight. However, we do prove tight bounds for several hereditary properties. We summarize a few such interesting bounds in the Table 1.

Finally we note a non-constant lower bound, which holds for *any* hereditary property. Our proof again relies on the Sunflower Lemma.

▶ **Theorem 5.** *For any hereditary graph property $\mathcal{P}$*

$$\textit{min-cost}(\mathcal{P}) = \Omega\left(\frac{\log n}{\log\log n}\right).$$

As we use sensitivity arguments all our lower bounds work for randomized case as well.

[4] assuming Weak Evasiveness
[5] when $d(G) \geq 7$

1.6 Organization

The rest of the paper is organized as follows: We introduce some preliminary notions in Section 2. We revisit some results on Weak Evasiveness under symmetry in Section 3. In Section 4, we provide proofs of Theorem 2 and Theorem 4. Proof of some tight bounds for Theorem 2 are deferred to Appendix. In Section 5 we state some results on restricted graph classes and their proofs are deferred to Appendix. Finally in Section 6 we discuss questions and directions that are naturally raised by our work.

The whole Appendix section of this paper can be found in the full version, which is available on the arXiv [3].

2 Preliminaries

▶ **Definition 6** (Randomized query complexity)**.** A randomized decision tree $\mathcal{T}$ is simply a probability distribution on the deterministic decision trees $\{T_1, T_2, \ldots\}$ where the tree T_i occurs with probability p_i. We say that $\mathcal{T}$ computes f correctly if for every input x: $\Pr_i[T_i(x) = f(x)] \geq 2/3$. The depth of $\mathcal{T}$ is the maximum depth of a T_i. The (bounded error) randomized query complexity of f, denoted by $R(f)$, is the minimum possible depth of a randomized tree computing f correctly on all inputs.

▶ **Definition 7** (Monotone, Transitive and Evasive Boolean functions)**.** A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to be *monotone* increasing if for any $x \leq y$, we have $f(x) \leq f(y)$, where $x \leq y$ means $x_i \leq y_i$ for all $i \in [n]$. Similarly one can define a monotone decreasing function. A Boolean function $f(x_1, \ldots, x_n)$ is said to be *transitive* if there exists a group G that acts transitively on the variables x_is such that f is invariant under this action, i.e., for every $\sigma \in G$: $f(x_{\sigma_1}, \ldots, f_{\sigma_n}) = f(x_1, \ldots, x_n)$. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to be *evasive* if $D(f) = n$.

▶ **Definition 8** (Hereditary graph properties)**.** A property $\mathcal{P}$ of graphs is simply a collection of graphs. The members of $\mathcal{P}$ are said to satisfy $\mathcal{P}$ and non-members are said to fail $\mathcal{P}$. A property is hereditary if it is closed under deletion of vertices as well as edges[6]. For instance: acyclicity, planarity, and 3-colorability are hereditary properties, whereas connectivity and containing a perfect matching are not. Every hereditary property $\mathcal{P}$ can be uniquely expressed as a (possibly infinite) family $\mathcal{F}_{\mathcal{P}}$ of its forbidden subgraphs. For instance: acyclicity can be described as forbidding all cycles. Given a graph G, a hitting set $S_{G,\mathcal{P}}$ for $\mathcal{P}$ is a subset of $V(G)$ such that removing $S_{G,\mathcal{P}}$ from G would make the property $\mathcal{P}$ present[7]. Hereditary graph properties in node-query setting are monotone decreasing Boolean functions. Sometimes we refer hereditary properties by their negation. For instance: containing triangle.

▶ **Definition 9** (Sensitivity and block-sensitivity [12])**.** The i^{th} bit of an input $x \in \{0,1\}^n$ is said to be sensitive for $f : \{0,1\}^n \to \{0,1\}$ if $f(x_1, \ldots, x_i, \ldots, x_n) \neq f(x_1, \ldots, 1 - x_i, \ldots, x_n)$. The sensitivity of f on x, denoted by $s_{f,x}$ is the total number of sensitive bits of x for f. The sensitivity of f, denoted by $s(f)$, is the maximum of $s_{f,x}$ over all possible choices of x. A block $B \subseteq [n]$ of variables is said to be sensitive for f on input x, if flipping the values of all x_i such that $i \in B$ and keeping the remaining x_i the same, results in flipping the output of f. The block sensitivity of f on an input x, denoted by $bs_{f,x}$ is the maximum

[6] on the other hand, vertex-hereditary is closed only under vertex-deletion (e.g. being chordal).
[7] such that every graph in $\mathcal{F}_{\mathcal{P}}$ shares a node with $S_{\mathcal{G},\mathcal{P}}$.

number of *disjoint* sensitive blocks for f on x. The block sensitivity of a function f, denoted by $bs(f)$, is the maximum value of $bs_{f,x}$ over all possible choices of x. It is known that $D(f) \geq R(f) \geq bs(f) \geq s(f)$. For monotone functions, $bs(f) = s(f)$.

3 Presence of symmetry in node-query setting: Does it guarantee weak-evasiveness?

In edge-query setting, Aanderaa-Rosenberg-Karp Conjecture [15, 7] asserts that any non-trivial monotone graph property must be evasive, i.e., one must query all $\binom{n}{2}$ edges in worst-case. The following generalization of the ARK Conjecture asserts that only monotonicity and modest amount of symmetry, namely transitivity, suffices to guarantee the evasiveness [21].

▶ Conjecture 10 (Evasiveness Conjecture). Any non-constant monotone transitive function f on n variables has $D(f) = n$.

This conjecture appears to be notoriously hard to prove even in several interesting special cases. Recently Kulkarni [16] formulates:

▶ Conjecture 11 (Weak Evasiveness Conjecture). If f_n is a sequence of monotone transitive functions on n variables then for every $\epsilon > 0$:

$$D(f_n) = \Omega(n^{1-\epsilon}).$$

Although Weak EC appears to be seemingly weaker, Kulkarni [16] observes that it is equivalent to the EC itself. His results hint towards the possibility that disproving Weak EC might be as difficult as separating TC^0 from NC^1. However: proving special cases of Weak EC appears to be relatively less difficult. In fact, Rivest and Vuillemin [23] confirm the Weak EC for graph properties and recently Kulkarni, Qiao, and Sun [18] confirm Weak EC for 3-uniform hyper graphs and Black [5] extends this result to k-uniform hyper graphs. All these results are studied in the edge-query setting. It is natural to ask whether the Weak EC becomes tractable in node-query setting. The monotone functions in node-query setting translate precisely to the hereditary graph properties. Here we show that it does become tractable for several hereditary graph properties. But first we need the following lemma [8, 24]:

▶ **Lemma 12.** *Let f be a non-trivial monotone transitive function. Let k be the size of a 1-input with minimal number of 1s. Then: $D(f) = \Omega(n/k^2)$.*

Let $\mathcal{G}_\mathcal{T}$ denote the class of transitive graphs. Let H be a fixed graph. Let $\mathcal{P}_H$ denote the property of containing H as a subgraph. The following theorem directly follows from Lemma 12.

▶ **Theorem 13.**

$$\mathcal{G}_\mathcal{T}\text{-}min\text{-}cost(\mathcal{P}_H) = \Omega(n).$$

The above result can be generalized for any finite family of forbidden subgraphs. We do not yet know how to prove it for infinite family in general. However below we illustrate a proof for one specific case when the infinite family is the family of cycles. First we need the following lemma:

▶ **Lemma 14.** *Let G be a graph on n vertices, m edges, and maximum degree d_{max}. Let $\mathcal{C}$ denote the property of being acyclic. Then,*

$$cost(\mathcal{C}, G) \geq (m - n)/d_{max}.$$

Proof. To make G acyclic one must remove at least $m-n$ edges. Removing one vertex can remove at most d_{max} edges. Thus the size of minimum feedback vertex set (FVS) is at least $(m-n)/d_{\max}$. The adversary answers all vertices outside this FVS to be present. Now the algorithm must query every vertex in the minimum FVS. ◀

▶ **Theorem 15.**

$$\mathcal{G}_{\mathcal{T}}\text{-}min\text{-}cost(\mathcal{C}) = \Omega(n).$$

Proof. Since G is transitive, G is d regular for some d [11]. Therefore $m = dn/2$ and $d_{max} = d$. Hence from Lemma 14 we get the desired bound. ◀

We also show similar bound for the property of being planar:

▶ **Lemma 16.** *Let G be a graph on n vertices, m edges, and maximum degree d_{max}. Let $\mathcal{P}'$ denote the property of being planar. Then,*

$$cost(\mathcal{P}', G) \geq (m - 3n + 6)/d_{max}.$$

Proof. To make G planar one has to remove at least $(m - 3n + 6)$ edges from the graph G. Removing one vertex can remove at most d_{max} edges. Thus the size of minimum hitting set of G is at least $(m - 3n + 6)/d_{\max}$. The adversary answers all vertices outside this minimum hitting set to be present. Now the algorithm must query every vertex in the minimum hitting set. ◀

▶ **Theorem 17.**

$$\mathcal{G}_{\mathcal{T}}\text{-}min\text{-}cost(\mathcal{P}') = \Omega(n).$$

Proof. Since G is transitive, G is d regular for some d [11]. Therefore $m = dn/2$ and $d_{max} = d$. Hence for $d \geq 7$ using Lemma 16 we get the desired bound[8]. ◀

Following special case of Weak EC remains open:

▶ Conjecture 18. For any hereditary property $\mathcal{P}$, for any $\epsilon > 0$:

$$\mathcal{G}_{\mathcal{T}}\text{-}min\text{-}cost(\mathcal{P}) = \Omega(n^{1-\epsilon}).$$

4 Absence of symmetry in node-query setting: How low can query complexity go?

4.1 A general upper bound

Let $\mathcal{P}$ be a hereditary graph property and $d_{\mathcal{P}}$ denote the minimum possible degree of a minimal forbidden subgraph for $\mathcal{P}$.

Proof of Theorem 2: Let $k = c{\cdot}n^{1/(d_{\mathcal{P}}+1)}$ where we choose the constant c appropriately. Construct a graph G on n vertices as follows (See Figure 1):

- Start with a clique on vertices $v_1, \ldots, v_k$.
- For every $S \subseteq [k]$ such that $|S| = d_{\mathcal{P}}$
  - add k new vertices $u_1^S, \ldots, u_k^S$ and
  - connect every vertex $v_i : i \in S$ to each of these new k vertices $u_1^S, \ldots, u_k^S$.

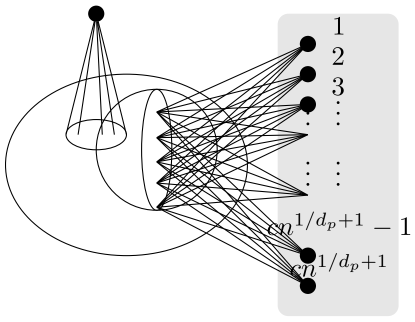


Figure 1 Construction of G for a general upper bound .

Algorithm 1:

```
- Query $v_1, \ldots, v_k$.
- If at least $c_{\mathcal{P}}$ of these vertices are present then $\mathcal{P}$ must fail.
- Otherwise there are at most $c_{\mathcal{P}} - 1$ vertices present
  (wlog: $v_1, \ldots, v_{c_{\mathcal{P}}-1}$).
    - For every subset $S \subseteq [c_{\mathcal{P}} - 1]$ such that $|S| = d_{\mathcal{P}}$, query $u_1^S, \ldots, u_k^S$.
    - If the graph induced on the nodes present (after all these
      $\binom{c_{\mathcal{P}}-1}{d_{\mathcal{P}}} \times k$ queries) satisfies $\mathcal{P}$ then answer Yes.
      Otherwise answer No.
```

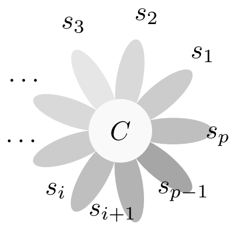
Now we describe an algorithm (See Algorithm 1) to determine $\mathcal{P}$ in $O(n^{1/(d_{\mathcal{P}}+1)})$ queries. Let $c_{\mathcal{P}}$ denote the smallest integer such that the clique on $c_{\mathcal{P}}$ vertices satisfies $\mathcal{P}$.

Note that any vertex that is not queried by the above algorithm can have at most $d_{\mathcal{P}} - 1$ edges to the vertices in the clique $v_1, \ldots, v_k$. Since $d_{\mathcal{P}}$ is the minimum degree of a minimal forbidden subgraph for $\mathcal{P}$, these vertices now become irrelevant for $\mathcal{P}$. Thus the algorithm can correctly declare the answer based on only the queries it has made. It is easy to check that the query complexity of the above algorithm is $O(k)$ which is $O(n^{1/(d_{\mathcal{P}}+1)})$. $\square$

This completes the proof of Theorem 2. Corollary 3 follows from this by observing that $d_{\mathcal{P}} \geq 2$ for any non-trivial $\mathcal{P}$.

4.2 General lower bounds

Now we show that any hereditary property with *finitely many forbidden subgraphs* has a lower bound of $\Omega(n^{1/k})$, for a constant k depending only on the property.



[8] Currently our proof works only when $d \geq 7$, but we believe that it can be extended for any degree d.

▶ **Definition 19** (Sunflower). A sunflower with core set C and p petals is a collection of sets $S_1, \ldots, S_p$ such that for all $i \neq j$: $S_i \cap S_j = C$.

We use the following lemma due to Erdös and Rado [9].

▶ **Lemma 20** (Sunflower Lemma). *Let $\mathcal{F}$ be a family of sets of cardinality k each. If $|\mathcal{F}| > k!(p-1)^k$ then $\mathcal{F}$ contains a sunflower with p petals.*

Proof of Theorem 4: Let G be a graph on n vertices such that every vertex of G is relevant for the property of containing H. Let

$$\mathcal{F} := \{S \mid |S| = k \ \& \ H \text{ is a subgraph of } G[S]\}.$$

Since every vertex of G is relevant for $\mathcal{P}_H$, we have: $|\mathcal{F}| \geq n/k$. Now from Lemma 20 we can conclude that $\mathcal{F}$ contains a sunflower on at least $|\mathcal{F}|^{1/k}/k = \Omega(n^{1/k})$ petals. Let C be the core of this sunflower. We consider the restriction of $\mathcal{P}_H$ on G where every vertex in C is present. Since $|C| < k$, $G[C]$ does not contain H. Now it is easy to check that one must query at least one vertex from each petal in order to determine $\mathcal{P}_H$.

□

Using similar technique we prove Theorem 5 showing that *min-cost*$(\mathcal{P})$ for any hereditary $\mathcal{P}$ can not fall to a constant.

▶ Theorem 5. (Restated) For any hereditary graph property $\mathcal{P}$

$$\textit{min-cost}(\mathcal{P}) = \Omega\left(\frac{\log n}{\log\log n}\right).$$

Proof. Let G be a graph on n vertices such that every vertex of G is relevant for $\mathcal{P}$. Let k be the largest integer such that G contains a minimal forbidden subgraph for $\mathcal{P}$ on k vertices. Note that we are concerned with vertex minimal certificates.

Case 1: $k \geq \frac{\log n}{2\log\log n}$.

Since one must query all the vertices of a minimal forbidden subgraph, we obtain a lower bound of $k = \Omega(\log n/\log\log n)$.

Case 2: $k < \frac{\log n}{2\log\log n}$.

Since every vertex of G is relevant for $\mathcal{P}$ and all the minimal forbidden subgraphs of $\mathcal{P}$ present in G are of size at most k, every vertex of G must belong to some minimal forbidden subgraph of size at most k. Consider the property $\mathcal{P}_k$ obtained from $\mathcal{P}$ by omitting the minimal forbidden subgraphs of $\mathcal{P}$ on k or more vertices. Our simple but crucial observation is that $\mathcal{P}$ and $\mathcal{P}_k$ are equivalent as far as G is concerned. Therefore, they have the same complexity. Now we define $\mathcal{F}_i$ for $i \leq k$ as follows:

$$\mathcal{F}_i := \{S \mid |S| = i \ \& \ G[S] \notin \mathcal{P} \ \& \ \forall T \subset S : G[T] \in \mathcal{P}\}.$$

Since every vertex of G is relevant for $\mathcal{P} \equiv \mathcal{P}_k$, we have: $|\bigcup_{i=1}^{k} \mathcal{F}_i| \geq n/k$. Since $\mathcal{F}_i$ and $\mathcal{F}_j$ are disjoint when $i \neq j$, we have $\sum_{i=1}^{k} |\mathcal{F}_i| \geq n/k$. Therefore one of the $\mathcal{F}_i$s must be of size at least n/k^2. We denote that $\mathcal{F}_i$ by $\mathcal{F}'$.

Now from Lemma 20 we can conclude that $\mathcal{F}'$ contains a sunflower on at least $|\mathcal{F}'|^{1/k}/k$ petals. Let C be the core of this sunflower. We consider the restriction of $\mathcal{P}$ on G where every vertex in C is present. Since $|C| < i$, by definition of $\mathcal{F}_i$ we must have $G[C] \in \mathcal{P}$. Now it is easy to check that one must query at least one vertex from each petal in order to determine $\mathcal{P}$. A simple calculation yields that one can obtain a lower bound of $\min\{k, \frac{2^{\Omega(\log n/k)}}{k}\}$. When $k = \log n/(2\log\log n)$, this gives us $\Omega(\log n/\log\log n)$ bound. ◀

4.3 Some tight bounds

We manage to show that Theorem 2 is tight for several special properties like Independence, Triangle-freeness, Bounded-degree etc. In the Appendix of the full version of this paper we present them in detail. In order to prove the tight bounds, we show several inequalities which might be of independent interest combinatorially. We present one such inequality below.

Lower bound based on the chromatic number

▶ **Theorem 21.** *Let $\mathcal{I}$ denote the property of being an independent subset of nodes (equivalently the property of being an empty graph). Then,*

$$\mathcal{G}\text{-}min\text{-}cost(\mathcal{I}) \geq n/\chi$$

where χ is the maximum chromatic number of a graph $G \in \mathcal{G}$.

Proof. Let $G \in \mathcal{G}$ be a graph on n vertices such that every vertex of G is relevant for $\mathcal{I}$, i.e., G does not contain any isolated vertices. Consider a coloring of vertices of G with χ colors. Let C_i denote the set of vertices colored with color i. We pick a coloring that maximizes $\max_{i \leq \chi}\{|C_i|\}$. Let C_{max} denote such a color class with maximum number of vertices in this coloring. Thus $|C_{\max}| \geq n/\chi$.

When $|C_{\max}| \leq (1 - \frac{1}{\chi})n$, the adversary answers all the vertices in $C_{\max}$ to be present. Since $C_{\max}$ is maximal and G does not contain any isolated vertices, every vertex outside $C_{\max}$ must be connected to some vertex in $C_{\max}$. As long as any of these outside vertices are present there will be an edge. Hence we get a lower bound of $n - |C_{\max}| \geq n/\chi$.

Now when $|C_{\max}| > (1 - \frac{1}{\chi})n$, since there are no isolated vertices in G, every vertex in $C_{\max}$ must have an edge to some vertex in $C_i \neq C_{\max}$. Furthermore as $|C_{\max}| > (1 - \frac{1}{\chi})n$, there are at least $(1 - \frac{1}{\chi})n$ edges incident on $C_{\max}$.

Now the vertices outside $C_{\max}$ are colored with $(\chi - 1)$ colors. Thus there must exists a C_i such that at least $\frac{(1-\frac{1}{\chi})n}{\chi-1} = n/\chi$ edges incident on $C_{\max}$ are also incident on C_i. Now the adversary answers all the vertices in that C_i to be present. Then one must check at least n/χ vertices from $C_{\max}$ because as soon as any one of them is present we have an edge in the graph. ◀

5 Results on restricted graph classes

5.1 Triangle-freeness in planar graphs

A graph G is called *inherently sparse* if every subgraph of G on k nodes contains $O(k)$ edges.

▶ **Theorem 22.** *Let $\mathcal{G}_s$ be a family of inherently sparse graphs on n vertices and $\mathcal{T}$ denote the property of being triangle-free. Then,*

$$\mathcal{G}_s\text{-}min\text{-}cost(\mathcal{T}) = \Omega(\sqrt{n}).$$

The proof of Theorem 22 is deferred to the Appendix of the full version of this paper. As a consequence we obtain the same for the class of planar graphs.

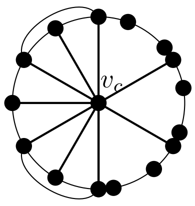


Figure 2 A wheel with d_{max} spokes.

5.2 Acyclicity in planar graphs

▶ **Theorem 23.** *Let $\mathcal{G}_{\mathcal{P}_3}$ be a family of 3-connected planar graphs and $\mathcal{C}$ denote the property of being acyclic. Then,*

$$\mathcal{G}_{\mathcal{P}_3} - min\text{-}cost(\mathcal{C}) = \Omega(\sqrt{n}).$$

Proof. Let $G \in \mathcal{G}_{\mathcal{P}_3}$ be a graph on n vertices and m edges such that every vertex is relevant for the acyclicity property. Let d_{max} denote the maximum degree of G.

Case 1: $d_{\max} > \sqrt{n}$: We use the following fact: In 3-connected planar graphs, removing any vertex leaves a facial cycle around it. We apply this for the maximum degree vertex. In other words, we have a (not necessarily induced) wheel with d_{max} spokes (some spokes might be missing). See Figure 2. The adversary answers the central vertex of the wheel to be present. We can find a matching of size $\Omega(n)$ among the vertices of the cycle. Hence we have $\Omega(n)$ sensitive blocks of length 2 each, which can not be left un-queried.

Case 2: $d_{\max} \leq \sqrt{n}$: We use the fact that every 3-connected graph must have at least $3n/2$ edges. Now using Lemma 14 we obtain a lower bound of $(m-n)/d_{max} \geq \Omega(\sqrt{n})$. ◀

We can generalize the above proof to any planar graph (See the Appendix of the full version of this paper).

6 Conclusion & open directions

- **Weak-evasiveness in the presence of symmetry:** Is it true that every hereditary graph property $\mathcal{P}$ in the node-query setting is weakly-evasive under symmetry, i.e., $\mathcal{G}_{\mathcal{T}}$-*min-cost*$(\mathcal{P}) = \Omega(n)$? What about the randomized case?
- **Polynomial lower bound in the absence of symmetry:** How low can *min-cost*$(\mathcal{P})$ go for a hereditary $\mathcal{P}$ in the absence of symmetry? Is it possible to improve the $\Omega(\log n/\log\log n)$ bound substantially?
- **Further restrictions on graphs:** How low can $\mathcal{G}$-*min-cost*$(\mathcal{P})$ go for hereditary properties $\mathcal{P}$ on restricted classes of graphs $\mathcal{G}$ such as social-network graphs, planar graphs, bipartite graphs, bounded degree graphs etc?
- **Tight bounds on *min-cost*:** What are the tight bounds for natural properties such as acyclicity, planarity, containing a cycle of length t, path of length t?
- **Extension to hypergraphs:** What happens for hereditary properties of (say) 3-uniform hypergraphs in node-query setting? We note that *min-cost*$(\mathcal{I}) = \Theta(n^{1/3})$ for 3-uniform hypergraphs. What about other properties?

- **Global vs local:** We note (See the Appendix of the full version of this paper) that global connectivity requires $\Theta(n)$ queries whereas the cost of s-t connectivity for fixed s and t can go as low as $O(1)$. What about other properties such as min-cut?
- **How about *max-cost* upper bounds? :** From algorithmic point of view, it might be interesting to obtain good upper bounds on the *max-cost*$(\mathcal{P})$ for some natural properties. It might also be interesting to investigate $\mathcal{G}$-*max-cost*$(\mathcal{P})$ for several restricted graph classes such as social-network graphs, planar graphs, bipartite graphs etc.

References

1 *Wikipedia - Abel-Ruffini Theorem.*

2 Laszlo Babai, Anandam Banerjee, Raghav Kulkarni, and Vipul Naik. Evasiveness and the distribution of prime numbers. *arXiv preprint arXiv:1001.4829*, 2010.

3 Nikhil Balaji, Samir Datta, Raghav Kulkarni, and Supartha Podder. Graph properties in node-query setting: effect of breaking symmetry. *CoRR*, abs/1510.08267, 2015. URL: `http://arxiv.org/abs/1510.08267`.

4 Abhishek Bhrushundi, Sourav Chakraborty, and Raghav Kulkarni. Property testing bounds for linear and quadratic functions via parity decision trees. In *Computer Science-Theory and Applications*, pages 97–110. Springer, 2014.

5 Timothy Black. Monotone properties of k-uniform hypergraphs are weakly evasive. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 383–391. ACM, 2015.

6 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

7 Amit Chakrabarti, Subhash Khot, and Yaoyun Shi. Evasiveness of subgraph containment and related properties. *SIAM Journal on Computing*, 31(3):866–875, 2001.

8 Sourav Chakraborty. On the sensitivity of cyclically-invariant boolean functions. In *Computational Complexity, 2005. Proceedings. Twentieth Annual IEEE Conference on*, pages 163–167. IEEE, 2005.

9 Paul Erdös and Richard Rado. Intersection theorems for systems of sets. *Journal of the London Mathematical Society*, 1(1):85–90, 1960.

10 Ehud Friedgut, Jeff Kahn, and Avi Wigderson. Computing graph properties by randomized subcube partitions. In *Randomization and approximation techniques in computer science*, pages 105–113. Springer, 2002.

11 Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

12 Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the sensitivity conjecture. *Theory of Computing, Graduate Surveys*, 4:1–27, 2011.

13 Russell Impagliazzo and Moni Naor. Decision trees and downward closures. In *Structure in Complexity Theory Conference*, pages 29–38, 1988.

14 Gabör Ivanyos. *Personal communication.*

15 Jeff Kahn, Michael Saks, and Dean Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984.

16 Raghav Kulkarni. Evasiveness through a circuit lens. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 139–144. ACM, 2013.

17 Raghav Kulkarni. Gems in decision tree complexity revisited. *ACM SIGACT News*, 44(3):42–55, 2013.

18 Raghav Kulkarni, Youming Qiao, and Xiaoming Sun. Any monotone property of 3-uniform hypergraphs is weakly evasive. *Theoretical Computer Science*,

588:16 – 23, 2015. URL: http://www.sciencedirect.com/science/article/pii/S030439751400855X, doi:http://dx.doi.org/10.1016/j.tcs.2014.11.012.

19 Raghav Kulkarni and Miklos Santha. Query complexity of matroids. In *Algorithms and Complexity*, pages 300–311. Springer, 2013.

20 Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. In *Proceedings of the Twenty-third Annual ACM Symposium on Theory of Computing*, STOC '91, pages 455–464, New York, NY, USA, 1991. ACM. URL: http://doi.acm.org/10.1145/103418.103466, doi:10.1145/103418.103466.

21 Frank H. Lutz. Some results related to the evasiveness conjecture. *Journal of Combinatorial Theory, Series B*, 81(1):110 – 124, 2001. URL: http://www.sciencedirect.com/science/article/pii/S0095895600920008, doi:http://dx.doi.org/10.1006/jctb.2000.2000.

22 Ryan O'Donnell. *Analysis of boolean functions.* Cambridge University Press, 2014.

23 Ronald L Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3(3):371–384, 1976.

24 Xiaoming Sun, Andrew C Yao, and Shengyu Zhang. Graph properties and circular functions: How low can quantum query complexity go? In *Computational Complexity, 2004. Proceedings. 19th IEEE Annual Conference on*, pages 286–293. IEEE, 2004.

25 Zhiqiang Zhang and Yaoyun Shi. On the parity complexity measures of boolean functions. *Theoretical Computer Science*, 411(26):2612–2618, 2010.

Stable States of Perturbed Markov Chains*

Volker Betz[1] and Stéphane Le Roux[2]

1 **Fachbereich Mathematik, TU Darmstadt, Deutschland**
`betz@mathematik.tu-darmstadt.de`

2 **Département d'Informatique, Université Libre de Bruxelles, Belgique**
`stephane.le.roux@ulb.ac.be`

Abstract

Given an infinitesimal perturbation of a discrete-time finite Markov chain, we seek the states that are stable despite the perturbation, *i.e.* the states whose weights in the stationary distributions can be bounded away from 0 as the noise fades away. Chemists, economists, and computer scientists have been studying irreducible perturbations built with monomial maps. Under these assumptions, Young proved the existence of and computed the stable states in cubic time. We fully drop these assumptions, generalize Young's technique, and show that stability is decidable as long as $f \in O(g)$ is. Furthermore, if the perturbation maps (and their multiplications) satisfy $f \in O(g)$ or $g \in O(f)$, we prove the existence of and compute the stable states and the metastable dynamics at all time scales where some states vanish. Conversely, if the big-O assumption does not hold, we build a perturbation with these maps and no stable state. Our algorithm also runs in cubic time despite the weak assumptions and the additional work. Proving its correctness relies on new or rephrased results in Markov chain theory, and on algebraic abstractions thereof.

1998 ACM Subject Classification G.3 Probability and Statistics; G.2.2 Graph Theory

Keywords and phrases evolution, metastability, tropical, shortest path, SCC, cubic time

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.18

1 Introduction

Motivated by the dynamics of chemical reactions, Eyring [4] and Kramers [12] studied how infinitesimal perturbations of a Markov chain affect its stationary distributions. It has been further investigated by, *e.g.*, probability theorists, economists, and computer scientists. In fields of application such as learning and game theory, it is sometimes unnecessary to describe the exact values of the limit stationary distributions: it suffices to know whether these values are zero or not. Thus, the *stochastically stable states* ([5], [10], [17]) were defined in several contexts as the states with positive probability in the limit. We rephrase a definition below.

▶ **Definition 1** (Markov chain and stability)**.** A finite discrete-time Markov chain is a function $m : S \times S \to [0,1]$ such that $\sum_{y \in S} m(x,y) = 1$ for all x in finite state space S. A stationary distribution is a probability distribution over the states that is invariant under one step of the MC. Let I be a subset of positive real numbers with 0 as a limit point for the usual topology[1]. A perturbation is a function $p : I \times S \times S \to [0,1]$ such that p_ϵ is a discrete-time MC for all $\epsilon \in I$. If p_ϵ is irreducible for all $\epsilon \in I$, then p is said to be an irreducible perturbation.

A state $x \in S$ is stochastically stable if there is a family of corresponding stationary distributions $(\mu_\epsilon)_{\epsilon \in I}$ s.t. $\liminf_{\epsilon \to 0} \mu_\epsilon(x) > 0$, *i.e.* $1 \in O(\mu(x))$. It is stochastically fully vanishing if $\limsup_{\epsilon \to 0} \mu_\epsilon(x) = 0$ for all $(\mu_\epsilon)_{\epsilon \in I}$. Non-stable states are called vanishing.

* The second author was partly supported by the ERC inVEST (279499) project.
[1] This implies that I is infinite. $]0,1]$ and $\{\frac{1}{2^n} \mid n \in \mathbb{N}\}$ are typical I.

Let us motivate Definition 1: we want to find out which states of a real-world system are the most likely to occur a lot in the long run. The system behaves like a MC that we are unable to know exactly, but we know what it is likely to look like, *i.e.* we know a family of MC indexed by ϵ where it occurs for sure, most likely for a very small ϵ. This alone is far too weak to decide state likeliness in the long run, but assuming a good asymptotic behavior of the family for small ϵ implies the existence of a likely state. This is our main result below.

▶ **Theorem 2.** *Consider a perturbation with state space S such that $f \in O(g)$ or $g \in O(f)$ for all f and g in the multiplicative closure of the transition probability functions $\epsilon \mapsto p_\epsilon(x,y)$ with $x \neq y$. Then the perturbation has stable states. Furthermore, any oracle deciding $f \in O(g)$ in constant time allows us to decide stability in $O(|S|^3)$.*

The finiteness of the state space implies that all perturbations have non-fully vanishing states.

1.1 Related works and comparisons

In 1990 Foster and Young [5] defined the stochastically stable states of a general (continuous) evolutionary process, as an alternative to the evolutionary stable strategies [16]. Stochastically stable states were soon adapted in [10] for 2×2 evolutionary games. Then Young [17, Theorem 4] proved a "finite version of results obtained by Freidlin and Wentzel" in [6] and characterized the stochastically stable states if the perturbation satisfies the following assumptions: 1) the p_ϵ are aperiodic and irreducible; 2) the p_ϵ converge to some p_0 when ϵ approaches zero; 3) every transition probability is a function of ϵ that is equivalent to $c \cdot \epsilon^\alpha$ for some non-negative real numbers c and α. The main tool in Young's proof was proved in [11] and is the special case for irreducible chains of the Markov chain tree theorem (see [13] or [6]). Young's characterization involves minimum directed spanning trees, which can be computed in $O(n^2)$ [7] for graphs with n vertices. Since there are at most n roots for directed spanning trees in a graph with n vertices, Young can compute the stable states in $O(n^3)$.

In 2000 Ellison [3] studied the stable states *via* the alternative notion of the radius of a basin of attraction, and wrote that the major drawback of his work compared to Young's is that it is "not universally applicable". In 2005, Greenwald and Wicks [9] designed an algorithm expressing the exact values of the limit stationary distribution of a perturbation, which, as a byproduct, also computes the stable states. Like [17] they consider perturbations that are related to the functions $\epsilon \mapsto \epsilon^\alpha$, but they only require that the functions converge exponentially fast. Also, instead of requiring that the P^ϵ be irreducible for $\epsilon > 0$, they only require that they have exactly one essential class. They do not analyze the complexity of their algorithm, though. We improve upon [17], [3], and [9] in several ways.

1. The perturbations in the literature relate to the maps $\epsilon \mapsto \epsilon^\alpha$. Their specific form and their continuity, especially at 0, are used in the existing proofs. Theorem 2 relaxes this assumption. Continuity, even at 0, is irrelevant, which allows for aggressive, *i.e.*, non-continuous "perturbations". We show that our assumption is (almost) unavoidable.
2. The perturbations in the literature are irreducible (or almost in [9]). It is general enough for perturbations using the maps $\epsilon \mapsto \epsilon^\alpha$, since it suffices to process each sink (aka bottom) irreducible component independently, and gather the results. This trick does not work for general perturbation maps, but Theorem 2 still does not assume irreducibility.
3. The perturbation is abstracted into a weighted graph and shrunk by combining recursively a shortest-path algorithm (w.r.t. some tropical-like semiring) and a strongly-connected-component algorithm. Using tropical-like algebra to abstract over Markov chains has already been done before, *e.g.* in [8], but not to solve the stable state problem.

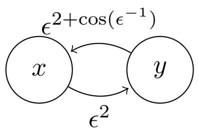


Figure 1

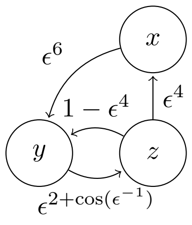


Figure 2

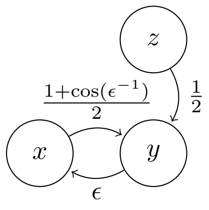


Figure 3

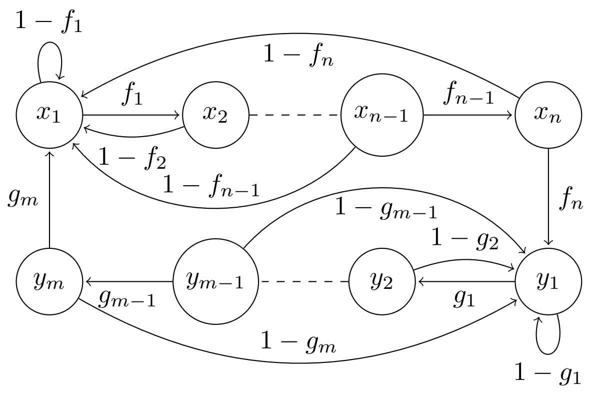


Figure 4

4. We compute the stable states in $O(n^3)$, the best known complexity as in [17], and the computation itself is a summary of the asymptotic behavior of the perturbation: it says at which time scales the vanishing states vanish, and the intermediate graph at each recursive stage of the algorithm accounts for the metastable dynamics at this time scale.

Section 1.2 analyses which assumptions are relevant for the existence of stable states; Section 2 proves the existential part of Theorem 2, *i.e.* it develops the probabilistic machinery to prove the existence of stable states; hinging on this, Section 3 proves the algorithmic part of Theorem 2, *i.e.* it abstracts the relevant objects using a new algebraic structure, presents the algorithm, and proves its correctness and complexity; Section 4 discusses two important special cases and an induction proof principle related to the termination of our algorithm. (Standard notations and proofs can be found in [1].)

1.2 Towards general assumptions

Even continuous perturbations that converge when ϵ approaches 0 may fail to have stable states. For instance let $S := \{x, y\}$ and for all $\epsilon \in]0, 1]$ let $p_\epsilon(x, y) := \epsilon^2$ and $p_\epsilon(y, x) := \epsilon^{2+\cos(\epsilon^{-1})}$ as in Figure 1, where the self-loops are omitted. In the unique stationary distribution x has a weight $\mu_\epsilon(x) = (1 + \epsilon^{-\cos(\epsilon^{-1})})^{-1}$. Since $\mu_{(2n\pi)^{-1}}(x) = \frac{2n\pi}{1+2n\pi} \rightarrow_{n\to\infty} 1$ and $\mu_{(2(n+1)\pi)^{-1}}(x) = \frac{1}{1+2(n+1)\pi} \rightarrow_{n\to\infty} 0$, neither x nor y is stable.

As mentioned above, the usual perturbations relate to the maps $\epsilon \mapsto \epsilon^\alpha$ with $\alpha \geq 0$, which rules out Figure 1 and implies the existence of stable states [17]. Here, however, we want to assume as little as possible about the perturbations, while still guaranteeing the existence of stable states. Towards it let us rephrase the big O notation as a binary relation. (Its useful and well-known algebraic properties are mentioned in [1].)

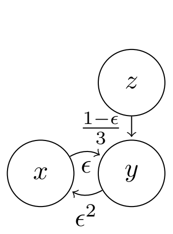
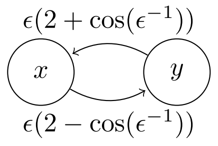
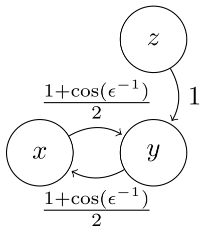


Figure 5



$\epsilon(2+\cos(\epsilon^{-1}))$
x
y
$\epsilon(2-\cos(\epsilon^{-1}))$

Figure 6



z
$\frac{1+\cos(\epsilon^{-1})}{2}$
1
x
y
$\frac{1+\cos(\epsilon^{-1})}{2}$

Figure 7

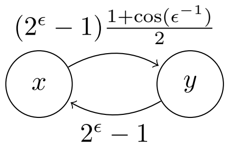


Figure 8

▶ **Definition 3** (Order). For $f, g : I \to [0,1]$, let us write $f \precsim g$ if there exist positive b and ϵ such that $f(\epsilon') \leq b \cdot g(\epsilon')$ for all $\epsilon' < \epsilon$; let $f \cong g$ stand for $f \precsim g \wedge g \precsim f$.

Requiring that every two transition probability maps f and g occurring in the perturbation satisfy $f \precsim g$ or $g \precsim f$ rules out the example from Figure 1, but not the one from Figure 2. There $\mu_\epsilon(z) \leq \mu_\epsilon(x) = \frac{\epsilon^{\cos(\epsilon^{-1})}}{1+\epsilon^{\cos(\epsilon^{-1})}(1+\epsilon^2)}$ and $\mu_\epsilon(y) = \frac{1}{1+\epsilon^{\cos(\epsilon^{-1})}(1+\epsilon^2)}$. So $\mu_\epsilon(z) \to_{\epsilon\to 0} 0$ and $\mu_{2n\pi}(y) \to_{n\to\infty} 0$ and $\mu_{2(n+1)\pi}(x) \to_{n\to\infty} 0$, no state is stable. Informally, z is not stable because it gives everything but receives at most ϵ; neither x nor y is stable since their interaction resembles Figure 1 due to ϵ^6 and $\epsilon^4 \cdot \epsilon^{2+\cos(\epsilon^{-1})}$. This remark is turned into a general Observation 4 below, which will motivate the "unavoidable" Assumption 5.

▶ **Observation 4.** *For $1 \leq i \leq n$ and $1 \leq j \leq m$ let $f_i, g_j : I \to [0,1]$ be s.t. $\prod_i f_i$ and $\prod_j g_j$ are not $\precsim$-comparable. Then there is a perturbation without stable states that is built only with the $f_1, \dots, f_n, g_1, \dots, g_m$ and the $1-f_1, \dots, 1-f_n, 1-g_1, \dots, 1-g_m$. See Figure 4.*

▶ **Assumption 5.** *The multiplicative closure of the maps $\epsilon \mapsto p_\epsilon(x,y)$ with $x \neq y$ is totally preordered by $\precsim$.*

E.g, the maps $\epsilon \mapsto c \cdot \epsilon^\alpha$ with $c > 0$ and $\alpha \in \mathbb{R}$ satisfy Assumption 5. We can afford such a weak Assumption 5 because we are not interested in the exact weights of some putative limit stationary distribution, but only whether the weights are bounded away from zero.

Let us show the significance of Assumption 5, which is satisfied in Figures 5 to 8: Young's result shows that y is the unique stable state of the perturbation in Figure 5, but it cannot say anything about Figures 6 to 8: Figure 6 is not regular, *i.e.*, $\frac{2+\cos(\epsilon^{-1})}{2-\cos(\epsilon^{-1})}$ does not converge, and neither do the weights $\mu_\epsilon(x)$ and $\mu_\epsilon(y)$, but it is possible to show that both limits inferior are $1/4$ nonetheless, so both x and y are stable; the transition probabilities in Figure 7 do not converge, and $\frac{1+\cos(\epsilon^{-1})}{2}$ and $1 - \frac{1+\cos(\epsilon^{-1})}{2}$ are not even comparable, but it is easy to see that $\mu_\epsilon(x) = \mu_\epsilon(y) = \frac{1}{2}$; and in Figure 8 the only stable state is x since its weight oscillates between $\frac{1}{2}$ and 1. Note that Assumption 5 rules out Figure 1 to 4 without stable states.

2 Existence of stable states

This section presents three transformations that simplify perturbations while retaining the relevant information about stability. Two of them are defined *via* the dynamics of the original perturbation. Their relevance relies on the close relation between the stationary distributions and the dynamics of MCs. Lemma 6 below pinpoints this relation, where $\mathbb{P}^x(\tau_y^+ < \tau_x^+)$ is the probability that starting from x the MC hits y before returning to x.

▶ **Lemma 6.** *A distribution μ of a finite Markov chain is stationary iff its support involves only essential states and for all states x and y we have $\mu(x)\mathbb{P}^x(\tau_y^+ < \tau_x^+) = \mu(y)\mathbb{P}^y(\tau_x^+ < \tau_y^+)$.*

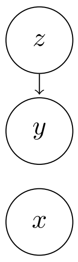


Figure 9

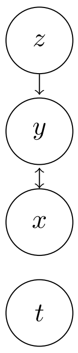


Figure 10

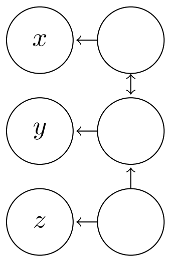


Figure 11

Lemma 6 can already help us find the stable states of small examples such as in Figures 1 and 6. In Figure 1 it says that $\mu_\epsilon(x)\epsilon^2 = \mu_\epsilon(y)\epsilon^{2+\cos(\epsilon^{-1})}$ so we find $\liminf \mu_\epsilon(x) = \liminf \mu_\epsilon(y) = 0$ without calculating the stationary distributions. In Figure 6 it says that $\mu_\epsilon(x)(2-\cos(\epsilon^{-1})) = \mu_\epsilon(y)(2+\cos(\epsilon^{-1}))$, so $\mu_\epsilon(x) \leq 3\mu_\epsilon(y)$ and $\frac{1}{4} \leq \mu_\epsilon(y)$, and likewise for x.

The dynamics, *i.e.*, terms like $\mathbb{P}^x(\tau_y^+ < \tau_x^+)$ are usually hard to compute, and so will be the two transformations that are defined *via* the dynamics, but Lemma 7 below shows that approximating them is safe as far as the stable states are concerned. *E.g.*, the coefficients in Figure 6 (19) can safely be replaced with ϵ (1), and Figure 13 with Figure 14. Lemma 7, where p, μ, *etc.* depend on ϵ, will simplify the computation of the stable states dramatically.

▶ **Lemma 7.** *Let p and $\tilde{p}$ be perturbations with the same state space, s.t. $x \neq y \Rightarrow p(x,y) \cong \tilde{p}(x,y)$. For all stationary distribution maps μ for p, there exists $\tilde{\mu}$ for $\tilde{p}$ such that $\mu \cong \tilde{\mu}$.*

2.1 Essential graph

The *essential graph* of a perturbation captures the non-infinitesimal flow between different states at the normal time scale. It is a very coarse yet useful description of the perturbation.

▶ **Definition 8** (Essential graph)**.** Given a perturbation with state space S, the essential graph is a binary relation over S and possesses the arc (x,y) if $x \neq y$ and $p(z,t) \precsim p(x,y)$ for all $z, t \in S$. The essential classes are the sink SCCs of the graph. The other SCCs are the transient classes. A state in an essential class is essential, the others are transient.

The essential classes will be named $E_1, \dots, E_k$. Observation 9 below implies that the essential graph is made of the arcs (x,y) such that $x \neq y$ and $p(x,y) \cong 1$, as expected.

▶ **Observation 9.** *Let p be a perturbation. There exist positive c and ϵ_0 such that for all $\epsilon < \epsilon_0$, for all simple paths γ in the essential graph, $c < p_\epsilon(\gamma)$.*

For example, the perturbations (with $I =]0,1]$) that are described in Figures 2, 3, 5, and 7 all have Figure 9 as essential graph, and $\{x\}$ and $\{y\}$ as essential class. Figure 10 (11) is the essential graph of Figure 12 (15), and $\{x,y\}$ and $\{t\}$ (x, y, z) are its essential classes. Note that the essential states of a perturbation and of the related MCs are not the same: in Figure 12, for all $\epsilon \in]0,1]$ all the states are essential for the related MCs. However, if $p_\epsilon = m$ for some m and all ϵ, the essential graph of p is the underlying graph of m. Thus the essential graphs generalize the underlying graphs like the perturbations generalize the MCs.

The essential graph alone cannot tell which states are stable: *e.g.*, swapping ϵ and ϵ^2 in Figure 5 yields the same graph, but then by Lemma 6 the only stable state is x instead of y.

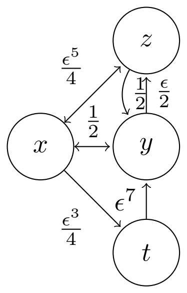


Figure 12

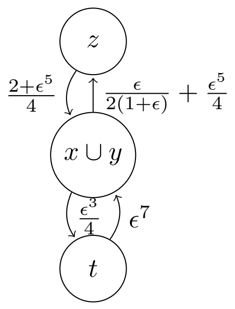


Figure 13

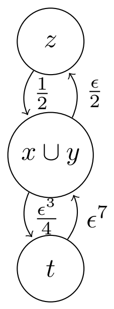


Figure 14

Yet, the graph makes the following case disjunction possible, along which we will either say that all states are stable, or perform one of the transformations from the next subsections.

1. Either the graph is empty (*i.e.* totally disconnected) and the perturbation is trivial, or
2. it is empty and the perturbation is non-trivial, or
3. it is non-empty and has a non-singleton essential class, or
4. it is non-empty and has only singleton essential classes.

Observation 9 motivates the convenient Assumption 10 below. Note that it is just made wlog, *i.e.*, up to focusing on a smaller neighborhood of 0 inside I, whereas Assumption 5 above is a key condition that will appear explicitly in our final result.

▶ **Assumption 10.** *There is $c > 0$ s.t. $p(\gamma) > c$ for all simple paths γ in the essential graph.*

2.2 Essential collapse

The essential collapse, defined below, amounts to merging one essential class of a perturbation into one meta-state and letting this state represent faithfully the whole class in terms of dynamics between the whole class and each of the outside states. ($\mathbb{P}^x(X_{\tau^+_{S \setminus E \cup \{x\}}} = y)$ is the probability that starting from x, the first state hit in $S \setminus E \cup \{x\}$ is y.)

▶ **Definition 11** (Essential collapse of a perturbation). Let p be a perturbation on state space S. Let x be a state in an essential class E, and let $\tilde{S} := (S \setminus E) \cup \{\cup E\}$. The essential collapse $\kappa(p,x) : I \times \tilde{S} \times \tilde{S} \to [0,1]$ of p around x is defined below.

$$\begin{aligned}
\kappa(p,x)(y,z) &:= p(y,z) && \text{for all } y,z \in S \setminus E\\
\kappa(p,x)(\cup E, \cup E) &:= \mathbb{P}^x(X_{\tau^+_{S\setminus E \cup \{x\}}} = x) &&\\
\kappa(p,x)(\cup E, y) &:= \mathbb{P}^x(X_{\tau^+_{S\setminus E \cup \{x\}}} = y) && \text{for all } y \in S \setminus E\\
\kappa(p,x)(y, \cup E) &:= \sum_{z \in E} p(y,z) && \text{for all } y \in S \setminus E
\end{aligned}$$

▶ **Observation 12.** *$\kappa(p,x)$ is again a perturbation, κ preserves irreducibility, and if $\{x\}$ is an essential class, $\kappa(p,x) = p$.*

For example, collapsing around x or y in Figure 6 has no effect. Figure 12 has essential classes $\{x,y\}$ and $\{t\}$. Figure 13 displays its essential collapse around x. It was calculated by noticing that $\mathbb{P}^x(X_{\tau^+_{\{x,z,t\}}} = t) = \frac{\epsilon^3}{4}$, and $\mathbb{P}^x(X_{\tau^+_{\{x,z,t\}}} = x) = \frac{1}{2} - \frac{\epsilon^3}{4} - \frac{\epsilon^5}{4} + \frac{1}{2} \cdot \mathbb{P}^y(X_{\tau^+_{\{x,z,t\}}} = x)$, and $\mathbb{P}^y(X_{\tau^+_{\{x,z,t\}}} = x) = \frac{1}{2} + \frac{1-\epsilon}{2} \cdot \mathbb{P}^y(X_{\tau^+_{\{x,z,t\}}} = x)$.

Proposition 16 will show that it suffices to compute the stable states of Figure 13 to compute those of Figure 12, and by Lemma 7 it suffices to compute those of the simpler

Figure 14. However, computing the exact values $\mathbb{P}^x(X_{\tau^+_{S\setminus E\cup\{x\}}} = y)$ can be difficult even on simple examples like above. Fortunately, Lemma 13 shows that they are $\cong$-equivalent to maxima that are easy to compute. *E.g.*, using Lemma 13 to approximate the essential collapse of Figure 12 around x yields Figure 14 directly, without Figure 13 as an intermediate.

▶ **Lemma 13.** *Let p be a perturbation with state space S satisfy Assumption 5, and let $\tilde{p}$ be the essential collapse $\kappa(p,x)$ of p around x in some essential class E. For all $y \in S \setminus E$, we have $\tilde{p}(\cup E, y) \cong \max_{z\in E} p(z,y)$ and $\tilde{p}(y, \cup E) \cong \max_{z\in E} p(y,z)$.*

Note that by Lemma 13, only the essential class is relevant during the essential collapse up to $\cong$, the exact state is irrelevant. Lemma 13 is also a tool used to prove, *e.g.*, Proposition 14 which shows that the essential graph may contain useful information about stability.

▶ **Proposition 14.** *Let a perturbation p with state space S satisfy Assumption 5, let μ be a corresponding stationary distribution map.*
1. *If y is a transient state, $\liminf_{\epsilon\to 0} \mu_\epsilon(y) = 0$.*
2. *If two states x and y belong to the same essential or transient class, $\mu(x) \cong \mu(y)$.*

Proposition 14.1 says that the transient states are vanishing, *e.g.* the nameless states in Figure 11. Proposition 14.2 says that two states in the same class are either both stable or both vanishing, *e.g.* $\{x\}$ and $\{y\}$ in Figure 10.

The essential collapse is useful since it preserves (non-)stability, as stated in Proposition 16. Its proof invokes Lemma 15 below, which shows that the essential collapse preserves the dynamics up to $\cong$, and Lemma 6, which relates the dynamics and the stationary distributions.

▶ **Lemma 15.** *Given a perturbation p with state space S, let $\tilde{p}$ be the essential collapse of p around x in some essential class E, and let $\tilde{x} := \cup E$. The following holds for all $y \in S \setminus E$.*
$\mathbb{P}^y(\tau_x < \tau_y) \cong \tilde{\mathbb{P}}^y(\tau_{\tilde{x}} < \tau_y) \quad \wedge \quad \mathbb{P}^x(\tau_y < \tau_x) \cong \tilde{\mathbb{P}}^{\tilde{x}}(\tau_y < \tau_{\tilde{x}})$

▶ **Proposition 16.** *Let a perturbation p with state space S satisfy Assumption 5, and let x be in an essential class E.*
1. *Let $\tilde{p}$ be the chain after the essential collapse of p around x. Let μ ($\tilde{\mu}$) be a stationary distribution map of p ($\tilde{p}$). There exists a stationary distribution map $\tilde{\mu}$ for $\tilde{p}$ (μ for p) such that $\tilde{\mu}(\cup E) \cong \mu(x)$ and $\tilde{\mu}(y) \cong \mu(y)$ for all $y \in S \setminus E$.*
2. *A state $y \in S$ is stable for p iff either $y \in E$ and $\cup E$ is stable for $\kappa(p,x)$, or $y \notin E$ and y is stable for $\kappa(p,x)$.*

By definition, collapsing an essential class preserves the structure of the perturbation outside of the class, so Proposition 16 implies that the collapse commutes up to $\cong$. Especially, the order in which the collapses are performed is irrelevant when computing the stable states.

2.3 Transient deletion

If all the essential classes of a perturbation are singletons, Observation 12 says that the essential collapse is useless. If in addition the essential graph has arcs, there are transient states, and Definition 17 below deletes them to shrink the perturbation further.

▶ **Definition 17** (Transient deletion). Let a perturbation p with state space S, transient states T, and singleton essential classes, satisfy Assumption 5. The function $\delta(p)$ over $S\setminus T$ is derived from p by transient deletion: for all distinct $x, y \in S \setminus T$ let $\delta(p)(x,y) := \mathbb{P}^x(X_{\tau^+_{S\setminus T}} = y)$

▶ **Observation 18.** *$\delta(p)$ is again a perturbation, δ preserves irreducibility, and if all states are essential, $\delta(p) = p$.*

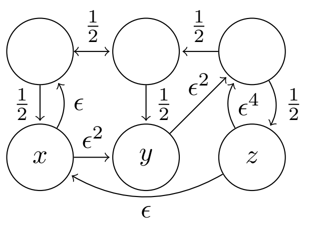


Figure 15

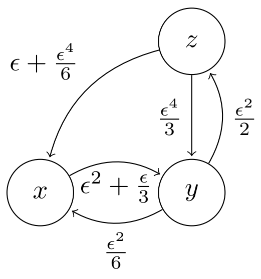


Figure 16

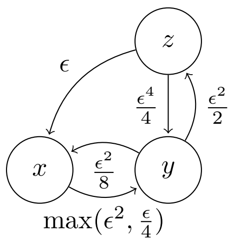


Figure 17

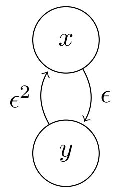


Figure 18

For example, in Figure 5 the essential classes are $\{x\}$ and $\{y\}$, z is transient, and the transient deletion yields Figure 18. Also, in Figure 15, the essential classes are $\{x\}$, $\{y\}$, and $\{z\}$, the transient states are nameless, and the transient deletion yields Figure 16. The transient deletion is useful thanks to Proposition 19 below.

▶ **Proposition 19.** *If a perturbation p satisfy Assumption 5 and has singleton essential classes, p and $\delta(p)$ have the same stable states.*

Like the essential collapse, the transient deletion is defined *via* the dynamics and is hard to compute. Like Lemma 13 did for the collapse, Lemma 20 approximates the transient deletion by an expression that is easy to compute. *E.g.*, Figure 15 yields Figure 17 without computing Figure 16. Note that $\max(\epsilon^2, \frac{\epsilon}{4})$ in Figure 17 may be simplified into ϵ by Lemma 7.

▶ **Lemma 20.** *If p with states S, transient states T, satisfies Assumption 5 and has singleton essential classes, then $\mathbb{P}^x(X_{\tau^+_{S\setminus T}} = y) \cong \max\{p(\gamma) : \gamma \in \Gamma_T(x, y)\}$ for all $x, y \in S \setminus T$.*

2.4 Outgoing scaling and existence of stable states

If the essential graph has no arc, the collapse and the deletion are useless to compute the stable states. This section says how to transform a non-trivial perturbation with empty (*i.e.* totally disconnected) essential graph into a perturbation with the same stable states but a non-empty essential graph, so that collapse or deletion may be applied. Intuitively, it is done by speeding up time until a first non-infinitesimal flow is observable between different states.

Towards it, the *ordered division* is defined in Definition 21. It allows us to divide a function by a function with zeros by returning a default value in the zero case. It is named ordered because we will "divide" f by g only if $f \precsim g$, so that only 0 may be "divided" by 0. Then Observation 22 further justifies the terminology.

▶ **Definition 21** (Ordered division). For $f, g : I \to [0, 1]$ and $n > 1$ let us define $(f \div_n g) : I \to [0, 1]$ by $(f \div_n g)(x) := \frac{f(x)}{g(x)}$ if $0 < g(x)$ and otherwise $(f \div_n g)(x) := \frac{1}{n}$.

▶ **Observation 22.** *$(f \div_n g) \cdot g = f$ for all n and $f, g : I \to [0, 1]$ such that $f \precsim g$.*

▶ **Definition 23** (Outgoing scaling). Let a perturbation p with state space S satisfy Assumption 5, let $m := |S| \cdot \max\{p(z, t) \mid z, t \in S \wedge z \neq t\}$, and let us define the following.

- $\sigma(p)(x, y) := p(x, y) \div_{|S|} m$ for all $x \neq y$
- $\sigma(p)(x, x) := (p(x, x) + m - 1) \div_{|S|} m$.

For example Figure 6 satisfies Assumption 5 and its essential graph is empty. Scaling it yields Figure 19, which satisfies Assumption 5 and whose essential graph has two arcs. Note that collapsing around x or y in Figure 6 has no effect, but in Figure 19 it yields a one-state perturbation. Proposition 24 below states how well the outgoing scaling behaves.

▶ **Proposition 24.** *1. If a perturbation p satisfies Assumption 5, so does $\sigma(p)$, and the essential graph of $\sigma(p)$ is non-empty.*
2. A state is stable for p iff it is stable for $\sigma(p)$.

The outgoing scaling divides the weights of the proper arcs by m, as if time were sped up by m^{-1}. The self-loops thus lose their meaning, but Proposition 24 proves it harmless. Note that the self-loops are also ignored in Assumption 5, Lemma 7, and Definition 8.

Let us describe a recursive computation of the stable states: if the perturbation is constant identity, all its states are stable; else, if the essential graph is empty, apply the outgoing scaling; else, apply one collapse or the transient deletion. This procedure is correct by Propositions 24.2, 16.2, and 19, hence Theorem 25 below (the existential part of Theorem 2).

▶ **Theorem 25.** *Let p be a perturbation such that $f \precsim g$ or $g \precsim f$ for all f and g in the multiplicative closure of the $p(x, y)$ with $x \neq y$. Then p has stable states.*

3 Abstract and quick algorithm

Following the procedure described before Theorem 25 but using the approximation Lemmas 13 and 20 instead of the precise collapse and deletion computes the stable states in $O(n^4)$, where n is the number of states. To improve the speed to $O(n^3)$ we split the deletion into two stages, depending on the lengths of the relevant paths. To improve it further by a multiplicative factor we merge the collapses into these two stages. It would have been cumbersome to define the collapse-deletion merge directly *via* the dynamics in Section 2, and to prove its correctness *via* probabilistic techniques, hence the usefulness of the rather atomic collapse and deletion in the first part of our work. Ensuring that they are safely performed up to $\cong$ is a straightforward sanity check, by Lemma 7, but handling the collapse-deletion merge requires particular attention. Also, the proof for the scaling involves a new algebraic structure accommodating the ordered division: we call it an order-division semiring $(F, 0, 1, \cdot, \leq, \div)$, where F is the quotient of the transition-probability maps by $\cong$, 0 is the zero function, $\div$ is the abstraction of the ordered division, *etc.*, and $[\chi]$ and $[\sigma]$ are the abstractions of the collapse-deletion merge and of the scaling, respectively. All this is well-defined thanks to Assumption 5. Definitions and proofs can be found in Section 3.1 in [1].

Based on these abstractions, this section presents the algorithm (computing the stable states and more), its correctness, and its complexity in $O(n^3)$. Algorithm 1 mainly consists in applying recursively the function $[\chi] \circ [\sigma]$ until a totally disconnected graph is produced. It does not explicitly refer to perturbations since this notion was abstracted on purpose. Instead, the algorithm manipulates digraphs with arcs labeled in an ordered-division semiring, in which inequality, multiplication and ordered division are implicitly assumed to be computable.

One call to the `FindHubRec` corresponds to $[\chi] \circ [\sigma]$, *i.e.* Lines 7 and 9 to $[\sigma]$, and Lines 10 till 18 to $[\chi]$. Before calling `FindHubRec`, Lines 2 and 3 produce an isomorphic copy of the input that is easier to handle when making unions and keeping track of the remaining vertices. Note that Line 9 does not update the $P(x, x)$: it would be useless indeed, since the self-loops are irrelevant by Observation 34 in [1]. Line 10 computes the essential graph up to self-loops, and Line 11 computes the essential classes by a modified Tarjan's algorithm. The computation of $[\chi](P)(\cup E_i, \cup E_j) := \max_{\leq}\{P(\gamma) : \gamma \in \Gamma_T(E_i, E_j)\}$ is performed in two stages: the first stage at Line 12 considers only paths of length one; the second stage at Line 18 considers the paths with a vertex in T. This case disjunction reduces the size of the graph on which the shortest path algorithm from Line 16 is run (and thus the complexity of `FindHub`). Note that it is called with laws $\cdot$ and max instead of $+$ and min. Moreover,

Algorithm 1: FindHub

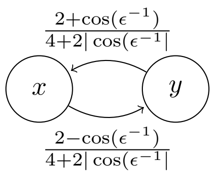
```
Function FindHub is
    input  : $(S, P)$, where $P : S \times S \to F$
    // $(F, 0, 1, \cdot, \leq, \div)$ is an ordered-division semiring.
    output : a subset of $S$
    $\hat{S} \leftarrow \{\{s\} | s \in S\}$;                                   // For bookkeeping.
    for $x, y \in S$ do $\hat{P}(\{x\}, \{y\}) \leftarrow P(x, y)$;             // For bookkeeping.
    return FindHubRec($\hat{S}$,$\hat{P}$);
end
Function FindHubRec is
    input  : $(S, P)$, where $S$ is a set of sets and $P : S \times S \to F$
    output : a subset of $S$
    $M \leftarrow \max\{P(x, y) \mid (x, y) \in S \times S \wedge x \neq y\}$;
    if $M = 0$ then return $\cup S$;                                        // Recursion base case
    for $x, y \in S$ and $x \neq y$ do $P(x, y) \leftarrow P(x, y) \div M$;      // Outgoing scaling.
    $A \leftarrow \{(x, y) \in S \times S \mid P(x, y) = 1\}$;                   // $A$ is a digraph.
    $(E_1, \dots, E_k) \leftarrow$TarjanSinkSCC($S$,$A$);          // Returns the sink SCCs of $A$.
    // Maximal labels of direct arcs, below.
    for $1 \leq i, j \leq k$ do $P'(\cup E_i, \cup E_j) \leftarrow \max\{P(x, y) \mid (x, y) \in E_i \times E_j\}$;
    // Maximal labels of all relevant paths, in the remainder.
    $T \leftarrow S \setminus (E_1 \cup \cdots \cup E_k)$;
    $P_T \leftarrow P$;                                                   // Initialisation.
    for $(x, y) \in (S \setminus T) \times S$ do $P_T(x, y) \leftarrow 0$;  // Drops arcs not starting in $T$.
    for $y \in T$ do $P_T(y, \_) \leftarrow$Dijkstra($S$,$P_T$,$y$, $\cdot$, max);
    // $P_T(y, \_)$ is the "distance" function from $y \in T$, using $\cdot$ and max.
    for $1 \leq i, j \leq k$ and $i \neq j$ and $(x_i, x_j, y) \in E_i \times E_j \times T$ do
        $P'(\cup E_i, \cup E_j) \leftarrow \max(P'(\cup E_i, \cup E_j), P(x_i, y) \cdot P_T(y, x_j))$;
    end for
    return FindHubRec($\{\cup E_1, \dots, \cup E_k\}, P'$)
end
```

since our weights are at most 1 we can use [14] or [2] (which assume non-negative weights) to implement Line 16. Proposition 26 below shows that our algorithm is fast, and our main algorithmic result follows, which is the algorithmic part of Theorem 2.

▶ **Proposition 26.** *The algorithm* FindHub *terminates within $O(n^3)$ computation steps, where n is the number of vertices of the input graph.*

▶ **Theorem 27.** *Let a perturbation p satisfy Assumption 5. A state is stochastically stable iff it belongs to* FindHub$(S, [p])$. *Provided that inequality, multiplication, and ordered division between equivalence classes of perturbation maps can be computed in constant time, stability can be decided in $O(n^3)$, where n is the number of states.*

One achievement of our algorithm is that it processes all weighted digraphs (*i.e.* abstractions of perturbations) uniformly: neither irreducibility nor any kind of connectedness is required. *E.g.* in Figures 20 to 25, the four-state perturbation is the disjoint union of two



$\frac{2+\cos(\epsilon^{-1})}{4+2|\cos(\epsilon^{-1}|}$
x
y
$\frac{2-\cos(\epsilon^{-1})}{4+2|\cos(\epsilon^{-1}|}$

Figure 19

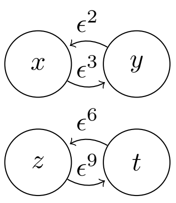


Figure 20 Initial perturbation.

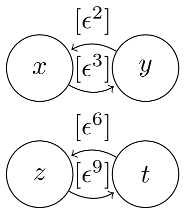


Figure 21 Abstraction.

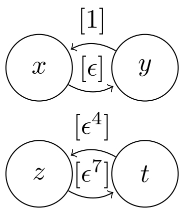


Figure 22 Outgoing scaling.

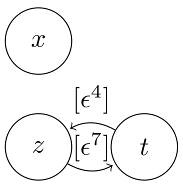


Figure 23 Transient deletion.

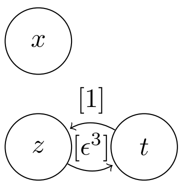


Figure 24 Outgoing scaling.

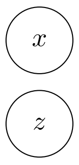


Figure 25 Transient deletion.

smaller perturbations. As expected the stable states of the union are the union of the stable states, *i.e.* $\{x, z\}$, but whereas the outgoing scaling applied to the bottom of Figure 21 (the perturbation restricted to $\{z, t\}$) would yield the bottom of Figure 24 directly by division by $[\epsilon^6]$, two rounds of ougtoing scaling lead to this stage when processing the four-state perturbations.

4 Discussion

This section studies two special cases of our setting: first, how assumptions that are stronger than Assumption 5 make not only some proofs easier but also one result stronger; second, how far Young's technique can be generalized. Then we notice that the termination of our algorithm defines an induction proof principle, which is used to show that the algorithm computes a well-known object when fed a strongly connected graph. Eventually, we discuss how to give the so-far-informal notion of time scale a formal flavor.

Stronger assumption

We consider Assumption 28, a stronger version of Assumption 5. It yields Proposition 29, a stronger version of Proposition 14.1. (The proofs are similar but the new one is simpler.)

▶ **Assumption 28.** *If $x \neq y$ and $p(x, y)$ is non-zero, it is positive; and $f \cong g$ or $f \in o(g)$ or $g \in o(f)$ for all f and g in the multiplicative closure of the $\epsilon \mapsto p_\epsilon(x, y)$ with $x \neq y$.*

▶ **Proposition 29.** *Let a perturbation p with state space S satisfy Assumption 28, and let μ be a stationary distribution map for p. If y is a transient state, $\lim_{\epsilon \to 0} \mu_\epsilon(y) = 0$.*

Under Assumption 5 some states may be neither stable nor fully vanishing: y in Figure 8 and x in Figure 1 where the bottom ϵ^2 is replaced with ϵ. Assumption 28 rules out such cases.

▶ **Corollary 30.** *States of perturbations under Assumption 28 are stable or fully vanishing.*

Generalization of Young's technique

Our proof of existence of and computation of the stable states of a perturbation are very different from Young's [17] who uses a finite version of the Markov chain tree theorem. Here we investigate how far Young's technique can be generalized. This will suggest that we were right to change approaches, but it will also yield a decidability result in Proposition 34.

Lemma 31 generalizes [17, Lemma 1]. Both proofs use the Markov chain tree theorem, but they are significantly different nonetheless. Let p be a perturbation with state space S. As in [17] or [8], for all $x \in S$ let $\mathcal{T}_x$ be the set of the spanning trees of (the complete graph of) $S \times S$ that are directed towards x. For all $x \in S$ let $\beta^x_\epsilon := \max_{T \in \mathcal{T}_x} \prod_{(z,t)\in T} p_\epsilon(z,t)$.

▶ **Lemma 31.** *A state x of irreducible p with state space S is stable iff $\beta^y \precsim \beta^x$ for all $y \in S$.*

Assumption 5 and Lemma 31 together yield Observation 32, a generalization of existing results about existence of stable states, such as [17, Theorem 4]. The underlying algorithm runs in time $O(n^3)$ where n is the number of states, just like Young's.

▶ **Observation 32.** *Let a perturbation p on state space S satisfy Assumption 5. If for all $x \neq y$ the map $p(x,y)$ is either identically zero or strictly positive, p has stable states.*

The stable states of a perturbation are computable even without the positivity assumption from Observation 32, but their existence is no longer guaranteed by the same proof. In this way, Observation 33 is like the existential part of Theorem 2, but with a bad complexity.

▶ **Observation 33.** *Let F be a set of perturbation maps of type $I \to [0,1]$ for some I. Let us assume that F is closed under multiplication by elements in F and by characteristic functions of decidable subsets of I, that $\precsim$ is decidable on $F \times F$, and that the supports of the functions in F are uniformly decidable. If $f \precsim g$ or $g \precsim f$ for all $f, g \in F$, stability is decidable in $O(n^5)$ for the perturbations p such that $x \neq y \Rightarrow p(x,y) \in F$.*

The assumption $f \precsim g$ or $g \precsim f$ for all $f, g \in F$ from Observation 33 relates to Assumption 5. Proposition 34 drops it while preserving decidability of stability, but at the cost of an exponential blow-up since the supports of the maps are no longer ordered by inclusion.

▶ **Proposition 34.** *Let F be a set of maps of type $I \to [0,1]$ for some I. Let us assume that F is closed under multiplication by elements in F and by characteristic functions of decidable subsets of I, that $\precsim$ is decidable on $F \times F$, and that the supports of the functions in F are uniformly decidable. Then stability is decidable for the p such that $x \neq y \Rightarrow p(x,y) \in F$.*

What does Algorithm 1 compute?

Applying sequentially the scaling, collapse, and deletion terminates, so it amounts to an *induction proof principle* for finite graphs with arcs labeled in an ordered-division semiring. Observation 35 is proved along this principle. It can also be proved by an indirect argument using Lemma 31 and Theorem 27, but the inductive proof is simple and from scratch.

▶ **Observation 35.** *Let $(F, 0, 1, \cdot, \leq, \div)$ be an ordered-division semiring, and let $P : S \times S \to F$ correspond to a strongly connected digraph, where an arc is absent iff its weight is 0. Then* `FindHub`(S, P) *returns the roots of the maximum directed spanning trees.*

Note that finding the roots from Observation 35 is also doable in $O(n^3)$ by computing the maximum spanning trees rooted at each vertex, by [7] which uses the notion of *heap*, whereas `FindHub` uses a less advanced algorithm.

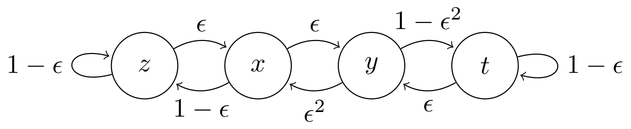


Figure 26 Vanishing time scale.

Vanishing time scales

Computing `FindHub` induces an order in which the states are found to be vanishing. Under the stronger Assumption 28, a notion of *vanishing time scale* may be defined, with the flavor of non-standard analysis [15]. Let $(\mathcal{T}, \cdot)$ be a group of functions $I \to]0, +\infty[$ such that $f \cong g$ or $f \in o(g)$ or $g \in o(f)$ for all f and g in $\mathcal{T}$. The elements of $[\mathcal{T}]$ are called the time scales. Let p over states S satisfy Assumption 28 and let $x \in S$ be deleted at the d-th recursive call of `FindHub`$(S, [p])$. Let $M_1, \dots, M_d$ be the maxima (*i.e.* M) from Line 7 in Algorithm 1 at the 1st,...,d-th recursive calls, respectively. We say that x vanishes at time scale $\prod_{1\leq i\leq d} M_i^{-1}$.

Figure 26 suggests that a similar account of vanishing times scales, even just a qualitative one, would be much more difficult to obtain by invoking the Markov chain tree theorem as in [17]. The only stable state is t; the state z vanishes at time scale $[\epsilon]^{-2}$; and x and y vanish at the same time scale [1] although the maximum spanning trees rooteed at x and y have different weights: ϵ^4 and ϵ^3, respectively.

Acknowledgements. We thank Ocan Sankur, Yvan Le Borgne, and anonymous referees for useful comments.

References

1 Volker Betz and Stéphane Le Roux. Stable states of perturbed Markov chains, 2016. http://arxiv.org/abs/1508.05299v2.
2 E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. `doi:10.1007/BF01386390`.
3 Glenn Ellison. Basins of attraction, long-run stochastic stability, and the speed of step-by-step evolution. *Review of Economic Studies*, 67:17–45, 2000.
4 H. Eyring. The activated complex in chemical reactions. *J. Chem. Phys.*, 3:107115, 1935.
5 Dean Foster and Peyton Young. Stochastic evolutionary game dynamics. *Theoretical Population Biology*, 38:219–232, 1990.
6 M. I. Freidlin and A. D. Wentzell. *Random perturbations of dynamical systems*. Springer-Verlag, second edition, 1998.
7 HN Gabow, Z Galil, T Spencer, and RE Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.
8 Buket Benek Gursoy, Steve Kirkland, Oliver Mason, and Sergei Sergeev. The Markov chain tree theorem in commutative semirings and the state reduction algorithm in commutative semifields. *Linear Algebra and its Applications*, 468:184–196, 2015. 18th ILAS Conference.
9 A. Greenwald J.R. Wicks. An algorithm for computing stochastically stable distributions with applications to multiagent learning in repeated games. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, 2005. http://arxiv.org/abs/1207.1424.
10 Michihiro Kandori, George J. Mailath, and Rafael Rob. Learning, mutation, and long run equilibria in games. *Econometrica*, 61:29–56, 1993.
11 Hans-Helmut Kohler and Eva Vollmerhaus. The frequency of cyclic processes in biological multistate systems. *J. Math. Biology*, 9:275–290, 1980.

12 H.A. Kramers. Brownian motion in a field of force and the diffusion model of chemical reactions. *Physica*, 7:284304, 1940.

13 FT Leighton and RL Rivest. The Markov chain tree theorem. Technical Report LCS-TM-249, MIT, 1983.

14 M. Leyzorek, R.S. Gray, A.A. Johnson, W.C. Ladew, S.R. Meaker, R.M. Petry Jr, and R.N. Seitz. A study of model techniques for communication systems. Technical report, Case Institute of Technology, Cleveland, Ohio„ 1957. Investigation of Model Techniques, First Annual Report.

15 Abraham Robinson. *Non-standard Analysis*. Princeton University Press, 1974.

16 J. Maynard Smith and G.R. Price. The logic of animal conflicts. *Nature*, 246:15–18, 1973.

17 Peyton Young. The evolution of conventions. *Econometrica*, 61:57–84, 1993.

On Degeneration of Tensors and Algebras*

Markus Bläser[1] and Vladimir Lysikov[2]

1 **Department of Computer Science, Saarland University, Saarbrücken, Germany**
`mblaeser@cs.uni-saarland.de`

2 **Cluster of Excellence MMCI and Department of Computer Science, Saarland University, Saarbrücken, Germany**
`vlysikov@cs.uni-saarland.de`

Abstract

An important building block in all current asymptotically fast algorithms for matrix multiplication are tensors with low border rank, that is, tensors whose border rank is equal or very close to their size. To find new asymptotically fast algorithms for matrix multiplication, it seems to be important to understand those tensors whose border rank is as small as possible, so called tensors of minimal border rank.

We investigate the connection between degenerations of associative algebras and degenerations of their structure tensors in the sense of Strassen. It allows us to describe an open subset of $n \times n \times n$ tensors of minimal border rank in terms of smoothability of commutative algebras. We describe the smoothable algebra associated to the Coppersmith-Winograd tensor and prove a lower bound for the border rank of the tensor used in the "easy construction" of Coppersmith and Winograd.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases bilinear complexity, border rank, commutative algebras, lower bounds

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.19

1 Introduction

Let V_1, V_2, V_3 be vector spaces. The tensor product $V_1 \otimes V_2 \otimes V_3$ is spanned by tensors of the form $v_1 \otimes v_2 \otimes v_3$, which are called *decomposable* tensors, i. e., any tensor $T \in V_1 \otimes V_2 \otimes V_3$ can be represented as a sum

$$T = \sum_{s=1}^{r} v_{1,s} \otimes v_{2,s} \otimes v_{3,s}. \tag{1}$$

This representation is called a *polyadic decomposition* of T. The minimal number of summands in a polyadic decomposition of T is called the *rank* of T. Tensor rank is a direct generalization of the usual notion of matrix rank, which can be defined as a minimal number of summands in a representation of a matrix as a sum of rank one matrices. Unlike in the matrix case, the set $\mathbf{R}_r$ of all tensors of rank at most r is in general not closed, so it is useful to consider not only exact polyadic decompositions, but also approximations of tensors by sums of the form (1). Given a tensor T, the minimal number r such that T is contained in the closure of $\mathbf{R}_r$ is called the *border rank* of T.

Rank and border rank of tensors have diverse applications (see [3, 7, 9] for more information). Our motivation originates from computational complexity theory of bilinear maps. Any

* This work was partially supported by the Deutsche Forschungsgemeinschaft under grant BL 511/10-1.

bilinear map $\varphi\colon U \times V \to W$ between finite-dimensional vector spaces is a contraction with some tensor from $U^* \otimes V^* \otimes W$, called the *structural tensor* of φ. Polyadic decompositions of the structural tensor can be interpreted as algorithms of a certain kind for computing φ, and the rank of the structural tensor of a bilinear map is a measure of its computational complexity. See [1] for a detailed exposition of bilinear complexity theory.

One interesting problem in this area is the classification of concise tensors of minimal border rank. A tensor from $V_1 \otimes V_2 \otimes V_3$ is *concise* if it is not contained in any proper subspace $V_1' \otimes V_2' \otimes V_3' \subset V_1 \otimes V_2 \otimes V_3$. The border rank of a concise tensor is bounded from below by $\max\{\dim V_i\}$. Tensors for which this bound is exact are called tensors *of minimal border rank*.

Tensors of minimal border rank correspond to bilinear maps that have low complexity and can be used to construct efficient bilinear algorithms. For example, the famous Coppersmith-Winograd algorithm for matrix multiplication [4] (as well as its recent improvements [12]) uses a tensor of minimal border rank as a basic block. Such a tensor, like the Coppersmith-Winograd tensor, usually appears “out of the blue” and this starting tensor, which at a first glance has only very little to do with matrix multiplication, is then used to design a fast matrix multiplication algorithm by looking at high powers of the starting tensor. Therefore, to make further progress in the design of fast matrix multiplication algorithms, a systematic description of the tensors of minimal border rank seems to be very helpful. As our first result, we describe (an open subset) of the tensors of minimal border rank in terms of their structure. It turns out, that these tensors are the multiplication tensors of so-called smoothable commutative algebras. These algebras are studied in algebraic geometry in connection with Hilbert schemes of points, and have received quite some attention in the recent years, however, their structure is not fully understood.

Recently, Landsberg and Michałek [10] described tensors of minimal border rank in $\mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$ that have a slice of rank n in terms of certain Lie algebras constructed from the slices of the tensor. In this paper we consider a slightly stronger condition, namely, existence of rank-n slices in two slicing directions, and prove that any tensor of minimal border rank satisfying this condition is equivalent to a structure tensor of a smoothable commutative algebra.

Furthermore, we describe a method which can be though of as a limiting version of the substitution method for the rank lower bounds (essentially the same method was independently described by Landsberg and Michałek [11]) and use it to prove a lower bound on the border rank of tensor powers of the restricted Coppersmith-Winograd tensor used in the “easy construction” of [4]. This easy tensor is not a tensor of minimal border rank. However, as pointed out in [1, Rem. 15.44], if this tensor had asymptotically minimal border rank, then the exponent of matrix multiplication would be 2. Asymptotically minimal border rank means that the border rank of the tensor powers converges to the size of the tensor. While our bound is nontrivial, it does not rule out that the easy tensor has asymptotically minimal border rank.

2 Preliminaries

2.1 Notation and basic definitions

All vector spaces are presumed to be finite-dimensional vector spaces over some fixed algebraically closed field k. Letters U, V, W, possibly indexed, denote vector spaces, A denotes algebras. ε usually denotes some indeterminate. Linear maps and tensors over $k(\varepsilon)$ are rendered in calligraphic font.

We do not distinguish between bilinear maps $U \times V \to W$ and corresponding tensors in $U^* \otimes V^* \otimes W$. When there is no confusion, multiplication in some algebra as a bilinear map is denoted by the same symbol as the algebra itself. In particular, k^r denotes the coordinate-wise multiplication of r-dimensional vectors.

A bilinear map $\varphi \in V^* \otimes V^* \otimes V$ is called *unital* if there is an *identity element* $e \in V$ such that $\varphi(e, x) = \varphi(x, e) = x$ for all $x \in V$.

Any tensor in $V_1 \otimes V_2 \otimes V_3$ with $\dim V_i = n_i$ is said to have *format* $n_1 \times n_2 \times n_3$. We are mostly interested in tensors of format $n \times n \times n$.

Rank and border rank of a tensor T are denoted by $R(T)$ and $\underline{R}(T)$ respectively.

The Zariski closure of a set $\mathbf{S}$ is denoted by $\overline{\mathbf{S}}$. We use the Zariski topology to define the border rank of tensors over k. However, over $\mathbb{C}$, the Zariski closure of the set of all tensors in $V_1 \otimes V_2 \otimes V_3$ of rank at most r coincides with its Euclidean closure, thus capturing the idea of approximation.

Let $T \in V_1 \otimes V_2 \otimes V_3$ and $T' \in V_1' \otimes V_2' \otimes V_3'$ be two tensors. T' is a *restriction* of T (denoted $T' \leq T$) if there exists a triple of linear maps $F_i \colon V_i \to V_i'$ such that $T' = (F_1 \otimes F_2 \otimes F_3)T$. We call the operator $(F_1 \otimes F_2 \otimes F_3)$ *a restriction operator* for $T' \leq T$.

Two tensors T_1 and T_2 are called *equivalent* ($T_1 \sim T_2$) if $T_1 \leq T_2$ and $T_2 \leq T_1$. If T_1 and T_2 have the same format, they are equivalent iff there is a bijective restriction operator for $T_1 \leq T_2$.

The tensor rank can be defined via restrictions of k^r (or, equivalently, diagonal tensors $\sum_{i=1}^r e_i \otimes e_i \otimes e_i$): polyadic decompositions of a tensor T are in one-to-one correspondence with restriction operators for $T \leq k^r$, so $R(T) \leq r$ iff $T \leq k^r$.

For more information, we refer to [1].

2.2 Degeneration of tensors

Degeneration of tensors was introduced by Strassen [15]. It is an approximate analogue of restriction: a tensor T' is a *degeneration* of T (denoted $T' \trianglelefteq T$) if

$$T' \in \overline{\{t \in V_1' \otimes V_2' \otimes V_3 \mid t \leq T\}}.$$

Strassen gives alternative descriptions of degeneration. One of these descriptions is in terms of representation theory. Consider the group $G = \mathrm{GL}(V_1) \times \mathrm{GL}(V_2) \times \mathrm{GL}(V_3)$. It acts on $V_1 \otimes V_2 \otimes V_3$ in a standard way:

$$(F_1, F_2, F_3) \cdot T = (F_1 \otimes F_2 \otimes F_3)T.$$

The orbits of this action are equivalence classes of tensors in $V_1 \otimes V_2 \otimes V_3$.

▶ **Lemma 2.1** (Strassen [15]). *Let $T \in V_1 \otimes V_2 \otimes V_3$ and $T' \in V_1' \otimes V_2' \otimes V_3'$ be two tensors. $T' \trianglelefteq T$ if and only if there exists a tensor $S \in V_1 \otimes V_2 \otimes V_3$ such that $T' \sim S$ and $S \in \overline{G \cdot T}$.*

Another description of degeneration uses base extension from k to $k(\varepsilon)$. For any vector space V over k its base extension $V(\varepsilon) = V \otimes k(\varepsilon)$ is a vector space over $k(\varepsilon)$. We have an injection $V \hookrightarrow V(\varepsilon)$ defined by $v \mapsto v \otimes 1$. Analogously, $V_1 \otimes V_2 \otimes V_3$ injects into $V_1(\varepsilon) \otimes_{k(\varepsilon)} V_2(\varepsilon) \otimes_{k(\varepsilon)} V_3(\varepsilon)$, so any tensor $T \in V_1 \otimes V_2 \otimes V_3$ can be viewed as a tensor over $k(\varepsilon)$, which we also denote by T.

Going in the other direction, we have a partial map from $k(\varepsilon)$ to k that takes a rational function regular at $\varepsilon = 0$ to its value at 0. It can be extended to partial maps from $V(\varepsilon)$ to V for each vector space V. If $\mathcal{T}|_{\varepsilon=0} = T$, we sometimes write $\mathcal{T} = T + O(\varepsilon)$, thinking of ε as an infinitesimal.

▶ **Lemma 2.2** (Strassen [15]). *Let $T \in V_1 \otimes V_2 \otimes V_3$ and $T' \in V_1' \otimes V_2' \otimes V_3'$ be two tensors. $T' \trianglelefteq T$ if and only if there exists $\mathcal{T} \in V_1(\varepsilon) \otimes_{k(\varepsilon)} V_2(\varepsilon) \otimes_{k(\varepsilon)} V_3(\varepsilon)$ such that $\mathcal{T}|_{\varepsilon=0} = T'$ and $\mathcal{T} \leq T$ as tensors over $k(\varepsilon)$.*

This lemma allows us to talk about specific ways in which T degenerates into T', which are represented by restriction operators for restrictions of the form $T' + O(\varepsilon) \leq T$ considered in the lemma. We call these operators *degeneration operators* for $T' \trianglelefteq T$.

Degenerations of k^r are related to border rank in the same way its restrictions are related to rank: since $R(\varphi) \leq r$ iff $\varphi \leq k^r$, by taking closures we have $\underline{R}(\varphi) \leq r$ iff $\varphi \trianglelefteq k^r$. In particular, Lemma 2.2 implies existence of *approximate polyadic decompositions*

$$T + O(\varepsilon) = \sum_{s=1}^{r} \mathcal{v}_{1,s} \otimes_{k(\varepsilon)} \mathcal{v}_{2,s} \otimes_{k(\varepsilon)} \mathcal{v}_{3,s}$$

2.3 Degeneration of algebras

Strassen's theory of tensor degenerations was inspired by the similar concept in the deformation theory of algebras.

Degeneration of algebras is usually restricted to associative or Lie algebras, but we define it for arbitrary bilinear maps in $V^* \otimes V^* \otimes V$, which can be thought of as nonassociative algebra structures on V. The group $\mathrm{GL}(V)$ acts on $V^* \otimes V^* \otimes V$ by change of basis:

$$(g \cdot \varphi)(x, y) = g\varphi(g^{-1}x, g^{-1}y).$$

The orbits of this action are isomorphism classes of nonassociative algebras.

Let $\varphi, \varphi' \in V^* \otimes V^* \otimes V$. We call φ' an *algebraic degeneration* of φ (denoted $\varphi' \trianglelefteq_{\mathrm{a}} \varphi$) if φ' lies in the orbit closure $\overline{\mathrm{GL}(V) \cdot \varphi}$. The name "algebraic degeneration" is used here to distinguish between two different notions of degeneration on $V^* \otimes V^* \otimes V$ and does not appear in the literature on degeneration of algebras. It is easy to see that $\varphi' \trianglelefteq_{\mathrm{a}} \varphi$ implies $\varphi' \trianglelefteq \varphi$.

We can extend the definition of algebraic degeneration to bilinear maps on different spaces of the same dimension by saying that if φ' is an algebraic degeneration of φ, then any ψ' isomorphic to φ' as a nonassociative algebra is also a algebraic degeneration of φ.

Since associativity and commutativity properties define closed subsets of $V^* \otimes V^* \otimes V$, degenerations of associative (resp. commutative) algebras are themselves associative (commutative).

▶ **Definition 2.3.** An unital algebra A of dimension n such that $A \trianglelefteq_{\mathrm{a}} k^n$ is called *smoothable*.

As follows from previous discussion, smoothable algebras are always associative and commutative.

In the geometric study of finite-dimensional commutative algebras they are sometimes studied as elements of a variety in $V^* \otimes V^* \otimes V$ or a similar scheme, and sometimes — as elements of a Hilbert scheme of points $\mathbf{Hilb}_n(\mathbb{A}_k^d)$, which parameterizes 0-dimensional schemes on d-dimensional affine plane, or, equivalently, ideals I in $R = k[x_1, \ldots, x_d]$ such that R/I is an n-dimensional algebra. The exact relationship between these two approaches is explored in [14]. We will only need the fact that topologies on $V^* \otimes V^* \otimes V$ and on $\mathbf{Hilb}_n(\mathbb{A}_k^d)$ give the same notion of smoothability, so we can use results from [2, 6] formulated in the language of Hilbert schemes.

There are analogues of Lemma 2.2 for algebraic degeneration (for example, [8, § 3.9] gives a geometric formulation of a similar statement). We only need the easier part of the equivalence which says that if φ' is approximated by bilinear maps isomorphic to φ, then it is an algebraic degeneration of φ.

▶ **Lemma 2.4.** *Let $\varphi, \varphi' \in V^* \otimes V^* \otimes V$ be two bilinear maps on V. If there exists an invertible $k(\varepsilon)$-linear map $\mathcal{F}\colon V(\varepsilon) \to V(\varepsilon)$ such that*

$$\mathcal{F}^{-1}\varphi(\mathcal{F}x, \mathcal{F}y)|_{\varepsilon=0} = \varphi'(x, y) \quad \textit{for all } x, y \in V,$$

then $\varphi' \trianglelefteq_{\mathrm{a}} \varphi$.

Proof. As ε varies, the bilinear map $\varphi^\varepsilon(x, y) = \mathcal{F}^{-1}\varphi(\mathcal{F}x, \mathcal{F}y)$ traces an algebraic curve in $V^* \otimes V^* \otimes V$. Since $\mathcal{F}$ is invertible, its values for Zariski almost all ε are also invertible, so an open subset of the curve $\{\varphi^\varepsilon\}$ lies in the orbit $\mathrm{GL}(V) \cdot \varphi$. Therefore, the value at $\varepsilon = 0$ lies in the closure of this orbit. ◀

We can rephrase this lemma as follows: tensor degeneration $\varphi' \trianglelefteq \varphi$ with a degeneration operator of the form $\mathcal{F}^* \otimes \mathcal{F}^* \otimes \mathcal{F}^{-1}$ implies algebraic degeneration $\varphi' \trianglelefteq_{\mathrm{a}} \varphi$.

3 Degenerations of associative algebras

In this section and later *algebra* means associative unital algebra over k.

3.1 Transformations of degeneration operators

Suppose $T \in V_1 \otimes V_2 \otimes V_3$ and $T' \in V_1' \otimes V_2' \otimes V_3'$ are two tensors such that $T' \trianglelefteq T$. Denote by $D(T' \trianglelefteq T)$ the set of all degeneration operators for $T' \trianglelefteq T$.

Let us describe some groups that act on $D(T' \trianglelefteq T)$. These groups are subgroups of $\mathrm{GL}(V_1(\varepsilon)) \times \mathrm{GL}(V_2(\varepsilon)) \times \mathrm{GL}(V_3(\varepsilon))$ and $\mathrm{GL}(V_1'(\varepsilon)) \times \mathrm{GL}(V_2'(\varepsilon)) \times \mathrm{GL}(V_3'(\varepsilon))$ which act on the domain and image of operators in $D(T' \trianglelefteq T)$ in the usual way (a triple $(\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3)$ acts via $\mathcal{F}_1 \otimes \mathcal{F}_2 \otimes \mathcal{F}_3$).

Let $T \in V_1 \otimes V_2 \otimes V_3$ be a tensor. Its *isotropy group* $\Gamma(T)$ is defined as the subgroup of $\mathrm{GL}(V_1) \times \mathrm{GL}(V_2) \times \mathrm{GL}(V_3)$ which leaves T fixed. Isotropy groups of bilinear maps and their action on the set of all bilinear algorithms were studied by de Groote [5]. Similarly, we define the *ε-isotropy group* $\Gamma^\varepsilon(T)$ of T as the subgroup of $\mathrm{GL}(V_1(\varepsilon)) \times \mathrm{GL}(V_2(\varepsilon)) \times \mathrm{GL}(V_3(\varepsilon))$ that fixes T considered as a tensor over $k(\varepsilon)$.

Suppose $\mathcal{F} \in \mathrm{GL}(V(\varepsilon))$ is a $k(\varepsilon)$-linear map such that $\mathcal{F} = \mathrm{id} + O(\varepsilon)$. Then for each $\nu \in V(\varepsilon)$ we have $\mathcal{F}\nu|_{\varepsilon=0} = \nu|_{\varepsilon=0}$, when $\nu|_{\varepsilon=0}$ is defined. Let $E(V_1, V_2, V_3)$ be the subgroup of $\mathrm{GL}(V_1(\varepsilon)) \times \mathrm{GL}(V_2(\varepsilon)) \times \mathrm{GL}(V_3(\varepsilon))$ consisting of all triples of such operators.

▶ **Lemma 3.1.** *The groups $\Gamma(T')$ and $E(V_1', V_2', V_3')$ act on $D(T' \trianglelefteq T)$ on the left and $\Gamma^\varepsilon(T)$ acts on the right via composition.*

Proof. Let $\mathcal{F} = \mathcal{F}_1 \otimes \mathcal{F}_2 \otimes \mathcal{F}_3$ be a degeneration operator for $T' \trianglelefteq T$, i. e., $T' + O(\varepsilon) = \mathcal{F}T$. The described actions preserve this relation, since if $G \in \Gamma(T')$, then $GT' = T'$ and $G(O(\varepsilon)) = O(\varepsilon)$; if $\mathcal{G} \in E(V_1', V_2', V_3')$, then $\mathcal{G}(T' + O(\varepsilon)) = T' + O(\varepsilon)$; and if $\mathcal{G} \in \Gamma^\varepsilon(T)$, then $\mathcal{G}T = T$. ◀

We use these transformations in case when T is the structure tensor of some algebra. Suppose A is an algebra and a, b, c are three invertible elements of A. Let L_x and R_x denote left and right multiplication by x respectively. Then $((L_a R_b)^*, (L_b^{-1} R_c)^*, L_a^{-1} R_c^{-1})$ is an element of the isotropy group $\Gamma(A)$ arising from the identity $xy = a^{-1}(axb)(b^{-1}yc)c^{-1}$. The use of this identity is sometimes called *sandwiching* in the literature. Since the tensor over $k(\varepsilon)$ corresponding to A is $A(\varepsilon) = A \otimes k(\varepsilon)$, an analogous expression with $a, b, c \in A(\varepsilon)$ can be used to construct elements of $\Gamma^\varepsilon(A)$.

3.2 Main theorem

▶ **Theorem 3.2.** *Let A be an algebra and $\varphi \in A^* \otimes A^* \otimes A$ be a unital bilinear map. Then $\varphi \trianglelefteq A$ iff $\varphi \trianglelefteq_{\mathrm{a}} A$.*

Proof. The implication $\varphi \trianglelefteq_{\mathrm{a}} A \Rightarrow \varphi \trianglelefteq A$ is obvious. Let us prove the opposite implication.

Let $\varphi \trianglelefteq A$ and $\mathcal{F}^* \otimes \mathcal{G}^* \otimes \mathcal{H}$ be a degeneration operator, i. e.,

$$\varphi(x, y) = \mathcal{H}(\mathcal{F}x \cdot \mathcal{G}y)|_{\varepsilon=0} \quad \text{for all } x, y \in A,$$

where the multiplication is in $A \otimes k(\varepsilon)$.

Let e be the identity element of φ. After the substitution $x = e$ we have

$$y = \varphi(e, y) = \mathcal{H}(\mathcal{F}e \cdot \mathcal{G}y)|_{\varepsilon=0} = \mathcal{H}L_{\mathcal{F}e}\mathcal{G}y|_{\varepsilon=0} \quad \text{for all } y \in A,$$

so $\mathcal{Q} := \mathcal{H}L_{\mathcal{F}e}\mathcal{G} = \mathrm{id} + O(\varepsilon)$. Applying $(\mathrm{id}, (\mathcal{Q}^{-1})^*, \mathrm{id}) \in E(A^*, A^*, A)$ to the degeneration operator $\mathcal{F}^* \otimes \mathcal{G}^* \otimes \mathcal{H}$, we obtain a new degeneration operator $\mathcal{F}^* \otimes \hat{\mathcal{G}}^* \otimes \mathcal{H}$ where $\hat{\mathcal{G}} = \mathcal{G}\mathcal{Q}^{-1} = L_{\mathcal{F}e}^{-1}\mathcal{H}^{-1}$.

Analogously, setting $y = e$ we get that $\mathcal{P} := \mathcal{H}R_{\hat{\mathcal{G}}e}\mathcal{F} = \mathrm{id} + O(\varepsilon)$ and using transformation $((\mathcal{P}^{-1})^*, \mathrm{id}, \mathrm{id}) \in E(A^*, A^*, A)$ we get another degeneration operator $\hat{\mathcal{F}}^* \otimes \hat{\mathcal{G}}^* \otimes \mathcal{H}$ where $\hat{\mathcal{F}} = \mathcal{F}\mathcal{P}^{-1} = R_{\hat{\mathcal{G}}e}^{-1}\mathcal{H}^{-1}$.

Finally, we use a sandwiching transformation $((L_{\mathcal{F}e}^{-1})^*, (R_{\hat{\mathcal{G}}e}^{-1})^*, L_{\mathcal{F}e}R_{\hat{\mathcal{G}}e})$ from $\Gamma^\varepsilon(A)$ and obtain a degeneration operator $\mathcal{S}^* \otimes \mathcal{S}^* \otimes \mathcal{S}^{-1}$ where

$$\mathcal{S} = (\mathcal{H}L_{\mathcal{F}e}R_{\hat{\mathcal{G}}e})^{-1}.$$

By Lemma 2.4 we have an algebraic degeneration $\varphi \trianglelefteq_{\mathrm{a}} A$. ◀

This theorem can be seen as an extension of the fact that associative algebras have equivalent structure tensors iff they are isomorphic ([1, Prop. 14.13]). The general idea of the proof — using symmetries of the tensors to transform maps that express the relationship between them — goes back to de Groote [5], but in our case some care needed to track the behaviour of degeneration operators as ε varies.

3.3 Tensors of minimal border rank

A special case of Theorem 3.2 when the algebra A is k^r can be used to study tensors of minimal border rank. First, we describe algebras of minimal border rank:

▶ **Corollary 3.3.** *A unital bilinear map on a vector space of dimension n is of minimal border rank iff it is a multiplication in a smoothable algebra.*

Proof. By Theorem 3.2 in the present case it is equivalent to $\varphi \trianglelefteq_{\mathrm{a}} k^n$, which is the definition of a smoothable algebra. ◀

For example, if $\operatorname{char} k \neq 2, 3$, the following algebras are smoothable [2], and, therefore, have minimal border rank:

1. any algebra generated by 2 elements;
2. any algebra of the form $k[x_1, \dots, x_d]/I$ where the ideal I is monomial;
3. any algebra with $\dim(R^2/R^3) = 1$ where $R = \operatorname{rad} A$;
4. any algebra with $\dim(R^2/R^3) = 2$, $\dim R^3 \leq 2$ and $R^4 = 0$ where $R = \operatorname{rad} A$;
5. any algebra of dimension 7 or less;

A description of smoothable algebras of dimension 8 is contained in [2, 6].

Using the description of algebras of minimal border rank, we can identify a certain open subset of tensors of minimal border rank.

▶ **Definition 3.4.** A tensor $T \in V_1 \otimes V_2 \otimes V_3$ of format $n \times n \times n$ is called *binding* if there are elements $\alpha_1 \in V_1^*$ and $\alpha_2 \in V_2^*$ such that the contractions $T\alpha_1 \in V_2 \otimes V_3$ and $T\alpha_2 \in V_1 \otimes V_2$ have rank n.

Note that a generic tensor of format $n \times n \times n$ is binding. In the terminology of [10] binding tensors are called 1_{V_1}- and 1_{V_2}-generic. We call these tensors binding because they allow us to relate spaces V_1 and V_2 to V_3 similarly to how a nondegenerate bilinear form allows to view spaces of its arguments as dual to each other. This is used in the proof of the following lemma.

▶ **Lemma 3.5.** *A binding tensor is equivalent to an unital bilinear map.*

Proof. Let $\dim V_1 = \dim V_2 = \dim V_3 = n$ and $T \in V_1 \otimes V_2 \otimes V_3$ be a binding tensor. Let $\alpha_1 \in V_1^*$ and $\alpha_2 \in V_2^*$ be as in Definition 3.4.

We can view $T\alpha_1$ and $T\alpha_2$ as linear isomorphisms $P_1 \colon V_1^* \to V_3$ and $P_2 \colon V_2^* \to V_3$. Applying $(P_2^{-1})^* \otimes (P_1^{-1})^* \otimes \mathrm{id}$ to T we get an equivalent bilinear map

$$\varphi(x_1, x_2) = T(P_2^{-1}x_1)(P_1^{-1}x_2).$$

This bilinear map is unital, since $\varphi(P_2\alpha_1, x) = x$ and $\varphi(x, P_1\alpha_2) = x$ for all $x \in V_1$, so

$$P_2\alpha_1 = \varphi(P_2\alpha_1, P_1\alpha_2) = P_1\alpha_2$$

is the identity element. ◀

▶ **Corollary 3.6.** *A binding tensor has minimal border rank iff it is equivalent to a smoothable algebra.*

These results suggest that structure tensors of smoothable algebras are possible candidates for basic blocks to construct fast matrix multiplication algorithms. We tried to use some of them in the same framework that is used by Coppersmith and Winograd (it is known as "laser method", see [1, 12] for more information). So far, these attempts did not lead to improved matrix multiplication algorithms.

3.4 Example: Coppersmith-Winograd tensor

Let $e^{[0]}, e_1^{[1]}, \ldots, e_q^{[1]}, e^{[2]}$ be a basis of a $(q+2)$-dimensional vector space, and $\alpha^{[0]}, \alpha_i^{[1]}, \alpha^{[2]}$ be the dual basis. The famous Coppersmith-Winograd algorithm [4] uses the tensor

$$\begin{aligned} T_{CW} = \sum_{i=1}^{q} &(e^{[0]} \otimes e_i^{[1]} \otimes e_i^{[1]} + e_i^{[1]} \otimes e^{[0]} \otimes e_i^{[1]} + e_i^{[1]} \otimes e_i^{[1]} \otimes e^{[0]}) + \\ &+ e^{[0]} \otimes e^{[0]} \otimes e^{[2]} + e^{[0]} \otimes e^{[2]} \otimes e^{[0]} + e^{[2]} \otimes e^{[2]} \otimes e^{[0]}, \end{aligned}$$

which we will call *Coppersmith-Winograd tensor*.

We can use the results of the previous section to exhibit a smoothable algebra with the structure tensor equivalent to the Coppersmith-Winograd tensor.

The Coppersmith-Winograd tensor is a tensor of minimal border rank, as witnessed by the approximate decomposition

$$\begin{aligned}T_{CW} + O(\varepsilon) = \varepsilon^{-2}\sum_{i=1}^{q}(e^{[0]} + \varepsilon e_i^{[1]}) \otimes (e^{[0]} + \varepsilon e_i^{[1]}) \otimes (e^{[0]} + \varepsilon e_i^{[1]}) - \\ - \varepsilon^{-3}(e^{[0]} + \varepsilon^2\sum_{i=1}^{q} e_i^{[1]}) \otimes (e^{[0]} + \varepsilon^2\sum_{i=1}^{q} e_i^{[1]}) \otimes (e^{[0]} + \varepsilon^2\sum_{i=1}^{q} e_i^{[1]}) + \\ + (\varepsilon^{-3} - q\varepsilon^{-2})(e^{[0]} + \varepsilon^3 e^{[2]}) \otimes (e^{[0]} + \varepsilon^3 e^{[2]}) \otimes (e^{[0]} + \varepsilon^3 e^{[2]}).\end{aligned} \tag{2}$$

The Coppersmith-Winograd tensor T_{CW} is binding (the layers corresponding to $\alpha^{[0]}$ have full rank). Applying Lemma 3.5, we obtain a bilinear map

$$\begin{aligned}\sum_{i=1}^{q}(\alpha^{[2]} \otimes \alpha_i^{[1]} \otimes e_i^{[1]} + \alpha_i^{[1]} \otimes \alpha^{[2]} \otimes e_i^{[1]} + \alpha_i^{[1]} \otimes \alpha_i^{[1]} \otimes e^{[0]}) + \\ + \alpha^{[2]} \otimes \alpha^{[2]} \otimes e^{[2]} + \alpha^{[2]} \otimes \alpha^{[0]} \otimes e^{[0]} + \alpha^{[0]} \otimes \alpha^{[0]} \otimes e^{[0]}\end{aligned}$$

which is unital with the identity $e^{[2]}$. By Corollary 3.6 this map is a multiplication in some smoothable algebra. Denote $e^{[2]}$ by 1 and $e_i^{[1]}$ by x_i. In this notation, $x_i x_j = 0$ for $i \neq j$ and $e^{[0]}$ corresponds to $x_1^2 = x_2^2 = \cdots = x_q^2$. To summarize,

▶ **Example 3.7.** The Coppersmith-Winograd tensors is equivalent to the smoothable algebra $A_{CW} \cong k[x_1, \ldots, x_q]/\left\langle x_i x_j, x_i^2 - x_j^2, x_i^3 \mid i \neq j \right\rangle$.

Performing transformations described in the proof of Theorem 3.2 for the decomposition (2), we can construct an algebraic degeneration of k^{d+2} to A_{CW} given by a degeneration operator $\mathcal{S}^* \otimes \mathcal{S}^* \otimes \mathcal{S}^{-1}$ where $\mathcal{S} \colon A_{CW}(\varepsilon) \to k(\varepsilon)^{q+2}$ has the following matrix relative to the basis $\{1, x_1, \ldots, x_q, x_1^2\}$ in A_{CW} and the standard basis in k^{q+2}:

$$\left[\begin{array}{c|ccccc|c} 1 & \varepsilon - (q-1)\varepsilon^2 & \varepsilon^2 & \varepsilon^2 & \cdots & \varepsilon^2 & -\varepsilon^3 \\ 1 & \varepsilon^2 & \varepsilon - (q-1)\varepsilon^2 & \varepsilon^2 & \cdots & \varepsilon^2 & -\varepsilon^3 \\ 1 & \varepsilon^2 & \varepsilon^2 & \varepsilon - (q-1)\varepsilon^2 & \cdots & \varepsilon^2 & -\varepsilon^3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \varepsilon^2 & \varepsilon^2 & \varepsilon^2 & \cdots & \varepsilon - (q-1)\varepsilon^2 & -\varepsilon^3 \\ 1 & \varepsilon^2 & \varepsilon^2 & \varepsilon^2 & \cdots & \varepsilon^2 & -\varepsilon^3 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 \end{array}\right].$$

We may simplify this matrix by applying a certain linear map of the form id $+O(\varepsilon)$, obtaining a new degeneration corresponding to a matrix

$$\left[\begin{array}{c|ccccc|c} 1 & \varepsilon & 0 & 0 & \cdots & 0 & -\varepsilon^3 \\ 1 & 0 & \varepsilon & 0 & \cdots & 0 & -\varepsilon^3 \\ 1 & 0 & 0 & \varepsilon & \cdots & 0 & -\varepsilon^3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 0 & 0 & 0 & \cdots & \varepsilon & -\varepsilon^3 \\ 1 & \varepsilon^2 & \varepsilon^2 & \varepsilon^2 & \cdots & \varepsilon^2 & -\varepsilon^3 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 \end{array}\right]. \tag{3}$$

In the language of schemes this degeneration can be interpreted as follows: the 0-dimensional scheme $\mathbf{S}_{CW}$ with coordinate ring A_{CW} is the flat limit of the family (parameterized by ε) of schemes containing $q+2$ points in $(q+2)$-dimensional affine space with coordinates given by the rows of the matrix (3).

Since A_{CW} is generated by q elements, $\mathbf{S}_{CW}$ is contained in a q-dimensional affine subspace, so we can consider instead of schemes in $(q+2)$-dimensional space their projections to this subspace, which corresponds to the middle part of (3).

For those unfamiliar with the terminology of schemes, here is an algorithmic interpretation: to approximately multiply two elements of A_{CW}, evaluate the corresponding polynomials of the form $a^{[0]} + \sum a_i^{[1]} x_i + a^{[2]} x_1^2$ at the $q+2$ points given by the middle part of the matrix, multiply the corresponding values, and interpolate the products to get a resulting polynomial.

4 Substitution method for border rank

In this section we describe a method for obtaining lower bounds which can be seen as a border rank version of the substitution method for tensor rank. Let $T \in U^* \otimes V \otimes W$. We can view it as a linear map $U \to V \otimes W$ and consider the restriction $T|_{U'} \in (U')^* \otimes V \otimes W$ for any subspace $U' \subset U$. If the border ranks of $T|_{U'}$ are known, we can derive the bound on the border rank of T.

▶ **Theorem 4.1.** *Let $T \in U^* \otimes V \otimes W$ and $\dim U = n$. For any d we have*

$$\underline{R}(T) \geq n - d + \min\{\underline{R}(T|_{U'}) \mid U' \subset U, \dim U' = d\}.$$

Proof. Suppose $\underline{R}(T) = r$. We can assume that T is concise, considering it as an element of a smaller subspace $(U')^* \otimes V' \otimes W' \subset U^* \otimes V \otimes W$ otherwise. We need to show that there exists a subspace $U' \subset U$, $\dim U' = d$, such that $\underline{R}(T|_{U'}) \leq r - n + d$.

Note that this is true for tensors T of rank r. Indeed, let $T = \sum_{s=1}^r f_s \otimes v_s \otimes w_s$ be a polyadic decomposition. Without loss of generality, $f_1, \dots, f_n$ form a basis of U^*, and for the d-dimensional subspace $U' \subset U$ defined by the equations $f_i = 0$ for $1 \leq i \leq n - d$ we have $\underline{R}(T|_{U'}) \leq R(T|_{U'}) \leq r - n + d$, since the first $n - d$ terms of the decomposition vanish on U'.

Moreover, if we have an approximate decomposition

$$T + O(\varepsilon) = \sum_{s=1}^{r} f_s(\varepsilon) \otimes v_s(\varepsilon) \otimes w_s(\varepsilon) = \mathcal{T},$$

we can assume that $f_1(\varepsilon), \dots, f_n(\varepsilon)$ are linearly independent for almost all values of ε (because concise tensors form an open set), and obtain a family of subspaces U'_ε such that $\mathcal{T}(\varepsilon)|_{U'_\varepsilon}$ has rank at most $r - n + d$. The family U'_ε defines an algebraic curve in the Grassmannian $\mathbf{Gr}(d, U)$. Grassmannians are projective varieties, so U'_ε can be extended to $\varepsilon = 0$ (see, for example, [13, Rem. 7.12, Thm. 7.22]).

Given an isomorphism $F\colon k^d \to U' \subset U$ and a tensor T, we can define $\hat{T} \in (k^d)^* \otimes V \otimes W$ as $\hat{T}(p) = T(Fp)$ so that $\hat{T} \sim T|_{U'}$ and the map $Z\colon (T, F) \mapsto \hat{T}$ is algebraic. In the neighborhood of U'_0 we can choose isomorphisms $\mathcal{F}\colon k^d \to U'_\varepsilon$ which vary continuously with ε. Using these isomorphisms, we include $T|_{U'_0}$ in an algebraic family $Z(\mathcal{T}, \mathcal{F})$ of tensors of rank at most $r - n + d$, therefore, its border rank does not exceed this value. ◀

Essentially the same method was independently described by Landsberg and Michałek [11]. They prove this lower bound when U' is a hyperplane in U (from which the general version follows easily) and use it to obtain a lower bound on the rank of matrix multiplication. We consider the other extremal case where $U' = \langle u \rangle$ is 1-dimensional. In this case $T|_{U'}$ is essentially the matrix $Tu \in V \otimes W$ and, since for matrices rank and border rank coincide, we have

▶ **Corollary 4.2.** $\underline{R}(T) \geq n - 1 + m(T)$ *where* $m(T) = \min_{u \in U \setminus \{0\}} \operatorname{rk}(Tu)$.

4.1 Border rank of the easy Coppersmith-Winograd tensor

In [4], Coppersmith and Winograd first describe a simplified version of the main construction. This "easy version" uses the tensor

$$T_{cw} = \sum_{i=1}^{q} (e^{[0]} \otimes e_i^{[1]} \otimes e_i^{[1]} + e_i^{[1]} \otimes e^{[0]} \otimes e_i^{[1]} + e_i^{[1]} \otimes e_i^{[1]} \otimes e^{[0]}),$$

with $q \geq 2$, which we will call *easy Coppersmith-Winograd tensor.*

The easy Coppersmith-Winograd tensor is a restriction of the full Coppersmith-Winograd tensor obtained using the projection along $e^{[2]}$ onto $\left\langle e^{[0]}, e_i^{[1]} \right\rangle$, so its border rank is at most $q+2$. It is known that this is the exact value of $\underline{R}(T_{cw})$ (see [1, Exercise 15.14(3)]).

We can write a bilinear map equivalent to T_{cw} in terms of the algebra A_{CW} described in §3.4. Let X be the subspace of A_{CW} spanned by x_i, M be the subspace spanned by 1 and X, and R be the radical of A_{CW} (the subspace spanned by x_i and x_1^2). Denote by ρ the projection of A_{CW} onto R along 1. Then T_{cw} is equivalent to the bilinear map $\varphi_{cw} \in M^* \otimes M^* \otimes R$ defined as $\varphi_{cw}(a,b) = \rho(ab)$ (the multiplication is in A_{CW}).

▶ **Lemma 4.3.** *Let $q \geq 2$. For any $\psi \in U^* \otimes V^* \otimes W$, we have $m(\varphi_{cw} \otimes \psi) \geq 2m(\psi)$.*

Proof. For a bilinear map ψ, the value $m(\psi)$ is the minimum dimension of the space $\psi(u,V)$ among all nonzero $u \in U$.

Consider a nonzero element $a = 1 \otimes u_0 + \sum_{i=1}^{q} x_i \otimes u_i \in M \otimes U$. If all $u_i = 0$, then

$$(\varphi_{cw} \otimes \psi)(a, M \otimes V) = (\varphi_{cw} \otimes \psi)(1 \otimes u_0, M \otimes V) = \varphi_{cw}(1, M) \otimes \psi(u_0, V) = X \otimes \psi(u_0, V)$$

has dimension at least $qm(\psi)$. Otherwise, without loss of generality assume $u_1 \neq 0$. The space $(\varphi_{cw} \otimes \psi)(a, M \otimes V)$ contains subspaces

$$S_0 = (\varphi_{cw} \otimes \psi)(a, 1 \otimes V) = \{\sum_{i=1}^{q} x_i \otimes \psi(u_i, v) \mid v \in V\}$$
$$S_1 = (\varphi_{cw} \otimes \psi)(a, x_1 \otimes V) = \{x_1 \otimes \psi(u_0, v) + x_1^2 \otimes \psi(u_1, v) \mid v \in V\}$$

which have at least $2m(\psi)$ linearly independent elements, namely, for each of at least $m(\psi)$ linearly independent vectors $z_k \in \psi(u_1, V)$ we have $x_1 \otimes z_k + x_2 \otimes w_2 + \cdots + x_q \otimes w_q \in S_0$ and $x_1^2 \otimes z_k + x_1 \otimes w_1 \in S_1$ for some $w_1, w_2, \ldots, w_q \in W$.

In both cases we have $\dim(\varphi_{cw} \otimes \psi)(a, M \otimes V) \geq 2m(\psi)$ for all $a \in M \otimes U$. ◀

▶ **Corollary 4.4.** $\underline{R}(T_{cw}^{\otimes n}) \geqslant (q+1)^n + 2^n - 1$.

Proof. Use the previous Lemma to show that $m(T_{cw}^{\otimes n}) = m(\varphi_{cw}^{\otimes n}) = 2^n$ and Corollary 4.2. ◀

If $\lim_{n\to\infty}(\underline{R}(T_{cw}^{\otimes n}))^{1/n} = q+1$, then the exponent of matrix multiplication would be 2. While the bound above is nontrivial, it is yet not strong enough to rule this out.

Acknowledgements. The authors thank Dmitry Chistikov and anonymous referees for helpful comments and Charilaos Zisopoulos for proofreading.

References

1 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic Complexity Theory*, volume 315 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, Berlin, 1997. `doi:10.1007/978-3-662-03338-8`.

2 Dustin Cartwright, Daniel Erman, Mauricio Velasco, and Bianca Viray. Hilbert schemes of 8 points. *Algebra & Number Theory*, 3(7):763–795, 2009. `doi:10.2140/ant.2009.3.763`.

3 Pierre Comon. Tensor decompositions: State of the art and applications. In *Mathematics in Signal Processing V*, pages 1–24. Oxford University Press, 2002. `arXiv:0905.0454`.

4 Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comp.*, 9(3):251–280, 1990. `doi:10.1016/S0747-7171(08)80013-2`.

5 Hans F. de Groote. On varieties of optimal algorithms for the computation of bilinear mappings i. the isotropy group of a bilinear mapping. *Theor. Comp. Sci.*, 7(1):1–24, 1978. `doi:10.1016/0304-3975(78)90038-5`.

6 Daniel Erman and Mauricio Velasco. A syzygetic approach to the smoothability of zero-dimensional schemes. *Adv. Math.*, 224(3):1143–1166, 2010. `doi:10.1016/j.aim.2010.01.009`.

7 Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51(3):455–500, 2009. `doi:10.1137/07070111X`.

8 Hanspeter Kraft. Geometric methods in representation theory. In *Representations of algebras*, pages 180–258. Springer, 1982. `doi:10.1007/BFb0094059`.

9 Joseph M. Landsberg. *Tensors: Geometry and Applications*, volume 128 of *Graduate Studies in Mathematics*. AMS, Providence, 2012. `doi:10.1090/gsm/128`.

10 Joseph M. Landsberg and Mateusz Michałek. Abelian tensors. Preprint, ArXiv, 2015. `arXiv:1504.03732`.

11 Joseph M. Landsberg and Mateusz Michałek. On the geometry of border rank algorithms for matrix multiplication and other tensors with symmetry. Preprint, ArXiv, 2016. `arXiv:1601.08229`.

12 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014. `doi:10.1145/2608628.2608664`.

13 James S. Milne. Algebraic geometry (v6.01), 2015. URL: `http://www.jmilne.org/math/CourseNotes/ag.html`.

14 Bjorn Poonen. The moduli space of commutative algebras of finite rank. *J. Eur. Math. Soc.*, 10(3):817–836, 2008. `doi:10.4171/JEMS/131`.

15 Volker Strassen. Relative bilinear complexity and matrix multiplication. *J. Reine Angew. Math.*, 375/376:406–443, 1987. `doi:10.1515/crll.1987.375-376.406`.

Using Contracted Solution Graphs for Solving Reconfiguration Problems

Paul Bonsma[*1] **and Daniël Paulusma**[†2]

1 **University of Twente, Enschede, The Netherlands,** `p.s.bonsma@ewi.utwente.nl`
2 **Durham University, UK,** `daniel.paulusma@durham.ac.uk`

Abstract

We introduce a dynamic programming method for solving reconfiguration problems, based on *contracted solution graphs*, which are obtained from solution graphs by performing an appropriate series of edge contractions that decrease the graph size without losing any critical information needed to solve the reconfiguration problem under consideration. As an example, we consider a well-studied problem: given two k-colorings α and β of a graph G, can α be modified into β by recoloring one vertex of G at a time, while maintaining a k-coloring throughout? By applying our method in combination with a thorough exploitation of the graph structure we obtain a polynomial-time algorithm for $(k-2)$-connected chordal graphs.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases reconfiguration, contraction, dynamic programming, graph coloring

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.20

1 Introduction

Given a search problem we may want to find out if one solution for a particular instance is "close" to another solution of that instance to get more insight into the solution space of the problem. Studying the solution space from this perspective could, for instance, be potentially interesting for improving the performance of corresponding heuristics [16]. Searching the solution space by making small "feasible" moves also turned out to be useful when analyzing randomized algorithms for sampling and counting k-colorings of a graph or when analyzing cases of Glauber dynamics in statistical physics (see Section 5 of the survey [19]).

In most general terms, the above situation can be modeled with solution graphs. We formalize this as follows: A *solution graph concept* $\mathcal{S}$ is obtained by defining a set of *instances*, *solutions* for these instances, and a (symmetric) *adjacency relation* between pairs of solutions. For every instance G of the problem, this gives a *solution graph* $\mathcal{S}(G)$, also called a *reconfiguration graph*, which has as node set all solutions of G, with edges as defined by the given adjacency relation. (If G has no solutions then $\mathcal{S}(G)$ is the empty graph.) The adjacency relation usually represents a smallest possible change (or *reconfiguration move*) between two solutions of the same instance. For example, the well-known *k-Color Graph concept* $\mathcal{C}_k$, related to the k-Colorability search problem, is defined as follows: instances are graphs G, and solutions are (proper) k-colorings of G. Two colorings are adjacent if and only if they differ in exactly one vertex. Note however that in general there may be more than one natural way to define the adjacency relation.

* Supported by the European Community's 7^{th} Framework Programme (FP7/2007-2013), grant agreement n° 317662.
† Supported by EPSRC Grant EP/K025090/1.

Solution graphs and their properties have been studied very intensively over the last couple of years for a variety of search problems, which include amongst others k-Coloring [3, 4, 8, 11, 12, 13, 15, 25], Satisfiability [16, 32], Independent Set [7, 9, 26], Shortest Path [5, 6, 26], List Coloring [18], List Edge Coloring [22, 23], $L(2,1)$-Labeling [24], H-Coloring [35] and Subset Sum [20]; see also the aforementioned survey [19]. The study of such solution graphs is commonly called *reconfiguration.*

Reconfiguration problems. Both algorithmic and combinatorial questions have been considered in the fast-growing area of reconfiguration. For instance, what is the diameter of $\mathcal{S}(G)$ (in terms of the size of the instance G) or if $\mathcal{S}(G)$ is not connected, what is the diameter of its (connected) components? In particular, is the diameter always polynomially bounded or not? This led to the introduction of the $\mathcal{S}$-Connectivity problem, which is that of deciding whether the solution graph $\mathcal{S}(G)$ of a given instance G is connected. Refining this problem leads to the following problem:

$\mathcal{S}$-Reachability.
Instance: an instance G with two solutions α and β.
Question: is there a path from α to β in $\mathcal{S}(G)$?

The $\mathcal{S}$-Reachability problem is a central problem in the area of reconfiguration, which has received much attention in the literature. The problem is sometimes called the *α-β-path* problem for $\mathcal{S}$ [19], whereas the specific case of $\mathcal{C}_k$-Reachability is also known as the k-Color Path problem [13]. It is known that $\mathcal{S}$-Reachability is PSPACE-complete for most of the aforementioned solution graph concepts even for special graph classes [8, 17, 21, 29, 34, 36]. For instance, $\mathcal{C}_k$-Reachability is PSPACE-complete even if $k = 4$ and instances are restricted to planar bipartite graphs [8]. This explains that efficient algorithms are only known for very restricted classes of instances. Hence, there is still a need for developing general algorithmic techniques for solving these problems in practice, and for sharpening the boundary between tractable and computationally hard instance classes. Our paper can be seen as the next step in these directions.

Method. One important algorithmic technique is *dynamic programming* (DP). In the area of reconfiguration, there are only relatively few successful examples of nontrivial DP algorithms (such as [5, 7, 18, 29]). In this paper, we focus on a DP technique based on the concept of *contracted solution graphs.* This method was first used by Bonsma [5] to obtain an efficient algorithm for a Shortest-Path-Reachability problem restricted to planar graphs. Recently, Hatanaka, Ito and Zhou [18] used this technique for proving that List-Coloring-Reachability is polynomial-time solvable for caterpillars. We will:

1. generalize the ideas of [5, 18] to a unified dynamic programming method,
2. introduce this method in a broader setting,
3. provide useful notation, terminology and basic lemmas, and
4. illustrate the method by giving a new application.

In Section 2 we give a detailed description of the general method of contracted solution graphs. Informally speaking, in dynamic programming one first computes the required information for parts of the instance, and combines/propagates this to compute the same information for ever larger parts of the instance, until the desired information is known for the entire instance. In our case, the instance G can be any relational structure on a ground set, such as (directed) graphs, hypergraphs, satisfiability formulas, or constraint satisfaction problems in general (see e.g. [10]). The order in which the information can be computed or in which parts should be considered is given by a *decomposition* of G. The elements of the ground set that are in a processed part H and that have incidences with the unexplored part are called *terminals*. The key idea behind the method is that reconfiguration moves in the processed part H that do not involve terminals are often irrelevant. The information that is relevant is captured by the notion of a *terminal projection*. These projections assign labels to solutions, yielding so-called *label components*, which are maximally connected subgraphs of $\mathcal{S}(H)$ induced by sets of solutions that all have the same label. A contracted solution graph is obtained from $\mathcal{S}(H)$ by contracting the label components into single vertices. We stress that the general method can readily be applied to *any kind of relational structure*, but in our example we focus on graphs, just as [5] and [18].

Relation to Other Results. In [18] dynamic programming was done over a *path decomposition* of the given caterpillar. In [5], a layer-based decomposition of the graph was used (for every $i \in \mathbb{N}$, the subgraph H_i consisted of all vertices at distance at most i of the given shortest path starting from a vertex s), which can also be viewed as a path decomposition. Here we focus on the more general *tree decompositions* instead. For our application, we give *full* dynamic programming rules for the $\mathcal{C}_k$-Reachability problem. In particular we introduce a join rule and we allow bags of size larger than 2. Our rules can be used directly for List-Coloring-Reachability as well and thus generalize the rules of [18].

Many well-studied $\mathcal{S}$-Reachability problems (including $\mathcal{C}_k$-Reachability for an appropriate constant k) are PSPACE-complete already for graphs of bounded bandwidth [29, 34], and therefore also for graphs of bounded treewidth. Recently, the PSPACE-completeness results from [29, 34] were strengthened to hold even for planar graphs of bounded bandwidth and low maximum degree [36]. Hence we cannot hope to obtain polynomial-time algorithms for graphs of treewidth w, for every constant w, and certainly not fixed parameter tractable (FPT) algorithms parameterized by w, although such results are common when working with decision problems that are only NP-complete instead of PSPACE-complete.

One way to cope with the above problem is to restrict the problem even further. For instance, in a number of recent papers [10, 25, 26, 32, 30, 31] the *length-bounded* version of the $\mathcal{S}$-Reachability problem was studied, that is the problem of finding a path of length at most ℓ in the solution graph between two given solutions, in particular with an aim to determine fixed-parameter tractability (observe that the length of a path between two solutions is a natural parameter). For instance, although $\mathcal{C}_k$-Reachability is PSPACE-complete for $k \geq 4$, the length-bounded version is FPT when parameterized by the length ℓ [10, 25] (in addition, it is polynomial-time solvable for $k \leq 3$ [25]). In this restricted context, other dynamic programming algorithms over tree decompositions for reconfiguration problems are known: in [29] FPT algorithms are given for various length-bounded reachability problems, parameterized by both the treewidth and the length bound ℓ. In [28], FPT algorithms are given for the reachability versions of different token reconfiguration problems for graphs of bounded degeneracy (and thus for bounded treewidth), when parameterized by the number of tokens.

Since we wish to solve $\mathcal{S}$-Reachability problems in general, we choose a different approach, and present a generally applicable method. However, because of the aforementioned PSPACE-completeness, we can obviously not guarantee that it terminates in polynomial time for all instances. Nevertheless, one can identify restricted instance classes for which it does yield polynomial-time algorithms, as illustrated by our new application and the two other examples [5, 18]. Moreover, our initial computational studies indicate that this method, with a few additions, performs well in practice for various instances of reconfiguration problems, for which the theoretical complexity status is not yet resolved.

Our Application. In Section 3 we illustrate the method by giving dynamic programming rules for the $\mathcal{C}_k$-Reachability problem, which describe how to compute new (larger) contracted solution graphs from smaller ones. Recall that similar dynamic programming rules can be given for other reconfiguration problems, as done already in [5, 18]. The given rules can be used when a tree decomposition of the graph is given. We emphasize that the rules solve the $\mathcal{C}_k$-Reachability problem correctly for *every* graph G (see e.g. [1, 27] for information on finding tree decompositions). Nevertheless, the algorithm is only *efficient* when the contracted solution graphs stay small enough (that is, polynomially bounded). As indicated by the PSPACE-hardness of the problem, this is not always the case. In the same section, we illustrate the DP rules and show that the size of the contracted solution graphs can grow exponentially, even for 2-connected 4-colorable unit interval graphs.

In Section 4 we apply our method to show that, for all $k \geq 3$, $\mathcal{C}_k$-Reachability is polynomially solvable for $(k-2)$-connected chordal graphs. As unit interval graphs are chordal, the result from Section 3 implies that we need to exploit the structure of chordal graphs to prove this. This is not surprising: although $\mathcal{C}_3$-Reachability can be solved in polynomial time for all graphs [13], $\mathcal{C}_k$-Reachability is PSPACE-complete even for bipartite graphs, and if $k \in \{4, 5, 6\}$ for planar graphs, and if $k = 4$ for planar bipartite graphs [8]. As the proof for the PSPACE-completeness result for bipartite graphs from [8] can be easily modified to hold for $(k-2)$-connected bipartite graphs, our result for $(k-2)$-connected chordal graphs cannot be extended to $(k-2)$-connected perfect graphs. On the positive side, $\mathcal{C}_k$-Connectivity is polynomial-time solvable on chordal graphs. This is due to a more general result of Bonamy et al. [4], which implies that for a chordal graph G, $\mathcal{C}_k(G)$ is connected if and only if G has no clique on more than $k-1$ vertices. Hence, our result can be seen as an extension of this result if in addition $(k-2)$-connectivity is imposed. Our result on $\mathcal{C}_k$-Reachability on $(k-2)$-connected chordal graphs is also the first time that dynamic programming over tree decompositions is used to solve the general version of a PSPACE-complete reachability problem in polynomial time for a graph class strictly broader than trees. In Section 5 we discuss possible directions for future work.

Preliminaries. For a connected graph G, a *vertex cut* is a set $S \subseteq V(G)$ such that $G - S$ is disconnected. Vertices in different components of $G - S$ are *separated* by S. For $k \geq 1$, a (connected) graph G is *k-connected* if $|V(G)| \geq k+1$ and every vertex cut S has $|S| \geq k$. The *contraction* of an edge uv of a graph G replaces u and v by a new vertex made adjacent to precisely those vertices that were adjacent to u or v in G (this does not create any multi-edges or loops). A graph is *chordal* if it has no induced cycle of length greater than 3.

Let G be a graph. A *k-color assignment* of G is a function $\alpha : V(G) \to \{1, \ldots, k\}$. For $v \in V(G)$, $\alpha(v)$ is called the *color* of v. It is a *k-coloring* if $\alpha(u) \neq \alpha(v)$ for every edge $uv \in E(G)$. A *coloring* of G is a k-coloring for some value of k. If α and β are colorings of G and a subgraph H of G, respectively, such that $\alpha|_{V(H)} = \beta$ (that is, α and β coincide on $V(H)$) then α and β are said to be *compatible.* For an integer k, the *k-color graph $\mathcal{C}_k(G)$* has as nodes all (proper) k-colorings of G, such that two colorings are adjacent if and only if they differ on one vertex. A *walk* from u to v in G is a sequence of vertices $v_0, \ldots, v_k$ with $u = v_0$, $v = v_k$, such that for all $i < k$, $v_iv_{i+1} \in E(G)$. A *pseudowalk* from u to v is a sequence of vertices $v_0, \ldots, v_k$ with $u = v_0$, $v = v_k$, such that for all $i < k$, either $v_i = v_{i+1}$, or $v_iv_{i+1} \in E(G)$. A *recoloring sequence* from a k-coloring α of G to a k-coloring β of G is a pseudowalk from α to β in $\mathcal{C}_k(G)$. A *labeled graph* is a pair G, ℓ where $G = (V, E)$ is a graph and $\ell : V \to X$ is a vertex labeling (which may assign the same label to different vertices). A *label preserving isomorphism* between two labeled graphs G_1, ℓ_1 and G_2, ℓ_2 is an isomorphism $\phi : V(G_1) \to V(G_2)$, such that $\ell_1(v) = \ell_2(\phi(v))$ for all $v \in V(G_1)$. Informally, two labeled graphs G_1, ℓ_1 and G_2, ℓ_2 are the same if there exists a label preserving isomorphism between them.

2 The Method of Contracted Solution Graphs

In this section we define the concept of *contracted solution graphs* (CSGs) for reconfiguration problems in general. Consider a solution graph concept $\mathcal{S}$, which for every instance G of $\mathcal{S}$ defines a solution graph that is denoted by $\mathcal{S}(G)$. A *terminal projection* for $\mathcal{S}$ is a function p that assigns a *label* to each tuple (G, T, γ) consisting of an instance G of $\mathcal{S}$, a set T of *terminals* for G and a solution γ for G. Terminal projections are used to decide which nodes are "equivalent" and can be contracted. We remark that G and T can be anything, but in our example and in previous examples in the literature [5, 18] G is always a graph, and T is a subset of its vertices. We also note that a terminal projection p can be seen as a node labeling for the solution graph $\mathcal{S}(G)$. So, for every instance G of $\mathcal{S}$, every choice of terminals T may give a different node labeling for the solution graph $\mathcal{S}(G)$. When G and T are clear from the context, we may write $p(\gamma)$ to denote the label of a node γ of $\mathcal{S}(G)$.

Example. Consider the k-color graph concept $\mathcal{C}_k$. Let G be a graph. We can define a terminal projection p as follows. Let T be a subset of $V(G)$. The nodes of $\mathcal{C}_k(G)$ are k-colorings and we give each node as label its restriction to T, that is, for every k-coloring γ of G, we set $p(\gamma) = p(G, T, \gamma) = \gamma|_T$. Note that $\gamma|_T$ is a k-coloring of $G[T]$.

Let p be a terminal projection for a solution graph concept $\mathcal{S}$. For an instance G of $\mathcal{S}$ and a terminal set T, a *label component* C of $\mathcal{S}(G)$ is a maximal set of nodes γ that all have the same label $p(\gamma)$ and that induce a connected subgraph of $\mathcal{S}(G)$. It is easy to see that every solution γ of G is part of exactly one label component, or in other words: the label components partition the node set of $\mathcal{S}(G)$. The *contracted solution graph (CSG)* $\mathcal{S}^c(G, T)$ is a labeled graph that has a node set that corresponds bijectively to the set of label components of G. For a node x of $\mathcal{S}^c(G, T)$, we denote by S_x the corresponding label component. Two distinct nodes x_1 and x_2 of $\mathcal{S}^c(G, T)$ are adjacent if and only if there exist solutions $\gamma_1 \in S_{x_1}$ and $\gamma_2 \in S_{x_2}$ such that γ_1 and γ_2 are adjacent in $\mathcal{S}(G)$. We define a label function ℓ^* for nodes of $\mathcal{S}^c(G, T)$ to denote the corresponding label in $\mathcal{S}(G)$. More precisely: for a node x of $\mathcal{S}^c(G, T)$, the label $\ell^*(x)$ is chosen such that $\ell^*(x) = p(\gamma)$ for all $\gamma \in S_x$. Note that the contracted solution graph $\mathcal{S}^c(G, T)$ can also be obtained from $\mathcal{S}(G)$ by contracting all label components into single nodes and choosing node labels appropriately.

Example. Figure 1(c) shows one component of $\mathcal{C}_4(G)$ for the (4-colorable) graph G from Figure 1(a). This is the component that contains all colorings of G whose vertices a, b, c, d are colored with colors $4, 3, 2, 1$, respectively (note that it is not possible to recolor any of these four vertices if one may recolor only one vertex at a time). So in Figure 1(c) the colors of the vertices a, b, c, d are omitted in the node labels, which only indicate the colors of e, f, g, in this order. For terminal set $T = \{f\}$, this component contains three label components (of equal size), and contracting them yields the CSG $\mathcal{C}_4^c(G, \{f\})$ shown in Figure 1(d). For $T = \{g\}$, there are seven label components, and the corresponding CSG $\mathcal{C}_4^c(G, \{g\})$ is shown in Figure 1(e). Note that $\mathcal{C}_4^c(G, \{g\})$ contains different nodes with the same label.

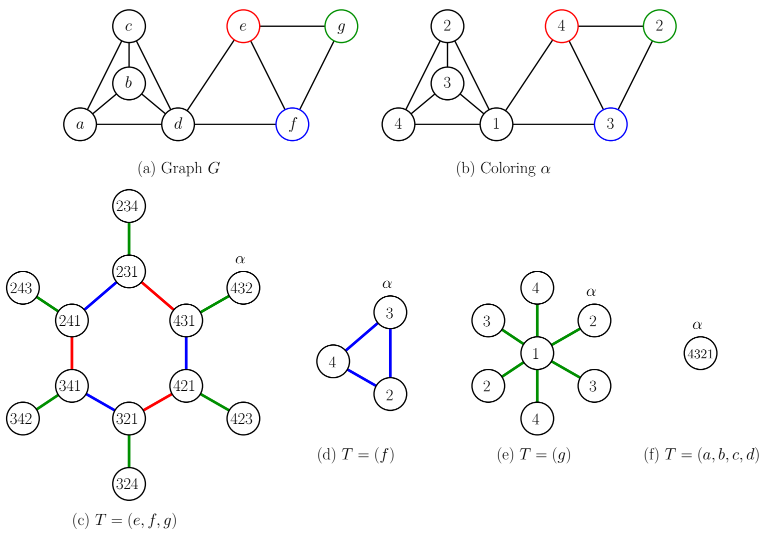


Figure 1 (a) A 4-colorable chordal graph G with $V(G) = \{a, b, c, d, e, f, g\}$. (b) a 4-coloring α, and one component of the CSGs of G for four different terminal sets T: (c) $\mathcal{C}_4^c(G, \{e, f, g\})$, (d) $\mathcal{C}_4^c(G, \{f\})$, (e) $\mathcal{C}_4^c(G, \{g\})$ and (f) $\mathcal{C}_4^c(G, \{a, b, c, d\})$. The $G[T]$-colorings in the node labels are given as sequences of colors, for the (ordered version of) T as indicated below each CSG. Example (c) can also be seen as the component of $\mathcal{C}_4(G)$ where vertices a, b, c, d receive colors $4, 3, 2, 1$.

We stress that the CSG $\mathcal{S}^c(G, T)$ is a labeled graph that includes the label function ℓ^* defined above. However, to keep its size reasonable, the CSG itself does not include the solution sets S_x for each node that were used to define it. For proving the correctness of dynamic programming rules for CSGs the following alternative characterization of CSGs (proof omitted) is useful; note that the sets S_x correspond exactly to the label components.

▶ **Lemma 1.** *Consider an instance G of a solution graph concept $\mathcal{S}$, terminal set T and terminal projection p. Let H, ℓ be a labeled graph. Then $H, \ell = \mathcal{S}^c(G, T)$ if and only if one can define nonempty sets of solutions S_x for each node $x \in V(H)$ such that the following properties hold:*

1. *$\{S_x \mid x \in V(H)\}$ is a partition of the nodes of $\mathcal{S}(G)$ (the solutions of G).*
2. *For every $x \in V(H)$ and every solution $\gamma \in S_x$: $p(G, T, \gamma) = \ell(x)$.*
3. *For every edge $xy \in E(H)$: $\ell(x) \neq \ell(y)$.*
4. *For every $x \in V(H)$: S_x induces a connected subgraph of $\mathcal{S}(G)$.*
5. *For every pair of distinct nodes $x, y \in V(H)$: $xy \in E(H)$ if and only if there exist solutions $\alpha \in S_x$ and $\beta \in S_y$ such that α and β are adjacent in $\mathcal{S}(G)$.*

A mapping S that assigns solution sets (or label components) S_x to each node x of $\mathcal{S}^c(G, T)$ that satisfies the properties given in Lemma 1 is called a *certificate* for $\mathcal{S}^c(G, T)$. Given such a certificate S and a solution γ for G, the *γ-node* of $\mathcal{S}^c(G, T)$ with respect to S is the node x with $\gamma \in S_x$. For readability, we will not always explicitly mention this certificate when talking about γ-nodes in $\mathcal{S}^c(G, T)$ (except in Lemma 2 below), but the reader should keep the following convention in mind: when γ-nodes are identified in $\mathcal{S}^c(G, T)$ for multiple solutions γ, *these are all chosen with respect to the same certificate.*

Example. In Figures 1(c)–(f), the α-node for the coloring α shown in Figure 1(b) is marked. In particular consider $\mathcal{C}_4^c(G, \{g\})$ in Figure 1(e). Since the certificate for $\mathcal{C}_4^c(G, \{g\})$ is not actually indicated in the figure, the other leaf with label 2 can also be chosen as the α-node (considering the nontrivial label-preserving automorphisms of the graph). Similarly, if we choose a coloring β that coincides with α except on nodes e and f, where we choose $\beta(e) = 3$ and $\beta(f) = 4$, then the same two leaves (the ones with label 2) of $\mathcal{C}_4^c(G, \{g\})$ can be chosen as the β-node. Nevertheless, if both an α-node and β-node are marked, then this will only be correct according to the above convention when they are distinct!

The main purpose of our definitions is the following key observation (we omit its proof).

▶ **Lemma 2.** *Let (G, T) be an instance of a solution graph concept $\mathcal{S}$. Let $\mathcal{S}^c(G, T)$ be the contracted solution graph for some terminal projection p. Let α and β be two solutions and let x and y be the α-node resp. β-node with respect to some certificate S. Then there is a path from α to β in $\mathcal{S}(G)$ if and only if there is a path from x to y in $\mathcal{S}^c(G, T)$.*

Lemma 2 implies that for a solution graph concept $\mathcal{S}$ and *any* terminal projection p and terminal set T, we can decide $\mathcal{S}$-Connectivity if we know $\mathcal{S}^c(G, T)$ (the answer is YES if and only if $\mathcal{S}^c(G, T)$ is connected) and the $\mathcal{S}$-Reachability problem if we know $\mathcal{S}^c(G, T)$ and the α-node and the β-node (the answer is YES if and only if these two nodes are in the same component). However, for obtaining an *efficient* algorithm using this strategy, we must choose the terminal projection p smartly: we need to throw away enough irrelevant information to ensure that $\mathcal{S}^c(G, T)$ will be significantly smaller than $\mathcal{S}(G)$, yet maintain enough information to ensure the efficient computation of $\mathcal{S}^c(G, T)$, without first constructing $\mathcal{S}(G)$. Our strategy for doing this is to use dynamic programming to compute $\mathcal{S}^c(H, T')$ for ever larger subgraphs H of G, while ensuring that all of the CSGs stay small throughout the process. The remainder of this paper shows a successful example of this strategy.

3 Dynamic Programming Rules for Recoloring

The following terminology is based on widely used techniques for dynamic programming over tree decompositions; see Section 4 and [2, 27, 33] for background information. A *terminal graph* (G, T) is a graph G together with a vertex set $T \subseteq V(G)$, whose vertices are called the *terminals*. If $T = V(G)$, then (G, T) is called a *leaf*. If $v \in T$, then we say that the new terminal graph $(G, T \setminus \{v\})$ is obtained from (G, T) by *forgetting* v (or *using a forget operation*). If $T \neq V(G)$, $v \in T$ and $N(v) \subseteq T$ then we say that (G, T) can be obtained from $(G - v, T \setminus \{v\})$ by *introducing* v (or *using an introduce operation*). Note that for a terminal graph (G', T') with $T' \neq \emptyset$, different graphs can be obtained from (G', T') by introducing a vertex v, whereas forgetting a terminal always yields a unique result. As we will see, the condition that each neighbor of the new vertex v must be in T is necessary. We say that (G, T) is the *join of* (G_1, T) *and* (G_2, T) (or *can be constructed using a join operation*) if

- G_1 and G_2 are induced subgraphs of G,
- $V(G_1) \cap V(G_2) = T$ and $V(G_1) \cup V(G_2) = V(G)$,
- $V(G_1) \neq T$ and $V(G_2) \neq T$, and
- for every $uv \in E(G)$, it holds that $uv \in E(G_1)$ or $uv \in E(G_2)$.

We will now focus on CSGs for the k-color graph concept $\mathcal{C}_k$, using the terminal projection $p(G, T, \gamma) = \gamma|_T$. We will show how to compute the CSG $\mathcal{C}_k^c(G, T)$ when (G, T) is obtained using a forget, introduce or join operation from a (pair of) graph(s) for which we know the CSG(s). We recall that a variant of these CSGs have been considered before by Hatanaka, Ito and Zhou [18], namely for the case that $|T| = 1$ in the context of list colorings of caterpillars. Similar dynamic programming rules were given in [18]: for the case that $|T| = 1$, they presented a combined introduce and forget rule, and a restricted type of join rule.

We start by stating a trivial rule for computing $\mathcal{C}_k^c(G,T)$ for leaves, which follows from the facts that $\mathcal{C}_k(G)$ has k-colorings of G as nodes and that the label $\ell(x)$ of a node x in $\mathcal{C}_k^c(G,T)$ is a k-coloring of $G[T]$.

▶ **Lemma 3** (Leaf). *Let (G,T) be a terminal graph with $T = V(G)$. Then $\mathcal{C}_k^c(G,T)$ is isomorphic to $\mathcal{C}_k(G)$ and its label function ℓ is the isomorphism from $\mathcal{C}_k^c(G,T)$ to $\mathcal{C}_k(G)$. Moreover, for every k-coloring γ of G, the γ-node of $\mathcal{C}_k^c(G,T)$ is the node v with $\ell(v) = \gamma$.*

We now give the rules for the forget, introduce and join operations. Figure 2 illustrates the first two rules. We show how Lemma 1 can be applied to prove Lemma 4; the other proofs are similar.

▶ **Lemma 4** (Forget). *Let (G,T) be a terminal graph. For every $v \in T$, it holds that $H', \ell' = \mathcal{C}_k^c(G, T \setminus \{v\})$ can be computed from $H, \ell = \mathcal{C}_k^c(G,T)$ as follows:*

- *For every node x in H with $\ell(x) = \gamma$, let $\ell'(x) = \gamma|_{T\setminus\{v\}}$.*
- *Iteratively contract every edge between two nodes x and y with $\ell'(x) = \ell'(y)$ and assign label $\ell'(z) := \ell'(x)$ to the resulting node z.*

Moreover, for any coloring γ of G, the γ-node of $\mathcal{C}_k^c(G, T \setminus \{v\})$ is the node that results from contracting the set of nodes that includes the γ-node of $\mathcal{C}_k^c(G,T)$.

Proof sketch: Let S denote the certificate for H, ℓ, so for every node x of H, S_x denotes the set of k-colorings of G (or *solutions*), such that these sets satisfy the properties stated in Lemma 1. In addition, for every coloring γ for which a γ-node x has been marked in H, we may assume that $\gamma \in S_x$. We will prove the statement using Lemma 1 again, by giving a certificate S' for H', ℓ', and proving that the five properties hold for these.

The graph H' is obtained by iteratively contracting edges of H, so every node y of H' corresponds to a connected set of nodes of H, which we will denote by M_y. So $\{M_y \mid y \in V(H')\}$ is a partition of $V(H)$. For every node $y \in V(H')$, we define $S'_y = \cup_{x \in M_y} S_x$. For every k-coloring γ of G such that the γ-node $x \in V(H)$ is marked, we define the γ-node of H' to be the node y with $x \in M_y$. Clearly, $\gamma \in S'_y$ then holds, so this is correct. One can now verify that the solution sets S'_x satisfy the five properties stated in Lemma 1. ◀

▶ **Lemma 5** (Introduce). *Let (G,T) be a terminal graph obtained from a terminal graph $(G - v, T \setminus \{v\})$ by introducing v. Then $H', \ell' = C_k^c(G,T)$ can be computed as follows from $H, \ell = \mathcal{C}_k^c(G - v, T \setminus \{v\})$:*

- *For every node x of H with label $\ell(x)$, and every color $c \in \{1, \ldots, k\}$: if the (unique) function $\delta : T \to \{1, \ldots, k\}$ with $\delta(v) = c$ and $\delta|_T = \ell(x)$ is a coloring of $G[T]$ then introduce a node x_c with label $\ell'(x_c) = \delta$.*
- *For every pair of distinct nodes x_c and y_d: add an edge between them if and only if (1) $x = y$ or (2) xy is an edge in H and $c = d$.*

Moreover, for every k-coloring γ of G, if x is the $\gamma|_{V(G)\setminus\{v\}}$-node in H and $\gamma(v) = c$, then x_c is the γ-node of H'.

▶ **Lemma 6** (Join). *Let (G,T) be a terminal graph that is the join of terminal graphs (G_1,T) and (G_2,T). Let $H_1, \ell_1 = C_k^c(G_1,T)$ and $H_2, \ell_2 = C_k^c(G_2,T)$. Then $H, \ell = C_k^c(G,T)$ can be computed as follows:*

- *For every pair of nodes $x \in V(H_1)$ and $y \in V(H_2)$: if $\ell_1(x) = \ell_2(y)$ then introduce a node (x,y) with $\ell((x,y)) = \ell_1(x)$.*
- *For two distinct nodes (x,y) and (x',y'), add an edge between them if and only if xx' is an edge in H_1 and yy' is an edge in H_2.*

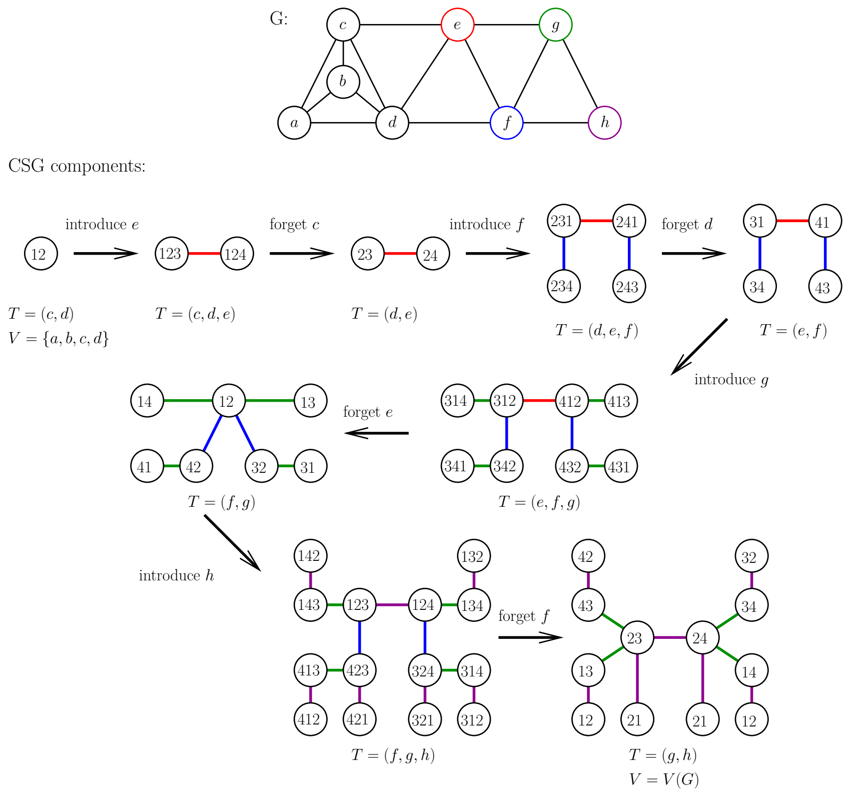


Figure 2 An example of computing CSGs using forget and introduce operations. A 4-colorable 2-connected chordal graph G with $V(G) = \{a, b, c, d, e, f, g, h\}$ is shown. Note that G is in fact unit interval and isomorphic to the graph G_8^I defined in Section 3. Starting with one component of the CSG $\mathcal{C}_4^c(G[\{a, b, c, d\}], \{c, d\})$, the corresponding component of $\mathcal{C}_4^c(G, \{g, h\})$ is computed, using four forget and introduce operations. The $G[T]$-colorings in the node labels are given as sequences of colors for the ordered version of T as indicated below each CSG. For instance, for $T = (c, d)$, the node label 12 indicates the coloring γ with $\gamma(c) = 1$ and $\gamma(d) = 2$.

Moreover, for every k-coloring γ of G, if x is the $\gamma|_{V(G_1)}$-node in H_1 and y is the $\gamma|_{V(G_2)}$-node in H_2, then (x, y) is the γ-node in H.

Remark 1. The DP rules in this section can be generalized further to capture the rules of [18] for the *list coloring* generalization $\mathcal{C}_L$ of $\mathcal{C}_k$. In this generalization, an instance G, L consists of a graph G together with color lists $L(v) \subseteq \{1, \dots, k\}$ for each $v \in V(G)$. Solutions are now list colorings, which are colorings α of G such that $\alpha(v) \in L(v)$ for each $v \in V(G)$. Adjacency is defined as before. So the *list coloring solution graph* $\mathcal{C}_L(G, L)$ is an induced subgraph of $\mathcal{C}_k(G)$. Hence, it is straightforward to generalize our DP rules to $\mathcal{C}_L$, namely by simply omitting all nodes that correspond to invalid vertex colors.

We now show that components of $\mathcal{C}_k^c(G)$ can grow exponentially even if G is chordal and $k = 4$. First, when considering 4-colorable chordal graphs with cut vertices, it is easy to obtain CSGs that have exponentially large components: take p copies of the graph shown in Figure 1(a), and identify the g-vertices of all of these graphs. Call the resulting graph G_p^*. We can show that, for every integer $p \geq 1$, $\mathcal{C}_4^c(G_p^*, \{g\})$ has a component with $1 + 3 \cdot 2^p$ nodes.

We can construct CSGs with exponentially large components for $(k-2)$-connected k-

colorable chordal graphs, or even 2-connected 4-colorable unit interval graphs, as follows. For $p \geq 4$, let the graph G^I_p have vertex set $\{v_0, \ldots, v_{p-1}\}$, and edge set $\{v_0v_3\} \cup \{v_iv_{i+1} \mid 0 \leq i \leq p-2\} \cup \{v_iv_{i+2} \mid 0 \leq i \leq p-3\}$. A graph isomorphic to G^I_8 is shown in Figure 2. Note that each G^I_p is unit interval. To state our claim more precisely, for every $p = 4q + 4$ with $q \in \mathbb{N}$, we can show that the CSG $\mathcal{C}^c_4(G^I_p, \{v_{p-2}, v_{p-1}\})$ has 4! components on at least 2^q nodes.

Both examples show that we need to do more than only computing CSGs to solve the problem for $(k-2)$-connected chordal graphs. Next, we will characterize the CSGs and show that it suffices to compute only a part of them.

4 Recoloring Chordal Graphs

We will show that CSGs can be used to efficiently decide the $\mathcal{C}_k$-Reachability problem for $(k-2)$-connected chordal graphs. To prove this we use the fact that for a chordal graph G and any clique T of G, the terminal graph (G,T) can recursively be constructed from simple cliques using a polynomial number of clique-based introduce, forget and join operations. We remark that some statements given here are similar to (and can alternatively be deduced from) well-known facts about tree decompositions [14] and nice tree decompositions [27]. However, for readability, and since we need to prove a new bound on the size of any tree decomposition, we give a self-contained presentation.

A *nice tree decomposition* of a terminal graph (G,T) (where G is not necessarily chordal and T may not be a clique) is a tuple $(\mathcal{T}, X, r)$, where $\mathcal{T}$ is a tree with root r and X is an assignment of *bags* $X_u \subseteq V(G)$ for each $u \in V(\mathcal{T})$ that can be defined recursively as follows:

1. If $T = V(G)$, then the tree $\mathcal{T}$ consists of one (root) node r with bag $X_r = T$.
2. If $v \in V(G) \setminus T$ and $(\mathcal{T}', X, r')$ is a nice tree decomposition of $(G, T \cup \{v\})$, then a nice tree decomposition for (G,T) can be obtained by adding a new root r with $X_r = T$, and adding the edge rr'.
3. If (G,T) can be obtained from $(G-v, T\setminus\{v\})$ using an introduce operation and $(\mathcal{T}', X, r')$ is a nice tree decomposition of $(G-v, T\setminus\{v\})$, then a nice tree decomposition for (G,T) can be obtained by adding a new root r with $X_r = T$, and adding the edge rr'.
4. If (G,T) can be obtained from (G_1,T) and (G_2,T) using a join operation, and $(\mathcal{T}_1, X, r_1)$ and $(\mathcal{T}_2, X, r_2)$ are nice tree decompositions of (G_1,T) and (G_2,T), then a nice tree decomposition for (G,T) can be obtained by adding a new root r with $X_r = T$ and adding edges rr_1 and rr_2.

We call a node $u \in V(\mathcal{T})$ a *leaf*, *forget node*, *introduce node* or *join node* if u is added as the root in case (1), (2), (3) or (4), respectively. The *width* of $(\mathcal{T}, X, r)$ is $\max_{u \in V(\mathcal{T})} |X_u| - 1$.

▶ **Lemma 7.** (proof omitted) *Let $(\mathcal{T}, X, r)$ be a nice tree decomposition of (G,T) of width at most $w \geq 1$, and let $n = |V(G)| \geq 1$. Then $|V(\mathcal{T})| \leq (w+4)n$.*

The bound from Lemma 7 holds for *any* nice tree decomposition, in contrast to the (stronger) bound of [27] which states that for any graph G, a nice tree decomposition of G of width at most $4n$ can be constructed (for an appropriate choice of the terminal set T). A nice tree decomposition $(\mathcal{T}, X, r)$ of (G,T) is *chordal* if for every node $u \in V(\mathcal{T})$, X_u is a clique of G. If $(\mathcal{T}, X, r)$ is a chordal nice tree decomposition of a k-colorable graph G, then the width of $(\mathcal{T}, X, r)$ is at most $k-1$. Hence, Lemma 7 shows that any chordal nice tree decomposition has at most $(k+3)n$ nodes. This bound is asymptotically sharp.

▶ **Theorem 8.** (proof omitted) *There are k-colorable chordal graphs G for which any chordal nice tree decomposition has at least $\Omega(kn)$ nodes.*

In order to show how to find a chordal nice tree decomposition in polynomial time we need the following lemma (proof omitted), which tells us how to select the proper type of root node when constructing such a tree decomposition. A terminal graph (G_1, T_1) is called *smaller* than another terminal graph (G_2, T_2) if $2|V(G_1)| - |T_1| < 2|V(G_2)| - |T_2|$.

▶ **Lemma 9.** *Let (G, T) be a terminal graph where $G = (V, E)$ is a chordal graph, and T is a clique with $T \neq V$. If $G - T$ is disconnected, then (G, T) can be obtained from a pair of smaller chordal terminal graphs (G_1, T) and (G_2, T) using a join operation. Otherwise, (G, T) can be obtained from a smaller chordal terminal graph (G', T') using either a forget or introduce operation, where T' is again a clique. For every such (G, T), the relevant operation and subgraph(s) can be found in polynomial time.*

By combining Lemmas 7 and 9 we obtain the following result.

▶ **Corollary 10.** *Let G be a chordal k-colorable graph on n vertices, and let T be a clique of G. In polynomial time, we can find a chordal nice tree decomposition of (G, T) on at most $(k + 3)n$ nodes.*

Proof. Lemma 9 shows how we can choose the proper type of root node. We can build the chordal nice tree decomposition by adding this node to the tree decomposition(s) of (a) smaller graph(s). The entire chordal nice tree decomposition is constructed by continuing this process recursively. Lemma 7 shows that the resulting chordal nice tree decomposition has at most $(w + 4)n$ nodes, where $w + 1$ is the maximum bag size. Since every bag is a clique of G and the graph is k-colorable, we have $w + 1 \leq k$, so there are at most $(k + 3)n$ nodes. Since we have a polynomial number of nodes, and for every node we spend polynomial time (Lemma 9), the entire process terminates in polynomial time. ◀

The precise complexity bound in Corollary 10 depends on implementation details beyond the scope of this paper.

Using an inductive proof based on Lemma 9, we will now characterize the shape of CSGs for $(k - 2)$-connected k-colorable chordal graphs. For integers m, k with $1 \leq m \leq k$, a labeled graph H, ℓ is an (m, k)-*color-complete graph* if there exists a set T with $|T| = m$ such that:

- for all vertices $v \in V(H)$, $\ell(v)$ is a k-coloring of a complete graph on vertex set T,
- every such k-coloring of T appears at exactly one vertex of H, and
- two vertices of H are adjacent if and only if their labels differ on exactly one element of T.

From this definition it follows that for every pair of integers m and k, there is a unique (m, k)-color complete graph, up to the choice of T. An (m, k)-color-complete graph has $k!/(k - m)!$ vertices (this is the number of ways to k-color a complete graph on m vertices), and every vertex has degree $m(k - m)$. In particular, if $m = k$ then the graph consists of $k!$ isolated vertices (which is a forest). A labeled graph H, ℓ is said to satisfy the *injective neighborhood property (INP)* if for every vertex $u \in V(H)$ and every pair of distinct neighbors $v, w \in N(u)$, it holds that $\ell(v) \neq \ell(w)$. Note that (m, k)-color-complete graphs trivially satisfy the INP. We prove Theorem 12 by first showing that for our graphs the following invariant (Theorem 11) is maintained by introduce, forget and join operations. This invariant can be proven by induction based on Lemma 9, using the rules from Section 3.

▶ **Theorem 11.** *Let $k \geq 3$. Let $G = (V, E)$ be a $(k - 2)$-connected k-colorable chordal graph, and let $T \subseteq V(G)$ be a clique of G with $m = |T| \geq k - 2$. Then $\mathcal{C}^c_k(G, T)$ is an (m, k)-color-complete graph, or it is a forest that satisfies the injective neighborhood property.*

Remark 2. Figure 1 shows that if we relax the connectivity requirement to $(k-3)$-connectedness, the above property does not necessarily hold anymore: the examples in Figure 1(c) and (d) are not forests, and the example in Figure 1(e) does not satisfy the INP.

The characterization of $\mathcal{C}_k^c(G,T)$ in Theorem 11 does not yet guarantee that simply keeping track of the (relevant component of the) CSG yields a polynomial-time algorithm, as shown by the second example in Section 3. However, we will now show that it suffices to only keep track of the following essential information, which remains polynomially bounded.

Let $G=(V,E)$ be a graph with $T\subseteq V$, and let α and β be k-colorings of a supergraph of G. (G should be viewed as a subgraph that occurs during the dynamic programming, while α and β are the colorings of the full graph.) Let $\alpha'=\alpha|_V$ and $\beta'=\beta|_V$. If $\mathcal{C}_k^c(G,T)$ is a forest with the α'-node x and β'-node y in the same component, then we define the *α-β-path* to be the unique path in $\mathcal{C}_k^c(G,T)$ with end vertices x and y (together with its vertex labels). Given the two colorings α and β, the *essential information* for $\mathcal{C}_k^c(G,T)$ consists of the following:

- whether the α' and β' nodes appear in the same component,
- whether $\mathcal{C}_k^c(G,T)$ is a forest, and
- in case the answers to both questions are yes: the α-β-path in $\mathcal{C}_k^c(G,T)$.

▶ **Theorem 12.** *For a k-colorable $(k-2)$-connected chordal graph G with two k-colorings α and β, we can decide in polynomial time whether $\mathcal{C}_k(G)$ contains an α-β path.*

Proof sketch: Corollary 10 shows that for every chordal k-colorable graph G on n vertices, we can find in polynomial time a chordal nice tree decomposition on at most $(k+3)n$ nodes. So every node of this tree decomposition corresponds to a $(k-2)$-connected chordal subgraph H of G with terminal set T, such that either H is a clique with $T=V(H)$ (leaf nodes), or (H,T) can be obtained from the graph(s) corresponding to its child node(s) using a forget, introduce or join operation. For every one of those terminal subgraphs, we compute the essential information, bottom up. If at any point, the α' and β' nodes are separated, the answer is NO. Forget, introduce and join operations maintain a forest. The lemmas from Section 3 show how the α-β-path can be computed. We return YES if in the root node, a color-complete graph or an α-β-path is obtained (Lemma 2). The total number of operations (tree decomposition nodes) is $O(kn)$. For every operation, the essential information can be computed in polynomial time (in the input size, which includes the path length). One can show that the maximum length of any α-β path that occurs during the computation is $O(kn)$. Hence, the algorithm terminates in polynomial time. ◀

We stress that (m,k)-color complete graphs, which have $k!/(k-m)!$ nodes, are not computed explicitly in our algorithm. So indeed, in order to obtain a polynomial-time algorithm, we do not need to assume that k is a constant.

5 Discussion

An obvious question is whether our polynomial-time algorithm for can be extended to all chordal graphs, or whether $\mathcal{C}_k$-Reachability is PSPACE-hard for chordal graphs. As $\mathcal{C}_3$-Reachability is polynomial-time solvable in general [13], the first open case is the complexity of $\mathcal{C}_4$-Reachability for chordal graphs (with at least one cut vertex). We refer to Remark 2 for a discussion on why our current proof technique does not work for this case. We also note that the complexity of $\mathcal{C}_4$-Reachability is open for proper interval graphs (initial experimental results for these graphs seem to suggest that even this problem is not straightforward to solve). The two most important future research goals are the following.

1. Explore for which other solution graph concepts $\mathcal{S}$ the DP method can be used to obtain polynomial-time algorithms for the $\mathcal{S}$-REACHABILITY problem. The DP method has now been used to obtain polynomial-time algorithms for several reconfiguration problems, but its true strength is not always revealed when using the viewpoint of worst-case algorithm analysis. For instance, when considering randomly generated k-colorable chordal or interval graphs, we observed that the method performs well on most instances, despite the fact that specialized examples can be constructed that exhibit exponential growth. As we noticed when considering other reconfiguration problems, this behavior seems to occur in general. Because of this, we will write a subsequent paper which will include computational studies, where we apply extensions of this method to various other reconfiguration problems such as well-studied variants of independent set reconfiguration problems (see e.g. [9, 26]).

2. Explore which known reconfiguration problems can be solved efficiently using CSGs. The method of using CSGs can easily be applied to solve the $\mathcal{S}$-CONNECTIVITY problem. Hence, this well-studied problem is a suitable candidate problem for the second research goal.

References

1 H.L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

2 H.L. Bodlaender, P. Bonsma, and D. Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In *Proc. IPEC*, volume 8246 of *LNCS*, pages 41–53. Springer, 2013.

3 M. Bonamy and N. Bousquet. Recoloring bounded treewidth graphs. *Electronic Notes in Discrete Mathematics*, 44:257–262, 2013.

4 M. Bonamy, M. Johnson, I.M. Lignos, V. Patel, and D. Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27:132–143, 2014. `doi:10.1007/s10878-012-9490-y`.

5 P. Bonsma. Rerouting shortest paths in planar graphs. In *Proc. FSTTCS 2012*, volume 18 of *LIPIcs*, pages 337–349. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

6 P. Bonsma. The complexity of rerouting shortest paths. *Theoretical Computer Science*, 510:1 – 12, 2013.

7 P. Bonsma. Independent set reconfiguration in cographs and their generalizations. *Journal of Graph Theory*, 2015. `doi:10.1002/jgt.21992`.

8 P. Bonsma and L. 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`.

9 P. Bonsma, M. Kamiński, and M. Wrochna. Reconfiguring independent sets in claw-free graphs. In *Proc. SWAT 2014*, volume 8503 of *LNCS*, pages 86–97. Springer, 2014.

10 P. Bonsma, A.E. Mouawad, N. Nishimura, and V. Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Proc. IPEC 2014*, volume 8894 of *LNCS*, pages 110–121. Springer, 2014.

11 L. Cereceda, J. van den Heuvel, and M. Johnson. Connectedness of the graph of vertex-colourings. *Discrete Mathematics*, 308(5):913–919, 2008.

12 L. Cereceda, J. van den Heuvel, and M. Johnson. Mixing 3-colourings in bipartite graphs. *European Journal of Combinatorics*, 30(7):1593–1606, 2009.

13 L. Cereceda, J. van den Heuvel, and M. Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.

14 R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, fourth edition, 2010.

15 C. Feghali, M. Johnson, and D. Paulusma. A reconfigurations analogue of Brooks' theorem. In *Proc. MFCS 2014*, volume 8635 of *LNCS*, pages 287–298. Springer, 2014.

16 P. Gopalan, P.G. Kolaitis, E. Maneva, and C.H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6), 2009.

17 A. Haddadan, T. Ito, A.E. Mouawad, N. Nishimura, H. Ono, A. Suzuki, and Y. Tebbal. The complexity of dominating set reconfiguration. In *Proc. WADS 2015*, volume 9214 of *LNCS*, pages 398–409. Springer, 2015.

18 T. Hatanaka, T. Ito, and X. Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 98(6):1168–1178, 2015.

19 J. van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.

20 T. Ito and E.D. Demaine. Approximability of the subset sum reconfiguration problem. In *TAMC 2011*, volume 6648 of *LNCS*, pages 58–69. Springer, 2011. `doi:10.1007/978-3-642-20877-5_7`.

21 T. Ito, E.D. Demaine, N.J.A. Harvey, C.H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011.

22 T. Ito, M. Kamiński, and E.D. Demaine. Reconfiguration of list edge-colorings in a graph. *Discrete Applied Mathematics*, 160(15):2199–2207, 2012.

23 T. Ito, K. Kawamura, and X. Zhou. An improved sufficient condition for reconfiguration of list edge-colorings in a tree. *IEICE TRANSACTIONS on Information and Systems*, 95(3):737–745, 2012.

24 Takehiro Ito, Kazuto Kawamura, Hirotaka Ono, and Xiao Zhou. Reconfiguration of list $L(2,1)$-labelings in a graph. *Theoretical Computer Science*, 544:84–97, 2014.

25 M. Johnson, D. Kratsch, S. Kratsch, V. Patel, and D. Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016. URL: `http://dro.dur.ac.uk/15595/`.

26 M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.

27 T. Kloks. *Treewidth: computations and approximations*, volume 842 of *LNCS*. Springer, 1994.

28 D. Lokshtanov, A.E. Mouawad, F. Panolan, M.S. Ramanujan, and S. Saurabh. Reconfiguration on sparse graphs. In *Proc. WADS 2015*, volume 9214 of *LNCS*, pages 506–517. Springer, 2015.

29 A. E. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over tree decompositions. In *Proc. IPEC 2014*, volume 8894 of *LNCS*, pages 246–257. Springer, 2014.

30 A.E. Mouawad, N. Nishimura, and V. Raman. Vertex cover reconfiguration and beyond. In *Proc. ISAAC 2014*, volume 8889 of *LNCS*, pages 452–463. Springer, 2014.

31 A.E. Mouawad, N. Nishimura, V. Raman, N. Simjour, and A. Suzuki. On the parameterized complexity of reconfiguration problems. In *Proc. IPEC 2013*, volume 8246 of *LNCS*, pages 281–294. Springer, 2013.

32 A.E. Mouawad, N. Nishimura, Pathak V., and V. Raman. Shortest reconfiguration paths in the solution space of boolean formulas. In *Proc. ICALP 2015*, volume 9134 of *LNCS*, pages 985–996. Springer, 2015.

33 R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

34 M. Wrochna. Reconfiguration in bounded bandwidth and treedepth. arXiv:1405.0847, 2014. URL: `http://arxiv.org/abs/1405.0847`.

35 M. Wrochna. Homomorphism reconfiguration via homotopy. In *Proc. STACS 2015*, volume 30 of *LIPIcs*, pages 730–742. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

36 T.C. van der Zanden. Parameterized complexity of graph constraint logic. In *Proc. IPEC 2015*, volume 43 of *LIPIcs*, pages 282–293. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Pointer Quantum PCPs and Multi-Prover Games*

Alex B. Grilo[1], Iordanis Kerenidis[2], and Attila Pereszlényi[3]

1 **IRIF, CNRS, Université Paris Diderot, Paris, France**
2 **IRIF, CNRS, Université Paris Diderot, Paris, France and Centre for Quantum Technologies, National University of Singapore, Singapore**
3 **IRIF, CNRS, Université Paris Diderot, Paris, France**

Abstract

The quantum PCP (QPCP) conjecture states that all problems in QMA, the quantum analogue of NP, admit quantum verifiers that only act on a constant number of qubits of a polynomial size quantum proof and have a constant gap between completeness and soundness. Despite an impressive body of work trying to prove or disprove the quantum PCP conjecture, it still remains widely open. The above-mentioned proof verification statement has also been shown equivalent to the QMA-completeness of the Local Hamiltonian problem with constant relative gap. Nevertheless, unlike in the classical case, no equivalent formulation in the language of multi-prover games is known.

In this work, we propose a new type of quantum proof systems, the Pointer QPCP, where a verifier first accesses a classical proof that he can use as a pointer to which qubits from the quantum part of the proof to access. We define the Pointer QPCP conjecture, that states that all problems in QMA admit quantum verifiers that first access a logarithmic number of bits from the classical part of a polynomial size proof, then act on a constant number of qubits from the quantum part of the proof, and have a constant gap between completeness and soundness. We define a new QMA-complete problem, the Set Local Hamiltonian problem, and a new restricted class of quantum multi-prover games, called CRESP games. We use them to provide two other equivalent statements to the Pointer QPCP conjecture: the Set Local Hamiltonian problem with constant relative gap is QMA-complete; and the approximation of the maximum acceptance probability of CRESP games up to a constant additive factor is as hard as QMA. Our new conjecture is weaker than the original QPCP conjecture and hence provides a natural intermediate step towards proving the quantum PCP theorem. Furthermore, this is the first equivalence between a quantum PCP statement and the inapproximability of quantum multi-prover games.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Computational Complexity, Quantum Computation, PCP Theorem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.21

1 Introduction

The celebrated PCP theorem states that all languages in NP can be verified probabilistically by randomized verifiers that only check a constant number of bits of a polynomial size proof [6, 7, 10]. This theorem has far-reaching applications in complexity theory and especially in the inapproximability of certain optimization problems. This is because the PCP theorem can be recast in the following equivalent way: the approximation of MAX-SAT up to some constant additive factor is NP-complete. Let us also remark that the classical PCP

* This work was supported by the European Union (ERC project QCC 306537) and the ANR RDAM project.

theorem has a third very interesting equivalent formulation as approximation of the maximum acceptance probability of some polynomial size multi-prover interactive games [22]. This game formulation was fundamental in order to achieve better constants for the inapproximability results of a number of NP-hard problems.

One of the main questions in quantum complexity theory is whether one can prove an analogous statement for the class QMA, the quantum analogue of NP. The QPCP conjecture [2] has received a lot of attention due to its importance to both physics and theoretical computer science and an impressive body of work has provided either positive evidence [1, 11, 19] or negative [5, 3, 8]. There are many different ingredients that go into the proof of the classical PCP theorem, especially since there are two different ways of proving it, one through the proof system formulation and another more combinatorial way by looking directly at the inapproximability of constraint satisfaction problems. In the quantum setting, the positive and negative evidence has been mostly that certain techniques that had been used in the classical setting are applicable or not in the quantum setting. We note that the lack of a way of seeing the quantum PCP conjecture in a game context also prevents us from using some important techniques that are present in the classical case, such as the parallel repetition theorem. Overall, proving the quantum PCP theorem remains a daunting task.

The QPCP conjecture can be cast as a type of a proof system which we denote by $\mathsf{QPCP}(q, \alpha, \beta)$. Here a quantum verifier tosses a logarithmic number of classical coins and, based on the coin outcomes, decides on which q qubits from the polynomial-size quantum proof to perform a measurement. The measurement output decides on acceptance or rejection. A yes instance is accepted with probability at least α and a no instance is accepted with probability at most β, for some $\alpha > \beta$ [1, 2]. The formal conjecture is stated below.

▶ **Conjecture 1** (QPCP Conjecture – Proof verification version)**.** $\mathsf{QMA} = \mathsf{QPCP}(q, \alpha, \beta)$ *where* $q = O(1)$ *and* $\alpha - \beta = \Omega(1)$*.*

In quantum mechanics, the evolution of quantum systems are described by Hermitian operators called Hamiltonians. In nature, particles that are far apart tend not to interact so the global Hamiltonian can usually be described as a sum of local Hamiltonians. The Local Hamiltonian problem, denoted by $\textsc{LocalHam}(k, a, b)$, receives as input m Hamiltonians $H_1, \ldots, H_m$ where each one has norm at most 1 and describes the evolution of at most k qubits. The question is if there is a global state such that its energy is at most am or all states have energy at least bm for $b > a$. The area studying the above problem is called quantum Hamiltonian complexity [21, 13]. It began with Kitaev who showed that for $b - a \geq 1/\text{poly}(n)$, $\textsc{LocalHam}(5, a, b)$ is complete for the class QMA [4, 18]. It has subsequently been improved, reducing the locality of the Hamiltonians to two [16] and restricting their structure [17, 20, 9, 14]. These results imply that estimating the groundstate energy of a system within an inverse polynomial additive factor is hard. It is natural to ask if it still remains hard if we require only constant approximation. The physical interpretation of this problem is connected to the stability of entanglement in "room temperature".

The second version of the quantum PCP conjecture asks if $\textsc{LocalHam}(k, a, b)$ remains QMA-complete when $b - a$ is constant. It is stated formally in the conjecture below.

▶ **Conjecture 2** (QPCP Conjecture – Constraint satisfaction version)**.** *The Local Hamiltonian problem* $\textsc{LocalHam}(k, a, b)$ *is* QMA*-complete for* $k = O(1)$ *and* $b - a = \Omega(1)$*, where the* QMA*-hardness is with respect to quantum reductions.*

The two versions of the quantum PCP conjecture have been proven equivalent [2], and since Conjecture 2 is true for $b - a \geq 1/\text{poly}(n)$, we can also conclude that $\mathsf{QMA} = \mathsf{QPCP}(q, \alpha, \beta)$ with $q = O(1)$ and $\alpha - \beta \geq 1/\text{poly}(n)$.

Let us note that so far there is no multi-prover game equivalent to the QPCP conjecture, though the approximation of the maximum acceptance probability of certain multi-prover games up to an inverse-polynomial additive factor has been proven to be QMA-hard [12].

1.1 Our Results

In our work, we propose a new type of quantum proof systems, the Pointer QPCP, and formulate three equivalent versions of the Pointer QPCP conjecture. This may help towards proving or disproving the original QPCP conjecture. We start by describing a new proof system then we provide a new variant of the Local Hamiltonian problem and last we describe an equivalent polynomial size multi-prover game. Up to our knowledge, this is the first time a polynomial size multi-prover game has been proven equivalent to some QPCP conjecture.

Our new conjecture is a weaker statement than the original QPCP conjecture and hence it is an intermediate step which may be easier to prove. One may also try to prove the equivalence with the original conjecture, but despite being more structured than general QMA verifiers, Pointer QPCPs still have some characteristics, such as adaptiveness, which we do not know how to cast in term of the usual QPCPs. Moreover, having an equivalent game version of it might also lead to new methods that could potentially be relevant for attacking the original conjecture as well.

We now give some details of our results. We define a new quantum proof system, where the proof contains two separate parts, a classical and a quantum proof both of polynomial size. The verifier can first access a block from the classical proof and, depending on the content, he can then access a constant number of qubits from the quantum proof. Since the classical part can be seen as a pointer to the qubits that will be accessed, we denote this proof system by PointerQPCP(q, α, β). To be more specific, the verifier first reads a logarithmic number of bits from the classical part of the proof and then measures at most q qubits from the quantum part. He accepts a yes instance with probability at least α and a no instance with probability at most β. Since a Pointer QPCP is a generalization of QPCP, it follows that all problems in QMA have a PointerQPCP(q, α, β) proof system with $\alpha - \beta \geq 1/\text{poly}(n)$.

▶ **Conjecture 3** (Pointer QPCP Conjecture – Proof verification version). *It holds that* QMA $=$ PointerQPCP(q, α, β) *where* $q = O(1)$ *and* $\alpha - \beta = \Omega(1)$.

We note that quantum proof systems with classical and quantum parts have also appeared in [23]. There, the aim was to reduce the number of blocks being read in classical PCPs and hence, in the proposed model, a logarithmic size quantum proof is provided to the verifier who measures it and then reads only a single block from a polynomial size classical proof.

In addition to Pointer QPCPs, we also propose a "constraint satisfaction" version of the above conjecture which will turn out to be equivalent. We do this by defining a new variant of the Local Hamiltonian problem which we call the Set Local Hamiltonian problem. Here the input is m sets of a polynomial number of k-local Hamiltonians each, and we ask if there exists a representative Hamiltonian from each set such that the Hamiltonian corresponding to their sum has groundstate energy at most am or for every possible choice of representative Hamiltonians from each set, the Hamiltonian corresponding to their sum has groundstate energy at least bm. We denote the above problem by $\text{SLH}(k, a, b)$. Since the Local Hamiltonian problem is a special case of the Set Local Hamiltonian problem, where the sets are singletons, $\text{SLH}(k, a, b)$ is QMA-hard for $k \geq 2$ and $b - a \geq 1/\text{poly}(n)$.

▶ **Conjecture 4** (Pointer QPCP Conjecture – Constraint satisfaction version). *The* $\text{SLH}(k, a, b)$ *problem is* QMA*-complete for* $k = O(1)$ *and* $b - a = \Omega(1)$.

As mentioned earlier, the classical PCP theorem has another interesting equivalent formulation regarding the approximation of the maximum acceptance probability of multi-prover games [22], while the same is not known for the quantum case. We propose an equivalent multi-prover game formulation of the Pointer QPCP conjecture. Our game, which we call CRESP (Classical and Restricted-Entanglement Swapping-Provers) game, was inspired by the work of Fitzsimons and Vidick [12]. However, in order to prove an equivalence, we had to drastically change the game. In their work, a multi-prover game is proposed for the Local Hamiltonian problem in which the completeness-soundness gap is inverse polynomial. If we try to follow the same proof but with an instance of the Local Hamiltonian with constant gap, the gap does not survive and we end with an inverse-polynomial gap in the game. Hence we are not able to prove the equivalence with the standard QPCP conjecture.

We define our CRESP game to have one classical prover and logarithmically many quantum provers who are restricted both in the strategies they can perform and also in the initial quantum state they share. The verifier asks a single question of logarithmic length to all of them, the classical prover replies with logarithmically many bits, while the quantum provers reply with k 4-dimensional qudits. (For simplicity, we will omit the dimension of the qudit system in the rest of the paper.) The promise problem $\mathrm{CRESP}(k, \alpha, \beta)$ informally asks if we can distinguish between the cases when the provers win the game with probability at least α or at most β. Similarly to the previous problems, we will see that $\mathrm{CRESP}(k, \alpha, \beta)$ is QMA-complete for $\alpha - \beta \geq 1/\mathrm{poly}(n)$. See Theorem 22 for the precise statement.

▶ **Conjecture 5** (Pointer QPCP Conjecture – Game version). *The* $\mathrm{CRESP}(k, \alpha, \beta)$ *problem is* QMA*-complete for* $k = O(1)$ *and* $\alpha - \beta = \Omega(1)$*.*

Our main result is the equivalence of the above three formulations of the Pointer QPCP conjecture. It is stated formally in the following theorem.

▶ **Theorem 6** (Main theorem). *The three versions of the Pointer QPCP conjecture (Conjectures 3 to 5) are either all true or all false.*

The proof is divided into three steps: first, we show that Conjecture 3 implies Conjecture 4; second, we show that Conjecture 4 implies Conjecture 5; and finally, we prove that Conjecture 5 implies Conjecture 3.

The paper is organized as follows: In Section 2, we describe some standard definitions required for the rest of the paper. In Section 3, we present the definitions of our new notions, the Pointer QPCPs, the Set Local Hamiltonian problem, and the CRESP games. The proof of equivalence is presented in Section 4. We conclude the paper with some discussion and open problems in Section 5.

2 Preliminaries

In this section we provide some definitions that we use in the paper. We start by defining QMA, the quantum analogue of NP.

▶ **Definition 7** (Quantum Merlin-Arthur proof systems). Let $n \in \mathbb{Z}^+$ be the input size and p be a polynomial. A QMA protocol proceeds in the following steps.

1. The verifier receives an input x and a quantum proof $|\psi\rangle$ of size $p(n)$.
2. The verifier runs in polynomial time in n. He performs a general POVM measurement on $|\psi\rangle$ and decides on the acceptance or rejection of the input.

A promise problem $A = (A_{\mathrm{yes}}, A_{\mathrm{no}})$ belongs to QMA if it has a QMA proof system with the following properties.

Completeness. If $x \in A_{\text{yes}}$ then there is a $|\psi\rangle$ such that the verifier accepts w.p. at least $\frac{2}{3}$.
Soundness. If $x \in A_{\text{no}}$ then for all $|\psi\rangle$ the verifier accepts w.p. at most $\frac{1}{3}$.

Now we present the Local Hamiltonian problem, the quantum analogue of MAX-SAT.

▶ **Definition 8.** The *Local Hamiltonian* problem is denoted by $\textsc{LocalHam}(k, a, b)$ where $k \in \mathbb{Z}^+$ is called the locality and for $a, b \in \mathbb{R}$ it holds that $a < b$. It is the following promise problem. Let n be the number of the qubits of a quantum system. The input is a set of $m(n)$ Hamiltonians $H_1, \ldots, H_{m(n)}$ where m is a polynomial in n, $\forall i \in [m(n)] : 0 \leq H_i \leq \mathbb{1}$ and each H_i acts on k qubits out of the n qubit system. For $H \overset{\text{def}}{=} \sum_{j=1}^{m(n)} H_j$ the following two conditions hold.

- In a YES instance there exists a state $|\varphi\rangle \in \mathbb{C}^{2^n}$ such that $\langle\varphi| H |\varphi\rangle \leq a \cdot m(n)$.
- In a NO instance for all states $|\varphi\rangle \in \mathbb{C}^{2^n}$ it holds that $\langle\varphi| H |\varphi\rangle \geq b \cdot m(n)$.

Kitaev proved that for $k \geq 5$ and $b - a \geq 1/\mathit{poly}(n)$ the $\textsc{LocalHam}(k, a, b)$ problem is QMA-complete [18]. This completeness result was later improved for $k \geq 2$ [17].

We now define the quantum analogue of PCPs.

▶ **Definition 9** (Quantum Probabilistically Checkable Proofs). Let $n \in \mathbb{Z}^+$ be the input size and p be a polynomial. A QPCP protocol proceeds in the following steps.

1. The verifier receives an input x and a quantum proof $|\psi\rangle$ of size $p(n)$.
2. The verifier runs in time polynomial in n. He picks $O(\log n)$ bits uniformly at random, and based on the input and on the random bits, he performs a general POVM measurement on q qubits, and decides on acceptance or rejection of the input.

A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ belongs to $\mathsf{QPCP}(q, \alpha, \beta)$ if it has a QPCP proof system with the following properties.
Completeness. If $x \in A_{\text{yes}}$ then there is a $|\psi\rangle$ such that the verifier accepts w.p. at least α.
Soundness. If $x \in A_{\text{no}}$ then for all $|\psi\rangle$ the verifier accepts w.p. at most β.

We can easily prove the following statement:

▶ **Lemma 10.** *It holds that* $\mathsf{QMA} = \mathsf{QPCP}(q, \alpha, \beta)$ *where* $q = O(1)$ *and* $\alpha - \beta \geq 1/\text{poly}(n)$.

Proof. The containment $\mathsf{QPCP}(q, \alpha, \beta) \subseteq \mathsf{QMA}$ is trivial since the QMA verifier can read the whole proof and the power of QMA doesn't change if the gap is inverse-polynomial. The other direction of the containment follows from Kitaev's proof that the 5-Local Hamiltonian is QMA-complete [18, 4]. The QMA verifier in the proof is also a QPCP verifier. ◀

The quantum PCP conjecture has two equivalent versions:

▶ **Theorem 11** ([2]). *The class* QMA *is equal to the class* $\mathsf{QPCP}(q, \alpha, \beta)$ *with* $q = O(1)$ *and* $\alpha - \beta = \Omega(1)$ *(Conjecture 1) if and only if the Local Hamiltonian problem* $\textsc{LocalHam}(k, a, b)$ *is* QMA*-complete for* $k = O(1)$ *and* $b - a = \Omega(1)$*, where the* QMA*-hardness is with respect to quantum reductions (Conjecture 2).*

3 Pointer QPCPs, Set Local Hamiltonians, and CRESP Games

In this section, we present the definitions required for our conjectures. We start by defining Pointer QPCPs, a generalized version of QPCPs, in which the verifier can read a small number of bits from the classical part of the proof and then, based on that, read a constant number of qubits from the quantum part of the proof. Then we propose the Set Local Hamiltonian problem that can be thought of as a "constraint satisfaction" version of the conjecture. Finally, we define CRESP games which are restricted multi-prover games for which approximation of their value will turn out to be equivalent to the other two formulations.

3.1 Pointer QPCPs

▶ **Definition 12.** Let $n \in \mathbb{Z}^+$ be the input size, let q be a fixed parameter and let m, l, p be polynomials. A Pointer QPCP protocol proceeds in the following steps.

1. The verifier receives an input x and a two-part proof of size $m(n) + p(n)$ in the form $y_1...y_{m(n)} \otimes |\psi\rangle$, where where $y_i \in [l(n)]$ (i.e. each y_i can be written with $O(\log n)$ bits) and $|\psi\rangle$ is a state of $p(n)$ qubits. We refer to $y_1...y_{m(n)}$ as the classical part of the proof and $|\psi\rangle$ as the quantum part of the proof.
2. The verifier runs in time polynomial in n. He chooses uniformly at random a position $i \in [m(n)]$ of the classical proof to read. Then, based on his input, the random bits and the value of y_i, he chooses q qubits from the quantum proof, performs a general POVM measurement on them, and decides on acceptance or rejection of the input.

A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ belongs to $\mathsf{PointerQPCP}(q, \alpha, \beta)$ if it has a Pointer QPCP proof system with the following properties.

Completeness. If $x \in A_{\text{yes}}$ then there exists a $y_1...y_{m(n)} \otimes |\psi\rangle$ such that verifier accepts w.p. at least α.

Soundness. If $x \in A_{\text{no}}$ then for all $y_1...y_{m(n)} \otimes |\psi\rangle$ the verifier accepts w.p. at most β.

▶ **Lemma 13.** $\mathsf{QMA} = \mathsf{PointerQPCP}(q, \alpha, \beta)$ *where* $q = O(1)$ *and* $\alpha - \beta \geq 1/\text{poly}(n)$.

Proof. Since Pointer QPCPs are generalizations of QPCPs, we have that $\mathsf{QPCP}(q, \alpha, \beta) \subseteq \mathsf{PointerQPCP}(q, \alpha, \beta)$ for any values of q, α, and β. From Lemma 10, it follows that $\mathsf{QMA} \subseteq \mathsf{PointerQPCP}(q, \alpha, \beta)$. The other direction of the containment follows trivially since the QMA verifier can read the whole proof. ◀

Conjecture 3 asks whether QMA also has Pointer QPCPs with $q = O(1)$ and $\alpha - \beta = \Omega(1)$.

3.2 The Set Local Hamiltonian Problem

We define a new QMA-complete problem which is a generalization of the Local Hamiltonian problem and which will lead to another version of our conjecture.

▶ **Definition 14** (Set Local Hamiltonian Problem)**.** The *Set Local Hamiltonian* problem is denoted by $\text{SLH}(k, a, b)$ where $k \in \mathbb{Z}^+$ is called the locality and for $a, b \in \mathbb{R}$ it holds that $a < b$. It is the following promise problem. Let n be the number of the qubits of a quantum system, and m and l be two polynomials. The input for the problem are $m(n)$ sets of Hamiltonians. For all $i \in [m(n)]$ the set $\mathbf{H}_i$ contains $l(n)$ Hamiltonians, i.e., $\forall i \in [m(n)] : \mathbf{H}_i = \left\{H_{i,1}, \ldots, H_{i,l(n)}\right\}$. Each Hamiltonian is positive and has norm at most one, i.e., $\forall i \in [m(n)], \forall j \in [l(n)] : 0 \leq H_{i,j} \leq \mathbb{1}$. Each Hamiltonian acts non-trivially on at most k qubits out of the n qubits of the quantum system. The problem is to decide which one of the following two conditions hold.

- In a YES instance, there exists a function $f : [m(n)] \to [l(n)]$ and a state $|\varphi\rangle \in \mathbb{C}^{2^n}$ such that $\langle\varphi| \sum_{i=1}^{m(n)} H_{i,f(i)} |\varphi\rangle \leq a \cdot m(n)$.
- In a NO instance, for all functions $f : [m(n)] \to [l(n)]$ and for all states $|\varphi\rangle \in \mathbb{C}^{2^n}$, we have that $\langle\varphi| \sum_{i=1}^{m(n)} H_{i,f(i)} |\varphi\rangle \geq b \cdot m(n)$.

▶ **Lemma 15.** *The* $\text{SLH}(k, a, b)$ *problem is* QMA*-complete for* $k \geq 2$ *and* $b - a \geq 1/\text{poly}(n)$.

Proof. For the containment $\text{SLH}(k, a, b) \in \mathsf{QMA}$, let the witness have a classical part that contains the description of the function f and a quantum part that is supposed to be the state $|\varphi\rangle$. The quantum verifier can then apply the usual eigenvalue estimation on $\sum_{i=1}^{m(n)} H_{i,f(i)}$. The hardness of $\text{SLH}(k, a, b)$ comes trivially from the fact that Local Hamiltonian problem is a special case of the Set Local Hamiltonian problem with $l(n) = 1$. ◀

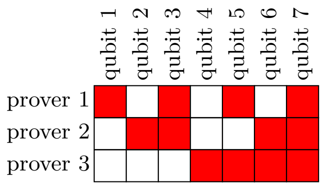


Figure 1 A possible distribution of the encoding of 7 qubits among 3 provers. The red cells correspond to the GHZ-like entangled states, while the white cells to $|0\rangle$ states.

Note that Conjecture 4 asks whether the Set Local Hamiltonian problem remains QMA-complete when the locality is constant and the gap between b and a is also constant.

3.3 CRESP Games

We now formally describe a new variant of quantum multi-prover games. These games are rather restricted but will allow us to state a third variant of our pointer QPCP conjecture.

3.3.1 Description of the Game

Let $n \in \mathbb{Z}^+$ be a parameter and m be a polynomial. The size of the game will be polynomial in n. The game is played by one classical prover, $\lceil \log(n+1) \rceil$ quantum provers, and a verifier. It is played as follows.

1. The quantum provers share the encoding of an arbitrary n-qubit state. (The encoding maps each qubit into a number of qudits and will be defined later.) They are not allowed to share any other resources.
2. The verifier picks a question i uniformly at random out of the $m(n)$ possible questions and sends the same question to all the provers (both quantum and classical).
3. The classical prover replies with $O(\log n)$ bits.
4. Each quantum prover replies by at most k qudits from their shared encoded state. All the quantum provers use the same strategy.
5. The verifier accepts or rejects, based on his question and the answers from the provers.

We denote these games by the acronym CRESP after the **C**lassical prover, the **R**estricted **E**ntanglement that the quantum provers can share and, since the only possible strategy the quantum provers can perform is to swap some of their qudits into the message register, we call them **S**wapping-**P**rovers.

3.3.2 Restriction on the Entanglement

The entangled state the provers share is of the following predefined form. First, the provers pick an arbitrary n-qubit state $|\phi\rangle \in \mathbb{C}^{2^n}$. The state $|\phi\rangle$ is encoded with a linear isometry $\mathcal{E} = \mathcal{E}_1 \otimes \mathcal{E}_2 \otimes \ldots \otimes \mathcal{E}_n$ where each qubit of $|\phi\rangle$ is encoded with $\mathcal{E}_i : \mathbb{C}^2 \to \bigotimes_{j=1}^{\lceil \log(n+1) \rceil} \mathcal{H}_{i,j}$. For all i and j, $\mathcal{H}_{i,j} \cong \mathbb{C}^4$, that is, $\mathcal{H}_{i,j}$ is a four-dimensional space which we simply call qudit. To define $\mathcal{E}_i$, let's fix some ordering on the non-empty subsets of $[\lceil \log(n+1) \rceil]$. Let Q_i be the i-th subset, $\mathcal{S}_i \overset{\text{def}}{=} \bigotimes_{j \in Q_i} \mathcal{H}_{i,j}$, and $\overline{\mathcal{S}_i} \overset{\text{def}}{=} \bigotimes_{j \notin Q_i} \mathcal{H}_{i,j}$. For each $i \in [n]$, we define

$\mathcal{E}_i$ by giving its action on the standard basis states.

$$\mathcal{E}_i(|0\rangle) \overset{\text{def}}{=} \frac{1}{\sqrt{2}} \left(|0\rangle^{\otimes|Q_i|} + |1\rangle^{\otimes|Q_i|}\right)_{\mathcal{S}_i} \otimes \left(|0\rangle^{\otimes\lceil\log(n+1)\rceil-|Q_i|}\right)_{\overline{\mathcal{S}_i}} \tag{1}$$

$$\mathcal{E}_i(|1\rangle) \overset{\text{def}}{=} \frac{1}{\sqrt{2}} \left(|2\rangle^{\otimes|Q_i|} + |3\rangle^{\otimes|Q_i|}\right)_{\mathcal{S}_i} \otimes \left(|0\rangle^{\otimes\lceil\log(n+1)\rceil-|Q_i|}\right)_{\overline{\mathcal{S}_i}} \tag{2}$$

We refer to the states in $\mathcal{S}_i$ as GHZ-like states. After $\mathcal{E}$ is applied, prover j receives the qudits that live in space $\bigotimes_{i=1}^{n} \mathcal{H}_{i,j}$. A possible distribution of the qudits is depicted in Figure 1.

We remark that despite being forced to share a state in a specific encoded form, the Provers still have all the freedom to choose the original state to be encoded, which lives in a 2^n dimensional Hilbert space.

3.3.3 Description of the CRESP Problem

We are interested in the maximum acceptance probability the provers can achieve, which is called the value of the game. Here the maximum is taken over all legitimate shared states and all legitimate provers' strategies. We now define the promise problem that corresponds to the approximation of the value of CRESP games.

▶ **Definition 16.** Let $k \in \mathbb{Z}^+$ and $\alpha, \beta \in \mathbb{R}$ with $\alpha > \beta$. Then, CRESP(k, α, β) is the following promise problem. The input is the description of a CRESP game defined above where the quantum provers answer at most k qudits and the following conditions hold.

- In a YES instance the value of the game is at least α.
- In a NO instance the value of the game is at most β.

We will prove that the CRESP(k, α, β) problem is QMA-complete for $k = O(1)$ and $\alpha - \beta \geq 1/\text{poly}(n)$. We defer this proof to Section 4.2 as it needs results that we will establish later. Again, we note that Conjecture 5 asks whether CRESP(k, α, β) remains QMA-complete when $k = O(1)$ and $\alpha - \beta = \Omega(1)$.

4 Equivalence of Our QPCP Conjectures

In this section we prove Theorem 6, the equivalence of the three versions of our Pointer QPCP conjecture. The proof proceeds in the following three steps. In Section 4.1, we show that if Conjecture 3 is true then Conjecture 4 is also true. We do this by reducing any problem with a Pointer QPCP proof system to the Set Local Hamiltonian problem. In Section 4.2 we show that if Conjecture 4 is true then Conjecture 5 is also true by giving a reduction from the Set Local Hamiltonian problem to our decision problem involving CRESP games. To complete the cycle, we prove in Section 4.3 that if Conjecture 5 is true then Conjecture 3 is also true by giving a Pointer QPCP proof system for an arbitrary CRESP game.

4.1 From Pointer QPCP to the Set Local Hamiltonian Problem

In this section we show that if Conjecture 3 is true then Conjecture 4 is also true. We show that any problem $P \in$ PointerQPCP(q, α, β) is polynomial-time reducible to SLH$(q, 1-\alpha, 1-\beta)$. Assuming Conjecture 3, this means that the Set Local Hamiltonian problem is QMA-hard. The containment of the Set Local Hamiltonian problem in QMA is implied by Lemma 15.

▶ **Theorem 17.** *Any problem $P \in$ PointerQPCP(q, α, β) can be reduced to* SLH$(q, 1-\alpha, 1-\beta)$ *in polynomial time.*

Proof. Let $y_1, \ldots, y_m$ be the classical part of the proof where $m = m(n)$ for a polynomial m and $|\psi\rangle$ be the quantum part of the proof, which contains $p(n)$ qubits, for a polynomial p. Suppose that each y_i can take $l = l(n)$ different values, for a polynomial l. We construct an instance of the Set Local Hamiltonian problem that consists of m sets of Hamiltonians $\mathbf{H}_i$, for $i \in [m]$, where $\mathbf{H}_i = \{H_{i,j}\}_{j\in[l]}$ and the Hamiltonians act on a $p(n)$-qubit system. Let $H_{i,j}$ be the rejection POVM generated by the Pointer QPCP verifier (who runs in quantum polynomial time) over the constant number of qubits when he reads register i from the classical part of the proof and it contains the value j, i.e., $j = y_i$.

First we prove that if there is a proof that makes the Pointer QPCP verifier accept with probability greater than α then there is a function f such that the groundstate of $\sum_{i=1}^{m} H_{i,f(i)}$ has energy at most $(1-\alpha)m$. Let $y_1...y_m \otimes |\psi\rangle$ be such proof and let α_i be the acceptance probability of the Pointer QPCP verifier when the verifier queries i. Since the verifier picks an i uniformly at random, it follows that $\frac{1}{m}\sum_i \alpha_i = \alpha$. Let $f(i) \overset{\text{def}}{=} y_i$. In this case, the energy of $|\psi\rangle$ on $\sum_i H_{i,f(i)}$ is

$$\langle\psi| \left(\sum_i H_{i,y_i}\right) |\psi\rangle \leq \sum_i (1-\alpha_i) = (1-\alpha)m.$$

For the other direction of the proof, suppose that there is a function f and a state $|\psi\rangle$ such that $\langle\psi| \left(\sum_i H_{i,f(i)}\right) |\psi\rangle \leq (1-\beta)\,m$. Then there is a proof that makes the Pointer QPCP verifier accept with probability bigger than β. Let $(f(1), f(2), ..., f(m)) \otimes |\psi\rangle$ be the proof for the Pointer QPCP verifier, and in this case the acceptance probability is

$$\frac{1}{m}\sum_i (1 - \langle\psi| H_{i,f(i)} |\psi\rangle) = 1 - \frac{1}{m}\langle\psi| \left(\sum_i H_{i,f(i)}\right) |\psi\rangle \geq \beta.$$

This finishes the proof of the reduction. ◀

4.2 From the Set Local Hamiltonian Problem to CRESP Games

In this section we show that if Conjecture 4 is true then Conjecture 5 is also true. We do this by giving a reduction from the SLH(k,a,b) problem to the CRESP$(k, 1-a/2, 1-b/2)$ problem. Assuming Conjecture 4, this implies that the CRESP$(k, 1-a/2, 1-b/2)$ problem is QMA-hard. We prove the containment CRESP$(k, 1-a/2, 1-b/2) \in$ QMA in Theorem 22.

We construct a CRESP game for the Set Local Hamiltonian problem. The main idea in the construction is the following. In our game, the verifier picks an index $i \in [m]$ uniformly at random and sends i to all the provers. The classical prover tells the verifier the specific Hamiltonian that should be taken from set i, i.e., the value of $f(i)$. The quantum provers replies with the encoding of the qubits of groundstate of the Hamiltonian $\sum_i H_{i,f(i)}$.

First, the verifier checks if the received qudits lie in the codespaces of the qubits of $H_{i,f(i)}$, and if not he rejects. Using the definition of the encoding, the projector onto the codespace of the qubit q is described by

$$(\Pi_q)_{\mathcal{S}_q} \otimes |0\rangle\langle 0|_{\overline{\mathcal{S}_q}}, \text{ with } \Pi_q = \frac{1}{2}\left(\sum_{u,v\in\{0,1\}} \left|u^{|Q_q|}\right\rangle \left\langle v^{|Q_q|}\right| + \sum_{w,z\in\{2,3\}} \left|w^{|Q_q|}\right\rangle \left\langle z^{|Q_q|}\right|\right).$$

If the above test succeeds then the verifier picks a bit uniformly at random and if it is 0, he accepts. Otherwise, the verifier decodes the answered qudits by inverting the mapping $\mathcal{E}$, defined by Equations 1 and 2, for all the qubits in Hamiltonian $H_{i,f(i)}$. Then, he performs

Protocol 1 CRESP Game for SLH(k, a, b)

1. The provers pick an n-qubit state $|\phi\rangle$ and share its encoding $\mathcal{E}(|\phi\rangle)$. (In the honest case, $|\phi\rangle$ is supposed to be the groundstate of Hamiltonian H.)
2. The verifier picks $i \in [m]$ uniformly at random and sends it to all the provers.
3. The classical prover sends some $j \in [l]$.
4. Each quantum prover sends k qudits.
5. The verifier performs the following tests.
   - T_1. Check if the answered qudits lie in the codespaces of the qubits of $H_{i,f(i)}$ and reject if not. Otherwise, continue.
   - T_2. Pick $b \in \{0, 1\}$ uniformly at random and accept if $b = 0$. Otherwise, continue.
   - T_3. Decode the received qudits and perform the measurement specified by $H_{i,f(i)}$ and accept or reject depending on the outcome.

the measurement that corresponds to $H_{i,f(i)}$ on the decoded qubits and accepts or rejects based on the outcome.

If the Hamiltonian $\sum_i H_{i,f(i)}$ has an eigenstate with small eigenvalue then the provers will pass the test with high probability. Using the fact that the provers share a state in the predefined encoding and the restriction on the quantum provers' strategies, we also show that the verifier will reject with high probability if all states have high eigenvalues. The description of the game is in Protocol 1.

▶ **Theorem 18.** *The game defined by Protocol 1 has completeness $1 - a/2$ and soundness $1 - b/2$.*

Proof. Lemma 19 proves completeness while Lemma 21 proves soundness. ◀

▶ **Lemma 19** (Completeness)**.** *If there is a function f such that the groundstate of $\sum_i H_{i,f(i)}$ has eigenvalue at most am then the maximum acceptance probability of the game is at least $1 - a/2$.*

Proof. Let the quantum provers share $\mathcal{E}(|\psi\rangle)$, the encoding of the groundstate $|\psi\rangle$ of $H \stackrel{\text{def}}{=} \sum_i H_{i,f(i)}$. When the verifier queries i, the classical prover answers $f(i)$ and all quantum provers honestly reply with their shares of the encodings of the k qubits corresponding to $H_{i,f(i)}$. The verifier always measures Π_i, and hence he accepts with probability

$$\frac{1}{2} + \frac{1}{2}\left(1 - \frac{1}{m}\sum_{i=1}^{m} \langle\psi| H_{i,f(i)} |\psi\rangle\right) = 1 - \frac{1}{2m}\langle\psi| H |\psi\rangle \geq 1 - \frac{a}{2}.$$ ◀

The following technical lemma is the key to prove soundness. It establishes that when the provers reply with the qudits that belong to the encoding of a different qubit, the verifier will detect it with probability at least half. We defer the full proof of this lemma to the full version of the paper.

▶ **Lemma 20.** *If the provers are asked for the encoding of qubit i and they answer with the qudits that correspond to the encoding of a different qubit, then the answered state projects to the correct codespace, i.e., the subspace that corresponds to the projector Π_i, with probability at most $1/2$.*

Proof Sketch. Since the provers have the same strategy for a fixed question and they can only do swaps, the only cheating strategy for the provers is to answer the encoding of a qubit which is different from the one the verifier asked for. In this case, by the properties of the chosen encoding, the state that should be a GHZ-like state is actually separable and it projects to the correct codespace with probability at most half. ◀

We sketch now the proof of the soundness property and defer its full proof to the full version of the paper.

▶ **Lemma 21** (Soundness). *If, for all functions f, the groundstate of $\sum_i H_{i,f(i)}$ has eigenvalue at least bm then the maximum acceptance probability of the game is at most $1 - b/2$.*

Proof Sketch. We can prove, using Lemma 20 and tests T_1 and T_2, that the optimal strategy for the provers is to answer honestly the encoding of the asked qubits. In this case, we can bound the maximum acceptance probability in the game using the fact that the original Hamiltonian has no low-energy groundstate. ◀

We now show that even though our game seems very restricted, it is in fact QMA-hard to approximate its value to within an inverse-polynomial precision.

▶ **Theorem 22.** *The* CRESP(k, α, β) *problem is* QMA*-complete for $k = O(1)$ and $\alpha - \beta \geq 1/\mathrm{poly}(n)$.*

Proof. The containment in QMA is simple: The QMA proof is the state the provers choose before the encoding together with the classical information that describes the behavior of all the provers. Then the QMA verifier can create the encoding and simulate the game. This leads to the same acceptance probability as that of the game which means that there is an inverse-polynomial gap between completeness and soundness in the QMA protocol. The QMA-hardness follows from Lemma 15 and Theorem 18. ◀

4.3 From CRESP Games to Pointer QPCPs

In this section we show that if Conjecture 5 is true then Conjecture 3 is also true. We do this by proving that CRESP$(k, \alpha, \beta) \in$ PointerQPCP(k, α, β). Assuming Conjecture 5, this implies that QMA $\subseteq$ PointerQPCP(k, α, β). The inclusion PointerQPCP$(k, \alpha, \beta) \subseteq$ QMA follows trivially, the same way as in Lemma 13.

▶ **Theorem 23.** CRESP$(k, \alpha, \beta) \in$ PointerQPCP(k, α, β).

Proof. In CRESP games, the strategy of the quantum provers consists of the choice of the shared state and the choice of which qudits to answer for each one of the verifier's questions. For the classical prover, the strategy consists of the his answers for each one of the verifier's questions. Therefore, we can have a Pointer QPCP whose proof will be as follows: for the classical part, for each possible question of the verifier, we include the indices of the qudits answered by the quantum provers and the answer of the classical prover. The quantum part of the proof will be the shared state before the encoding. With this information, the Pointer QPCP verifier can simulate the provers and the verifier of the CRESP game.

Formally, the verifier of the Pointer QPCP protocol is provided a proof of the form $y_1 \ldots y_m \otimes |\psi\rangle$, where y_i can be seen as a pair (s_i, c_i). The verifier will do the following.

1. He picks a question i uniformly at random as the verifier of the game.
2. He reads the corresponding strategy of the provers, i.e., (s_i, c_i).
3. He creates the encoding of the qubits that are specified by strategy s_i.

4. He simulates the verifier of the game using the encoded qubits as the quantum provers' answers and c_i as the classical prover's answer.
5. He accepts if and only if the verifier of the game accepts.

In our construction, we crucially use the fact that each quantum prover has the same strategy, as otherwise, the QPCP verifier would need to read out the strategies of each prover, which would require $\Omega\left(\log^2(n)\right)$ bits of information. Note that we only read out k qubits from the quantum part of the proof. We are left to prove completeness and soundness.

For completeness, it is not hard to see that if there is a strategy for the provers in the game with acceptance probability p then there is a Pointer QPCP that accepts with probability p as well, just by providing the values of s_i, c_i, and $|\psi\rangle$ that lead to acceptance with probability p in the game.

For soundness, if there are values of $y_i = (s_i, c_i)$ and $|\psi\rangle$ that make the Pointer QPCP verifier accept with some probability then these values can be translated to strategies of the provers in the CRESP game that will achieve the same acceptance probability. ◀

5 Discussions and Open Problems

We defined a new variant of quantum proof systems, the Pointer QPCPs, and provided three equivalent versions of the Pointer QPCP conjecture. Our conjecture is weaker than the original QPCP conjecture and hence may be easier to prove, which might also be facilitated with its equivalent game formulation. On the contrary, proving its equivalence to the QPCP conjecture, and hence dealing with the adaptivity property, might provide some insight on proving the original QPCP conjecture.

It is an interesting question to see whether we can define a more natural game which is equivalent to the Pointer QPCP conjecture. For our equivalence, we were forced to impose stringent constraints on the game. Nevertheless, it seems that if we allow the quantum provers to either share some more general entangled state or apply any operator to the state they share other than swapping, then it is not clear how not to lose the constant gap when constructing the witness [12, 15] or not to increase the question size to exponential [19].

Even with our constraints, CRESP games remain equivalent to Pointer QPCPs. Since Pointer QPCPs are a superclass of QPCPs, finding a game that is equivalent to the original QPCP would potentially impose even further constraints. One of the main problems going from a game back to Local Hamiltonians or QPCPs, is that to simulate the game, the strategies of the provers must be somehow simulated and when we try to do this with Local Hamiltonians, the gap vanishes, while for QPCPs, we require the classical pointer queries. Note that, in the Set Local Hamiltonian problem the gap doesn't depend on the size of the sets, by definition. Whereas, if we want to go to the usual Local Hamiltonian problem then the absolute gap is divided by the total number of Hamiltonians and so the gap vanishes.

Acknowledgements We thank Thomas Vidick for helpful discussions. Supported by ERC QCC, ANR RDAM.

References

1 Dorit Aharonov, Itai Arad, Zeph Landau, and Umesh Vazirani. The detectability lemma and quantum gap amplification. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC '09)*, pages 417–426, 2009.

2 Dorit Aharonov, Itai Arad, and Thomas Vidick. Guest column: the quantum PCP conjecture. *SIGACT News*, 44(2):47–79, 2013.

3 Dorit Aharonov and Lior Eldar. The commuting local Hamiltonian problem on locally expanding graphs is approximable in NP. *Quantum Information Processing*, 14(1):83–101, January 2015.

4 Dorit Aharonov and Tomer Naveh. Quantum NP - A Survey, 2002. `arXiv:arXiv:quantum-ph/0210077`.

5 Itai Arad. A Note About a Partial No-go Theorem for Quantum PCP. *Quantum Info. Comput.*, 11(11-12):1019–1027, November 2011. URL: `http://dl.acm.org/citation.cfm?id=2230956.2230966`.

6 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

7 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

8 Fernando G. S. L. Brandão and Aram W. Harrow. Product-state approximations to quantum ground states. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 871–880, 2013.

9 Toby S. Cubitt and Ashley Montanaro. Complexity classification of local Hamiltonian problems. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS '14)*, pages 120–129, 2014.

10 Irit Dinur. The PCP Theorem by Gap Amplification. *Journal of the ACM*, 54(3), June 2007.

11 Lior Eldar and Aram W. Harrow. Local Hamiltonians with no low-energy trivial states, 2015. URL: `http://arxiv.org/abs/1510.02082`, `arXiv:arXiv:quantum-ph/1510.02082`.

12 Joseph Fitzsimons and Thomas Vidick. A multiprover interactive proof system for the local Hamiltonian problem. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, (ITCS '15)*, pages 103–112, 2015.

13 Sevag Gharibian, Yichen Huang, Zeph Landau, and Seung Woo Shin. Quantum Hamiltonian complexity. *Foundations and Trends in Theoretical Computer Science*, 10(3):159–282, 2015.

14 Sean Hallgren, Daniel Nagaj, and Sandeep Narayanaswami. The local Hamiltonian problem on a line with eight states is QMA-complete. *Quantum Info. Comput.*, 13(9-10):721–750, September 2013.

15 Zhengfeng Ji. Classical verification of quantum proofs. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 885–898, 2016.

16 Julia Kempe, Alexei Kitaev, and Oded Regev. The complexity of the local Hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, May 2006.

17 Julia Kempe and Oded Regev. 3-local Hamiltonian is QMA-complete. *Quantum Info. Comput.*, 3(3):258–264, 2003.

18 A. Kitaev, A. Shen, and M. N. Vyalyi. *Classical and quantum computation*. Graduate studies in mathematics. American mathematical society, 2002. URL: `http://opac.inria.fr/record=b1100148`.

19 Anand Natarajan and Thomas Vidick. Constant-soundness interactive proofs for local hamiltonians. *CoRR*, abs/1512.02090, 2015. URL: `http://arxiv.org/abs/1512.02090`.

20 Roberto Oliveira and Barbara M. Terhal. The complexity of quantum spin systems on a two-dimensional square lattice. *Quantum Info. Comput.*, 8(10):900–924, 2008.

21 Tobias J Osborne. Hamiltonian complexity. *Reports on Progress in Physics*, 75(2):022001, 2012.

22 Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, June 1998.

23 Ran Raz. Quantum information and the PCP theorem. In *Proceedings of 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS '05)*, 2005.

A Formal Exploration of Nominal Kleene Algebra

Paul Brunet[1] **and Damien Pous**[*2]

1 Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP
2 Univ Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP

Abstract

An axiomatisation of Nominal Kleene Algebra has been proposed by Gabbay and Ciancia, and then shown to be complete and decidable by Kozen et al. However, one can think of at least four different formulations for a Kleene Algebra with names: using freshness conditions or a presheaf structure (types), and with explicit permutations or not. We formally show that these variations are all equivalent.

Then we introduce an extension of Nominal Kleene Algebra, motivated by relational models of programming languages. The idea is to let letters (i.e., atomic programs) carry a set of names, rather than being reduced to a single name. We formally show that this extension is at least as expressive as the original one, and that it may be presented with or without a presheaf structure, and with or without syntactic permutations. Whether this extension is strictly more expressive remains open.

All our results were formally checked using the Coq proof assistant.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages, F.4.1 Mathematical Logic

Keywords and phrases Nominal sets, Kleene algebra, equational theory, Coq

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.22

1 Introduction

Gabbay and Ciancia introduced a nominal extension of Kleene algebra [3], as a framework for trace semantics with dynamic allocation of resources. The associated semantics extends formal languages into nominal languages, where words have a nominal structure. Kozen et al. recently proved the completeness of the proposed axiomatisation [6], and proposed a coalgebraic treatment [5] yielding decidability of the equational theory.

They use the following syntax for nominal regular expressions:

$$e, f ::= a \in \Sigma \mid 0 \mid 1 \mid e + f \mid e \cdot f \mid e^\star \mid \nu a.e \ ,$$

where Σ is the alphabet, and $\nu a.e$ makes it possible to generate a fresh letter (or name) a before continuing as e. For instance, the expression $\nu a.\nu b.(a{\cdot}b)$ denotes the language of all words of length two consisting of two distinct letters.

While such a syntax is natural from a nominal point of view, other choices are possible. For instance, one might expect expressions to be typed or classified according to their set of free names. Similarly, name permutations, which are available in any model, can be reified at the syntactic level. We first list four axiomatisations of the corresponding presentations—one

* This author was funded by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157); this work has also been supported by the project ANR 12IS02001 PACE.

of them corresponding to Gabbay and Ciancia's axiomatisation, and we prove that all choices are in fact equivalent. For the sake of the proofs, we need to introduce the *positive* fragments, where the constant 0 denoting the empty language is excluded; these fragments are interesting because they are *stable*: any proof of an equality whose members belong to the fragment only uses expressions from that fragment.

Kleene algebra are known to be complete not only with respect to language models, but also relational models. There, the letters from the alphabet are interpreted as binary relations, and the regular operations correspond to standard operations on binary relations: union, composition, reflexive transitive closure. This makes it possible to interpret imperative programs, seen as state transformers: binary relations between memory states. Kozen actually designed an extension of Kleene algebra, Kleene algebra with tests [4], which makes it possible to represent not only the control flow of such programs, but also the tests and conditions appearing in while loops and branching statements.

Extending Kleene algebra with names seems appropriate to model imperative programs with local variables, where parts of the memory is visible only locally. The previous notion of nominal Kleene algebra is however not really appropriate for this purpose: letters of the alphabet (i.e., atomic programs, instructions) are equated with names bound by the $\nu a.e$ construct (i.e., memory locations). In contrast, the instruction $x \leftarrow y$ that assigns to variable x the value of variable y should be an elementary construction depending on the names x and y. For this reason, we provide an extension of the syntax where letters carry a list of names. (We could use arbitrary nominal sets, but we restrict to a concrete representation in this work for the sake of simplicity.) The typed version of this extension is more appropriate for modelling imperative programs with local variables; like above for plain nominal Kleene algebra, we show that the various presentations are equivalent. We moreover show that the extension is conservative: plain nominal Kleene algebra can be encoded into the extended ones. Whether a converse encoding is possible remains open.

Outline. We define the various theories in Section 2 and we compare them in Section 3. In Section 4 we provide a relational interpretation for our extended model. We conclude in Section 5.

Notation. We write $\wp_f(A)$ for the set of finite subsets of A. The set of natural numbers is written $\mathbb{N}$. Composition of two functions f and g is written $f \circ g$; it maps x to $f(g(x))$.

2 Expressions and proofs

2.1 Atoms and letters

Let $\mathcal{A}$ be an infinite set of *atoms* with decidable equality. We consider in this paper *finitely supported permutations of atoms*, simply called permutations in the following. They are bijections π such that there is a finite set $A \subseteq \mathcal{A}$ such that $a \notin A \Rightarrow \pi(a) = a$. The *inverse* of a permutation π is written π^{-1}. The *identity permutation* is denoted by $()$, and the permutation exchanging a and b, and leaving every other atom unchanged, is written $(a\ b)$. Finally, if π is a permutation and A is a finite set of atoms, $\pi(A) := \{\pi(a) \mid a \in A\}$ is *the image of A under π*.

We consider as *letters* an arbitrary nominal set $\mathcal{L}$[2, 7], which we assume to be decidable. Such a set is specified through the data of its set of elements, a function $\natural() : \mathcal{L} \to \wp_f(\mathcal{A})$ mapping every element to its *support*, and an *action* of the group of permutations on $\mathcal{L}$.

These functions must satisfy the following axioms:

$$\forall x \in \mathcal{L}, \forall \pi, (\forall a \in \natural(x), \pi(a) = a) \Rightarrow \pi(x) = x. \quad (1)$$

$$\forall x \in \mathcal{L}, \forall \pi, \natural(\pi(x)) = \pi(\natural(x)). \quad (2)$$

$$\forall x \in \mathcal{L}, \forall \pi, \pi', \pi(\pi'(x)) = \pi \circ \pi'(x). \quad (3)$$

2.2 Expressions and sets of expressions

We define a single type for expressions, containing all possible operators, and we define several fragments of it afterwards. Doing so makes it possible to share several definitions, enabling important code-reuse in our proof scripts.

▸ **Definition 1** (Expressions). The set $\mathbb{E}$ of *expressions* is composed of terms formed over the following syntax, where the letter A is a finite set of atoms, π denotes a permutation, a is an atom and l is a letter:

$$e, f \coloneqq 0 \mid 1 \mid e+f \mid e\cdot f \mid e^\star \mid \nu_a.e \mid l \mid a \mid \langle\pi\rangle e \mid \bot_A \mid id_A \mid w_a.e.$$

Product $(\cdot)$, sum $(+)$ and Kleene star $(\cdot^\star)$ are the regular operations, together with the associated constants 0 and 1, ν_a is name restriction. Variables can be either letters l or atoms a. We include a syntactic construction for explicit permutations $\langle\pi\rangle$. The remaining entries ($\bot_A$, id_A, and w_a) are for the presheaf presentation; we discuss them in Section 2.2.2

2.2.1 Untyped expressions

▸ **Definition 2** (Untyped expression). An expression e is *untyped* if it neither contains the operator w_a nor the constants $\bot_A$ and id_A. The set of untyped expressions is written $\mathbb{U}$.

We define freshness only for untyped expressions:

▸ **Definition 3** (Freshness). An atom a *is fresh for* e if the judgement $a \# e$ can be inferred in the following system.

$$\frac{}{a \# 1} \qquad \frac{}{a \# 0} \qquad \frac{a \notin \natural(l)}{a \# l} \qquad \frac{a \neq b}{a \# b} \qquad \frac{a \# e \quad a \# f}{a \# e+f}$$

$$\frac{a \# e \quad a \# f}{a \# e\cdot f} \qquad \frac{a \# e}{a \# e^\star} \qquad \frac{}{a \# \nu_a.e} \qquad \frac{a \# e}{a \# \nu_b.e} \qquad \frac{\pi^{-1}(a) \# e}{a \# \langle\pi\rangle e}.$$

Accordingly, the *support* of an untyped expression e, written $\natural(e)$, is the unique set such that $\forall a,\ a \# e \Leftrightarrow a \notin \natural(e)$.

2.2.2 Typed expressions

For the presheaf approach, we replace freshness assumptions with type derivations. In order to enforce uniqueness of types, we replace the constants 0 and 1 from the untyped syntax by the annotated constants $\bot_A$ and id_A, and we use explicit weakenings (w_a).

▸ **Definition 4** (Typed expressions). For any $e \in E$ and $A \in \wp_f(\mathcal{A})$, e *has the type* A if the judgement $e : A$ can be inferred in the following system:

$$\frac{}{id_A : A} \qquad \frac{}{\bot_A : A} \qquad \frac{l \in \mathcal{L}}{l : \natural(l)} \qquad \frac{a \in \mathcal{A}}{a : \{a\}} \qquad \frac{e : A \quad f : A}{e+f : A}$$

$$\frac{e : A \quad f : A}{e\cdot f : A} \qquad \frac{e : A}{e^\star : A} \qquad \frac{e : A \cup \{a\} \quad a \notin A}{\nu_a.e : A} \qquad \frac{e : A \setminus \{a\} \quad a \in A}{w_a.e : A} \qquad \frac{e : \pi^{-1}(A)}{\langle\pi\rangle e : A}$$

If this is the case, then e is *typed.* The set of typed expressions is written $\mathbb{T}$.

▸ Remark. This type system is syntax directed and yields a simple decision procedure.

2.2.3 Expressions over letters or atoms

A significant motivation for this work was to study the differences between having atoms or letters as variables in expressions. Hence we define two other subsets.

▸ **Definition 5** (Atomic expressions, literate expressions)**.** An expression e is called *atomic* (respectively *literate*) if it does not contain letters (respectively atoms) as variables. The set of atomic expressions is $\mathbb{E}\langle\mathcal{A}\rangle$, and the set of literate expressions is $\mathbb{E}\langle\mathcal{L}\rangle$.

Intuitively, there are two main differences between having atoms or letters as variables. First, a letter may depend on many atoms. Second, two letters with the same support can still be different, whereas the following equivalence holds :

$$\forall a,b\in\mathcal{A},\ a=b\Leftrightarrow(\forall c\in\mathcal{A},\ c\# a\Leftrightarrow c\# b)\,.$$

2.2.4 Positive expressions

For the sake of proofs, we also define the classes of expressions without 0 or $\bot_A$ as a sub-expression. A motivation for excluding these is that in any reasonable system $0\equiv 0\cdot e$. Hence if there is an atom a not fresh for e, we would have two equivalent expressions with different sets of fresh variables.

▸ **Definition 6** (Positive expression)**.** An expression e is *positive* if it does not contain 0 nor $\bot_A$ as a sub-expression. The set of positive expressions is $\mathbb{E}^+$.

For concision, we write $\mathbb{E}^+\langle\mathcal{L}\rangle$ for $\mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{E}^+$, and $\mathbb{E}^+\langle\mathcal{A}\rangle$ for $\mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{E}^+$.

2.2.5 Explicit permutations

Our syntax includes for explicit permutations $\langle\pi\rangle$, while permutations are usually considered as external operations. This allows one to manipulate permutations inside the expressions, and we shall see that this addition does not raise the complexity of the problem.

Nevertheless, we need to formally define the semantics of permutations on expressions.

▸ **Definition 7** (Action of a permutation on an expression)**.** Let $e\in\mathbb{E}$ be an expression and π a permutation. The *action of* π *on* e, written $\pi\bowtie e$, is defined as follows:

$$\begin{array}{ccc}
\pi\bowtie 1:=1 & \pi\bowtie 0:=0 & \pi\bowtie(w_a.e):=w_{\pi(a)}.(\pi\bowtie e)\\
\pi\bowtie id_A:=id_{\pi(A)} & \pi\bowtie\bot_A:=\bot_{\pi(A)} & \pi\bowtie(\nu_a.e):=\nu_{\pi(a)}.(\pi\bowtie e)\\
\pi\bowtie a:=\pi(a) & \pi\bowtie l:=\pi(l) & \pi\bowtie(\langle\pi'\rangle e):=\langle\,()\,\rangle(\pi\circ\pi')\bowtie e\\
\pi\bowtie(e^\star):=(\pi\bowtie e)^\star & \pi\bowtie(e\cdot f):=\pi\bowtie e\cdot\pi\bowtie f & \pi\bowtie(e+f):=\pi\bowtie e+\pi\bowtie f
\end{array}$$

Expressions without explicit substitutions are called *clean.*

▸ **Definition 8** (Clean expressions)**.** An expression e is *clean* if it never uses the operator $\langle\pi\rangle$. The set of clean expressions is $\mathbb{C}$.

Applying permutations preserves all classes we have listed so far:

$$\frac{Ax \vdash f = e}{Ax \vdash e = f} \qquad \frac{Ax \vdash e = f \quad Ax \vdash f = g}{Ax \vdash e = f} \qquad \frac{}{Ax \vdash e = f}\;(e,f) \in Ax$$

(a) Equivalence and axiom rules.

$$\frac{}{Ax \vdash 0 = 0} \qquad \frac{}{Ax \vdash 1 = 1} \qquad \frac{}{Ax \vdash id_A = id_A} \qquad \frac{}{Ax \vdash \bot_A = \bot_A}$$

$$\frac{}{Ax \vdash l = l} \qquad \frac{}{Ax \vdash a = a} \qquad \frac{Ax \vdash e = g \quad Ax \vdash f = h}{Ax \vdash e + f = g + h} \qquad \frac{Ax \vdash e = g \quad Ax \vdash f = h}{Ax \vdash e \cdot f = g \cdot h}$$

$$\frac{Ax \vdash e = f}{Ax \vdash e^\star = f^\star} \qquad \frac{Ax \vdash e = f}{Ax \vdash \nu_a.e = \nu_a.f} \qquad \frac{Ax \vdash e = f}{Ax \vdash w_a.e = w_a.f} \qquad \frac{Ax \vdash e = f}{Ax \vdash \langle\pi\rangle\, e = \langle\pi\rangle\, f}$$

(b) Congruence rules.

$$\frac{}{Ax \vdash e + f = f + e} \qquad \frac{}{Ax \vdash e + (f + g) = (e + f) + g} \qquad \frac{}{Ax \vdash e + e = e}$$

$$\frac{}{Ax \vdash e \cdot (f + g) = (e \cdot f) + (e \cdot g)} \qquad \frac{}{Ax \vdash (e + f) \cdot e = (e \cdot g) + (f \cdot g)}$$

$$\frac{}{Ax \vdash e \cdot (f \cdot g) = (e \cdot f) \cdot g} \qquad \frac{Ax \vdash f + e \cdot g \leqslant g}{Ax \vdash e^\star \cdot f \leqslant g} \qquad \frac{Ax \vdash f + g \cdot e \leqslant g}{Ax \vdash f \cdot e^\star \leqslant g}$$

(c) Constant-free Kleene algebra axioms.

Figure 1 Modular deduction system.

▶ **Lemma 9.** *For any subset of expressions S chosen from $\{\mathbb{T}, \mathbb{U}, \mathbb{E}\langle\mathcal{A}\rangle, \mathbb{E}\langle\mathcal{L}\rangle, \mathbb{E}^+, \mathbb{C}\}$, for any permutation π, and for any expression $e \in \mathbb{E}$, $e \in S \Leftrightarrow \pi \bowtie e \in S$. Furthermore, if e has the type A then $\pi \bowtie e : \pi(A)$, and if a is fresh for e then $\pi(a) \# \pi \bowtie e$.*

(Note the equivalence in the first point, which is why we keep a residual empty permutation when we apply a permutation to an expression of the shape $\langle\pi\rangle\, e$.)

2.3 Proofs

A generic framework for proofs

We now describe a generic framework for defining equational theories over $\mathbb{E}$. Given a relation $Ax \subseteq \mathbb{E} \times \mathbb{E}$, we define the judgement $Ax \vdash e = f$ to hold if it can be inferred in the system displayed in Figure 1 (where $Ax \vdash e \leqslant f$ is a shorthand for $Ax \vdash e + f = f$).

Notice that we have "hardwired" some laws of Kleene Algebra (KA) in this system, on the basis that they should hold for any reasonable equational system for Nominal Kleene Algebra. However, as we have two sets of constants, we cannot put inside the generic system the Kleene Algebra laws dealing with them. For instance when we consider expressions over the untyped syntax, the fact that $e \cdot 1 = e$ will be stated inside Ax. It is a simple matter to check that whatever Ax, the relation $Ax \vdash \_ = \_$ is an equivalence relation and $Ax \vdash \_ \leqslant \_$ is a preorder.

Sets of axioms

In Figures 2-6, we present a number of possible sets of axioms, which may be combined to axiomatise the different subsets we consider. All the axioms displayed here are implicitly universally quantified.

The first groups of axioms correspond to the axioms of KA for 1, declined in a typed and an untyped fashion. We then do the same for 0 and $\bot_A$, first with the KA axioms, and then for its interactions with $\langle\pi\rangle$, ν_a and w_a, always separating between the typed and untyped cases. These sets of axioms for constants are presented in Figures 2 and 3.

We then introduce sets of axioms to handle permutations. The axiom propagating w_a is set apart, as it only makes sense for typed expressions. This group is displayed in Figure 4. Notice that no law speaks about zeros, as it already has been dealt with in (3a).

The sets of axioms in Figure 5 are simple distributive laws of the restriction and weakening operators.

The next group, displayed in Figure 6, constitutes the core of the nominal theory of expressions. The untyped axioms are mostly the classic nominal axioms, taken from [6]. The only new axiom here is (6b), where we use syntactic permutations rather than semantic ones. The typed axioms are for the most part straightforward reformulations of the previous ones. Notice that in the typed case, we do not need to use freshness conditions, but rather typing statements. The last law of the set (6f) reflects the fact that for an untyped expression e, if $a \neq b$ then $a \# e \Leftrightarrow a \# \nu_b.e$.

2.4 Theories

A *theory* is given by a relation Ax, listing the axioms, and a set S from which we take expressions. As expressions may be typed or untyped, atomic or literate, clean or not and positive or not, there are 16 theories, listed in Table 1.

Notice that every subset of expressions mentioned in this table is associated with a single theory. In the following, for concision, we may refer to a theory by simply giving its base set. It is also worth mentioning that for every set S, the theories for $\mathbb{E}\langle\mathcal{L}\rangle \cap S$ and $\mathbb{E}\langle\mathcal{A}\rangle \cap S$ use the same set of axioms.

The theory $\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}$ corresponds precisely to the axiomatisation of NKA given in [6]. In our view, the best theory for defining the interpretation of a program would be $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}$, but a relational interpretation is best defined in $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{T}$.

A difficulty is that if we have a theory (S, Ax), with two expressions $e, f \in S$ such that $Ax \vdash e = f$, it may be the case that the proof uses expressions outside of S. This is generally what happens in systems with 0 (or $\bot_A$): if $e \notin S$ and $0 \in S$, then:

$$\frac{Ax \vdash 0 = 0 \cdot e \qquad Ax \vdash 0 \cdot e = 0}{Ax \vdash 0 = 0}$$

This is a bad property when one wants to prove results by structural induction on proofs. This phenomenon disappears with stable theories:

▶ **Definition 10.** A theory (S, Ax) is *stable* if for any expressions $e, f \in \mathbb{E}$ such that $Ax \vdash e = f$, $e \in S$ if and only if $f \in S$.

All of our positive theories (those included in $\mathbb{E}^+$) are stable.

$$\begin{aligned} e\cdot 1 &= e\\ 1\cdot e &= e\\ 1+e\cdot e^\star &\leqslant e^\star\\ 1+e^\star\cdot e &\leqslant e^\star \end{aligned}$$

(a) Untyped identity axioms.

$$\begin{aligned} e\cdot id_A &= e && \text{(if } e:A)\\ id_A\cdot e &= e && \text{(if } e:A)\\ id_A+e\cdot e^\star &\leqslant e^\star && \text{(if } e:A)\\ id_A+e^\star\cdot e &\leqslant e^\star && \text{(if } e:A) \end{aligned}$$

(b) Typed identity axioms.

Figure 2 Identity axioms.

$$\begin{aligned} e+0 &= e\\ e\cdot 0 &= e && \text{(if } e\in\mathbb{U})\\ 0\cdot e &= e && \text{(if } e\in\mathbb{U})\\ \langle\pi\rangle\,0 &= 0\\ \nu_a.0 &= 0 \end{aligned}$$

(a) Untyped zero axioms.

$$\begin{aligned} e+\bot_A &= e && \text{(if } e:A)\\ e\cdot\bot_A &= e && \text{(if } e:A)\\ \bot_A\cdot e &= e && \text{(if } e:A)\\ \langle\pi\rangle\,\bot_A &= \bot_{\pi(A)}\\ \nu_a.\bot_A &= \bot_{A\smallsetminus\{a\}} && \text{(if } a\in A)\\ w_a.\bot_A &= \bot_{A\cup\{a\}} && \text{(if } a\notin A) \end{aligned}$$

(b) Typed zero axioms.

Figure 3 Zero axioms.

$$\begin{aligned} \langle\pi\rangle\,(e+f) &= \langle\pi\rangle\,e+\langle\pi\rangle\,f\\ \langle\pi\rangle\,(e\cdot f) &= \langle\pi\rangle\,e\cdot\langle\pi\rangle\,f\\ \langle\pi\rangle\,(e^\star) &= (\langle\pi\rangle\,e)^\star\\ \langle\pi\rangle\,(\nu_a.e) &= \nu_{\pi(a)}.\,\langle\pi\rangle\,e\\ \langle\pi\rangle\,\langle\pi'\rangle\,e &= \langle\pi\circ\pi'\rangle\,e\\ \langle\pi\rangle\,1 &= 1\\ \langle\pi\rangle\,id_A &= id_{\pi(A)}\\ \langle\,()\,\rangle\,e &= e\\ \langle\pi\rangle\,a &= \pi\,(a) && \text{(if } a\in\mathcal{A})\\ \langle\pi\rangle\,l &= \pi\,(l) && \text{(if } l\in\mathcal{L}) \end{aligned}$$

(a) General permutation axioms.

$$\langle\pi\rangle\,(w_a.e) = w_{\pi(a)}.\,\langle\pi\rangle\,e$$

(b) Permutation vs. w_a

Figure 4 Permutation axioms.

$$\begin{aligned} w_a.\,(e+f) &= w_a.e+w_a.f\\ w_a.\,(e^\star) &= (w_a.e^\star)\\ w_a.\,(e\cdot f) &= w_a.e\cdot w_a.f\\ w_a.\,(id_A) &= id_{A\cup\{a\}} && \text{(if } a\notin A)\\ w_a.\,(w_b.e) &= w_b.\,(w_a.e) \end{aligned}$$

(a) Weakening.

$$\begin{aligned} \nu_a.\,(e+f) &= \nu_a.e+\nu_a.f\\ \nu_a.\,(\nu_b.e) &= \nu_b.\,(\nu_a.e) \end{aligned}$$

(b) Restriction.

Figure 5 Distributive laws of ν_a, w_a.

$$\nu_b.e = \nu_a.\,(a\ b) \bowtie e \qquad (\text{if } a \# e)$$

(a) Untyped α-conversion with $\bowtie$.

$$\nu_b.e = \nu_a.\langle\,(a\ b)\,\rangle\, e \qquad (\text{if } a \# e)$$

(b) Untyped α-conversion with $\langle\pi\rangle$.

$$\nu_b.e = \nu_a.\,(a\ b) \bowtie e \qquad (\text{if } \nu_b.e : A \text{ and } a \notin A)$$

(c) Typed α-conversion with $\bowtie$.

$$\nu_b.e = \nu_a.\langle\,(a\ b)\,\rangle\, e \qquad (\text{if } \nu_b.e : A \text{ and } a \notin A)$$

(d) Typed α-conversion with $\langle\pi\rangle$.

$$\begin{aligned} \nu_a.e &= e && (\text{if } a \# e)\\ \nu_a.f \cdot e &= \nu_a.\,(f \cdot e) && (\text{if } a \# e)\\ e \cdot \nu_a.f &= \nu_a.\,(e \cdot f) && (\text{if } a \# e) \end{aligned}$$

(e) Untyped nominal axioms.

$$\begin{aligned} \nu_a.w_a.e &= e && (\text{if } \nu_a.w_a.e : A)\\ (\nu_a.f) \cdot e &= \nu_a.\,(f \cdot w_a.e) &&\\ e \cdot (\nu_a.f) &= \nu_a.\,(w_a.e \cdot f) &&\\ \nu_b.w_a.e &= w_a.\nu_b.e && (\text{if } a \neq b) \end{aligned}$$

(f) Typed nominal axioms.

Figure 6 Nominal axioms.

Table 1 Theories.

Name	Set	Axioms							
NKAmpu	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$	(2a)		(4a)			(5b)	(6b)	(6e)
NKAspu	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$								
NKAnmpu	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}$	(2a)					(5b)	(6a)	(6e)
NKAnspu	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}$								
NKAmu	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U}$	(2a)	(3a)	(4a)			(5b)	(6b)	(6e)
NKAsu	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U}$								
NKAnmu	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}$	(2a)	(3a)				(5b)	(6a)	(6e)
NKAnsu	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}$								
NKAmpt	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{T}$	(2b)		(4a)	(4b)	(5a)	(5b)	(6d)	(6f)
NKAspt	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{T}$								
NKAnmpt	$\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{T} \cap \mathbb{C}$	(2b)				(5a)	(5b)	(6c)	(6f)
NKAnspt	$\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{T} \cap \mathbb{C}$								
NKAmt	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{T}$	(2b)	(3b)	(4a)	(4b)	(5a)	(5b)	(6d)	(6f)
NKAst	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{T}$								
NKAnmt	$\mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{T} \cap \mathbb{C}$	(2b)	(3b)			(5a)	(5b)	(6c)	(6f)
NKAnst	$\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{T} \cap \mathbb{C}$								

$$\begin{tikzcd}
& \mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{U} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{U} \arrow[ll, shift right] \arrow[rr] \arrow[dl, <->] \arrow[dd] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{U} \arrow[rr, shift left] \arrow[dl, <->] \arrow[dd] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{U} \arrow[ll, dashed, shift left] \\
\mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[ll, shift right] \arrow[dd] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[rr, shift left] \arrow[dd] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{U}\cap\mathbb{C} \arrow[ll, dashed, shift left] & \\
& \mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{T} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{T} \arrow[ll, shift right] \arrow[rr] \arrow[dl, <->] \arrow[uu, dashed] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{T} \arrow[rr, shift left] \arrow[dl, <->] \arrow[uu, dashed] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{T} \arrow[ll, dashed, shift left] \\
\mathbb{E}\langle\mathcal{A}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[rr, dashed, shift right] & & \mathbb{E}^+\langle\mathcal{A}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[ll, shift right] \arrow[uu, dashed] & & \mathbb{E}^+\langle\mathcal{L}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[rr, shift left] \arrow[uu, dashed] & & \mathbb{E}\langle\mathcal{L}\rangle\cap\mathbb{T}\cap\mathbb{C} \arrow[ll, dashed, shift left] &
\end{tikzcd}$$

Figure 7 The two cubes as sets.

3 Ordering theories

3.1 Definitions

We define two preorders to compare theories. The first one is the strongest one:

▸ **Definition 11** (Embedding preorder)**.** A theory (S, Ax) *can be embedded into* (S', Ax'), written $(S, Ax) \preccurlyeq (S', Ax')$ if there is a function ϕ such that for any $e \in S$, $\phi(e) \in S'$, and for any $e, f \in S$, $Ax \vdash e = f \Leftrightarrow Ax' \vdash \phi(e) = \phi(f)$. In that case we say that ϕ *is an embedding of* (S, Ax) *into* (S', Ax').

When a theory can be embedded into a second one, then every model of the latter one gives rise to a model former one. However, some intuitively equivalent theory cannot be compared using this preorder. For instance, $\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{T}$ cannot be embedded into $\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$. Indeed, while the typed expressions $id_{\{a\}}$ and $id_{\{a,b\}}$ are not equal (they have different types), they have the same untyped behaviour and both of them should be mapped to the untyped constant 1. To this end, we introduce a slightly weaker preorder:

▸ **Definition 12** (Reduction preorder)**.** A theory (S, Ax) *reduces to* (S', Ax'), which we denote by $(S, Ax) \ll (S', Ax')$, if for any pair $(e, f) \in S \times S$ there is a pair of expressions $(e', f') \in S'$ such that $Ax \vdash e = f \Leftrightarrow Ax' \vdash e' = f'$.

▸ **Lemma 13.** *If* $(S, Ax) \ll (S', Ax')$ *and if* (S', Ax') *is decidable, then so is* (S, Ax)*.*

▸ Remark. This lemma assumes an effective proof of $(S, Ax) \ll (S', Ax')$: there must be a way to build the pair e', f' from the pair e, f. Our (Coq) proofs below have this property.

3.2 Embeddings

We summarise the results we obtained using Coq on Figure 7. (The scripts are available online [1]). A plain arrow is drawn between two theories if the source of the arrow can be embedded into the target of the arrow, and a dashed arrow when the source reduces to the target. Thanks to the decidability result for $\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}$ [5], this ensures that all atomic theories are decidable.

We discuss in more details how we obtained some of these results.

3.2.1 Reducing to positive fragments

The first step consists in getting rid of the constants 0 and $\bot_A$, so that we can focus on stable theories (Definition 10). We only present here the untyped case. In other words we choose a theory (S, Ax), with S taken from the set $\{\mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U}, \mathbb{E}\langle\mathcal{A}\rangle \cap \mathbb{U} \cap \mathbb{C}, \mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U}, \mathbb{E}\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \mathbb{C}\}$, the corresponding positive theory being $(S \cap \mathbb{E}^+, Ax \smallsetminus (3a))$.

▸ **Definition 14.** If e is an untyped expression, $\mathit{extract}(e)$ is the unique normal form of e with respect to the following confluent rewriting system:

$$e + 0 \to e \qquad 0 + f \to f \qquad e \cdot 0 \to 0 \qquad 0 \cdot f \to 0 \qquad \nu_a.0 \to 0 \qquad \langle\pi\rangle\, 0 \to 0 \qquad 0^\star \to 1.$$

The interesting property of this function is that if $Ax \vdash e = 0$, then $\mathit{extract}(e)$ is syntactically equal to 0, and $\mathit{extract}(e) \in \mathbb{E}^+$ otherwise. Furthermore, for every $e \in S$, $Ax \vdash \mathit{extract}(e) = e$. The formal proof then relies on two key observations:
1. If $(e, f) \in Ax \smallsetminus (3a)$, then $Ax \smallsetminus (3a) \vdash \mathit{extract}(e) = \mathit{extract}(f)$.
2. If $(e, f) \in (3a)$, then $\mathit{extract}(e) = \mathit{extract}(f)$.

This allows to prove by induction on proofs that:

$$Ax \vdash e = f \Rightarrow Ax \smallsetminus (3a) \vdash \mathit{extract}(e) = \mathit{extract}(f).$$

Because the positive axiomatisation is included in Ax, we also get:

$$Ax \smallsetminus (3a) \vdash \mathit{extract}(e) = \mathit{extract}(f) \Rightarrow Ax \vdash \mathit{extract}(e) = \mathit{extract}(f).$$

The fact that $\mathit{extract}(e)$ is provably equal to e with the axioms Ax allows to close the proof of equivalence, with the entailment:

$$Ax \vdash \mathit{extract}(e) = \mathit{extract}(f) \Rightarrow Ax \vdash e = f.$$

However, if $Ax \vdash e = 0$ then $\mathit{extract}(e) \notin \mathbb{E}^+$. This means that we cannot directly use $\mathit{extract}()$ as an embedding between theories. We obtain the reduction as follows: when given the pair e, f, we compute $\mathit{extract}(e)$ and $\mathit{extract}(f)$. If both of these are equal to zero, then when map the pair to equal positive expressions, say $1, 1$. If both of them are non-zero, then we produce $\mathit{extract}(e), \mathit{extract}(f)$. Otherwise we produce different positive expressions, say $1, a$ in the atomic case and $1, l$ in the literate case.

3.2.2 From presheaves to freshness, and back

Let (S, Ax) be a positive untyped theory, meaning $S \subseteq \mathbb{E}^+ \cap \mathbb{U}$, and (S', Ax') be the corresponding positive typed theory. We show here how to transport S into S', and vice versa. This corresponds to the vertical arrows on Figure 7. The key tools in this case are the erasure and retyping functions.

▸ **Definition 15** (Erasure). The *erasure* of $e \in \mathbb{T}$, written $\lfloor e \rfloor$, is the expression obtained from e by removing all weakenings (w_a), and replacing all id_A with 1 and all $\bot_A$ with 0.

▸ **Lemma 16.** *If $e \in S'$ then $\lfloor e \rfloor \in S$, and if $e : A$ then $\natural(\lfloor e \rfloor) \subseteq A$.*

▸ **Definition 17** (Retyping). Let $e \in \mathbb{U}$, we define the retyping of e, written $\lceil e \rceil$, by structural induction:

$$\begin{array}{rlrlrll}
\lceil 0 \rceil &:= \bot_\varnothing & \lceil 1 \rceil &:= id_\varnothing & \lceil e + f \rceil &:= w_{\natural(f) \smallsetminus \natural(e)} \cdot \lceil e \rceil + w_{\natural(e) \smallsetminus \natural(f)} \cdot \lceil f \rceil & \\
\lceil a \rceil &:= a & \lceil l \rceil &:= l & \lceil e \cdot f \rceil &:= w_{\natural(f) \smallsetminus \natural(e)} \cdot \lceil e \rceil \cdot w_{\natural(e) \smallsetminus \natural(f)} \cdot \lceil f \rceil & \\
\lceil e^\star \rceil &:= \lceil e \rceil^\star & & & \lceil \nu_a.e \rceil &:= \nu_a \cdot \lceil e \rceil & (\text{if } a \in \natural(e)) \\
\lceil \langle\pi\rangle\, e \rceil &:= \langle\pi\rangle \lceil e \rceil & & & \lceil \nu_a.e \rceil &:= \nu_a.w_a \cdot \lceil e \rceil & (\text{if } a \notin \natural(e))
\end{array}$$

The notation $w_A.e$ is justified by the law $w_a.w_b.e = w_b.w_a.e$, holding in every typed theory.

▸ **Lemma 18.** *$e \in S$ entails $\lceil e \rceil \in S'$. Furthermore, $\lceil e \rceil$ has the type $\natural(e)$.*

These functions allow one to go back and forth between S and S':

▸ **Lemma 19.** *If $e \in S$, then $e = \lfloor \lceil e \rceil \rfloor$. If $e : A$, then:*

$$(6f)\ (5a)\ (4b) \vdash e = w_{A \smallsetminus \natural(\lfloor e \rfloor)} \cdot \lceil \lfloor e \rfloor \rceil .$$

Furthermore, if $S \subseteq \mathbb{C}$, we may remove the axiom $(4b)$.

From this lemma, we obtain that the retyping function is an embedding of S into S'. But it also shows a problem for the other direction. For instance the expressions e and $w_a.e$ have different types, and are thus different, but they will be mapped to the same expression. For this reason, we cannot use the erasure function to embed S' into S.

Nevertheless, we can use it to show that S' is simpler than S. Given a pair of expressions $e, f \in S'$, if e and f have the same type, then we produce the pair $\lfloor e \rfloor, \lfloor f \rfloor$ which is equiprovable. If it is not the case, we purposely produce different expressions, as in the previous section.

3.2.3 From atomic to literate

We assume there is an atom $\alpha \in \mathcal{A}$ and an element $\lambda \in \mathcal{L}$ with $\natural(\lambda) = \{\alpha\}$. We show the transformation from $\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$ to $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$, which corresponds to the central horizontal top arrow on Figure 7. Let NKApu be the set of axioms $(2a), (4a), (5b), (6b), (6e)$ corresponding to the theory of these sets.

▸ **Definition 20** (From atoms to letters.)**.** Given an expression $e \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$, we obtain the expression $\downharpoonleft e \downharpoonright \in \mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$ by replacing every atomic variable a by $\langle (a\ \alpha) \rangle \lambda$. We write $\downharpoonleft \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \downharpoonright$ for the set of expressions $f \in \mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$, such that there is an expression $e \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$ such that $f = \downharpoonleft e \downharpoonright$.

For any expression e, $e \in \downharpoonleft \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \downharpoonright$ if and only if each literate variable in e has the identifier x. It is also worth noting that $\downharpoonleft \_ \downharpoonright$ preserves freshness: $a \# e \Leftrightarrow a \# \downharpoonleft e \downharpoonright$. As for typed and untyped expressions, we define an inverse operation.

▸ **Definition 21** (Going back)**.** The inverse operation is only defined on literate expressions whose variables carry the identifier x, and thus have a singleton support. The expression $\upharpoonleft e \upharpoonright$ is then obtained by replacing every variable by the single atom in its support.

The function $\upharpoonleft \_ \upharpoonright$ is the inverse of $\downharpoonleft \_ \downharpoonright$, $\downharpoonleft \_ \downharpoonright$ preserves NKApu-equality, and $\upharpoonleft \_ \upharpoonright$ preserves NKApu-equality on the image of $\downharpoonleft \_ \downharpoonright$.

▸ **Lemma 22.** *$\forall e \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$, NKApu $\vdash e = \upharpoonleft \downharpoonleft e \downharpoonright \upharpoonright$.*

▸ **Lemma 23.** *$\forall e, f \in \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$, NKApu $\vdash e = f \Rightarrow$ NKApu $\vdash \downharpoonleft e \downharpoonright = \downharpoonleft f \downharpoonright$.*

▸ **Lemma 24.** *$\forall e, f \in \mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U} \cap \downharpoonleft \mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U} \downharpoonright$, NKApu $\vdash e = f \Rightarrow$ NKApu $\vdash \upharpoonleft e \upharpoonright = \upharpoonleft f \upharpoonright$.*

By putting all together, we obtain that $\downharpoonleft \_ \downharpoonright$ is an embedding of $\mathbb{E}^+\langle\mathcal{A}\rangle \cap \mathbb{U}$ into $\mathbb{E}^+\langle\mathcal{L}\rangle \cap \mathbb{U}$.

4 Relational interpretation of literate expressions

Our main motivation for developing the typed syntax was to define a relational interpretation of expressions. As explained in the introduction, the classical way of interpreting a program as a relation is to consider memory states as functions, associating values to memory cells. A program is then simply a relation between memory states. Furthermore, in most high level programming languages, one cannot access every part of the memory: a variable should be declared before it is used. There are also constructs allowing one to declare a local variable, which is hidden from the rest of the program. Both of these considerations can be encoded by considering functions with a finite domain: the set of memory locations that are visible in the current scope.

Let us be more precise. Consider that the set $\mathcal{A}$ of atoms corresponds to memory locations, and that locations may contain values from an arbitrary set $\mathcal{V}$. A memory state of domain $A \in \wp_f(\mathcal{A})$ is then a function from A to $\mathcal{V}$, and an expression of type A will be interpreted as a binary relation over memory states of domain A (whence the presheaf structure).

Regular operations are interpreted using the standard operations on binary relations; in particular, id_A is interpreted as the identity relation on $\mathcal{V}^A$. To interpret letters, we need to fix an equivariant map ϕ that assign to a letter x a relation between memory states with domain $\natural(x)$. Several choices are possible for the operations of restriction (ν_a) and weakening (w_a), yielding slightly different theories. Here is a possibility which gives rise to a model of our theory: if R is a relation over $\mathcal{V}^A$, then we define

$$\nu_a.R := \{(f\restriction_A, g\restriction_A) \mid (f,g) \in R\} \quad ; \qquad \text{(if } a \in A)$$
$$w_a.R := \{(f,g) \mid (f\restriction_A, f\restriction_A) \in R \text{ and } f(a) = g(a)\} \quad . \qquad \text{(if } a \notin A)$$

(Where $f\restriction_A$ is the restriction of a function $f \in \mathcal{V}^B$ for some superset B of A.)

Note that this model is not free: for all relations R, S we have $\nu_a.(R \cdot S) \subseteq (\nu_a.R)\cdot(\nu_a.S)$, which is not an inequation provable from the axioms.

Example.

Consider the program $\texttt{swap}(x,y)$ that exchanges the contents of the variables x and y. The natural implementation of this program is the following: $\texttt{let}\ t\ \texttt{in}\ t \leftarrow x; x \leftarrow y; y \leftarrow t$.

The instruction $\texttt{x} \leftarrow \texttt{y}$ may be represented by a nominal element $\texttt{assign}(\texttt{x},\texttt{y})$ with support $\{\texttt{x},\texttt{y}\}$, and such that $\pi(\texttt{assign}(\texttt{x},\texttt{y})) = \texttt{assign}(\pi(\texttt{x}),\pi(\texttt{y}))$. Accordingly, the program $\texttt{swap}$ is represented by the following expression, where the location is hidden using a top-level restriction.

$$\nu_{\texttt{t}}.(\texttt{assign}(\texttt{t},\texttt{x})\cdot\texttt{assign}(\texttt{x},\texttt{y})\cdot\texttt{assign}(\texttt{y},\texttt{t})) \quad .$$

Alternatively, one can obtain an expression with a single letter using explicit permutations: let a_1 and a_2 be two atoms, and set $\mathbb{a} := \texttt{assign}(a_1,a_2)$. The instruction $\texttt{x} \leftarrow \texttt{y}$ may be represented by $\langle(\texttt{x}\ a_1)\ (\texttt{y}\ a_2)\rangle\mathbb{a}$, and the program swap by

$$\nu_t.(\langle(\texttt{t}\ a_1)\ (\texttt{x}\ a_2)\rangle\mathbb{a})\cdot(\langle(\texttt{x}\ a_1)\ (\texttt{y}\ a_2)\rangle\mathbb{a})\cdot(\langle(\texttt{y}\ a_1)\ (\texttt{t}\ a_2)\rangle\mathbb{a}).$$

5 Future work

We leave two questions for future work. First, is it possible to reduce the literate theory of nominal Kleene algebra to that of atomic nominal Kleene algebra? If not, is there a free language theoretic model for which we could obtain decidability?

Second, is there a free relational model for our literate theory?

Acknowledgements. We would like to thank Daniela Petrisan and Alexandra Silva for the discussions that have led to this work.

References

1 Paul Brunet. Web appendix to this abstract, 2016. `http://perso.ens-lyon.fr/paul.brunet/nka`.

2 Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 214–224. IEEE, 1999.

3 Murdoch James Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Foundations of Software Science and Computational Structures, FoSSaCS 2011*, pages 365–380. Springer, 2011.

4 Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. `doi:10.1145/256167.256195`.

5 Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal Kleene Coalgebra. In *Automata, Languages, and Programming, ICALP 2015*, pages 286–298. Springer, 2015.

6 Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and incompleteness in Nominal Kleene Algebra. In *Relational and Algebraic Methods in Computer Science, RAMiCS 2015*, pages 51–66. Springer, 2015.

7 Andrew M Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57. Cambridge University Press, 2013.

On the Implicit Graph Conjecture

Maurice Chandoo

Leibniz Universität Hannover, Theoretical Computer Science, Appelstr. 4, 30167 Hannover, Germany
chandoo@thi.uni-hannover.de

Abstract

The implicit graph conjecture states that every sufficiently small, hereditary graph class has a labeling scheme with a polynomial-time computable label decoder. We approach this conjecture by investigating classes of label decoders defined in terms of complexity classes such as P and EXP. For instance, GP denotes the class of graph classes that have a labeling scheme with a polynomial-time computable label decoder. Until now it was not even known whether GP is a strict subset of GR where R is the class of recursive languages. We show that this is indeed the case and reveal a strict hierarchy akin to classical complexity. We also show that classes such as GP can be characterized in terms of graph parameters. This could mean that certain algorithmic problems are feasible on every graph class in GP. Lastly, we define a more restrictive class of label decoders using first-order logic that already contains many natural graph classes such as forests and interval graphs. We give an alternative characterization of this class in terms of directed acyclic graphs. By showing that some small, hereditary graph class cannot be expressed with such label decoders a weaker form of the implicit graph conjecture could be disproven.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases adjacency labeling scheme, complexity classes, diagonalization, logic

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.23

1 Introduction

The class of interval graphs has at most $2^{\mathcal{O}(n \log n)}$ graphs on n vertices. Neither adjacency matrices nor lists are asymptotically space optimal to represent this class since only $\mathcal{O}(n \log n)$ bits should be used to store a graph on n vertices. However, due to the geometrical representation of this class every vertex of an interval graph can be assigned an interval on a discrete line with $2n$ points. Stated differently, every vertex can be labeled with two numbers between 1 and $2n$ and adjacency of two vertices can be determined by comparing the four numbers. Storing two such numbers for all n vertices requires $n \log 4n^2$ bits and thus is asymptotically optimal. Labeling schemes, also known as implicit representation, generalize this kind of representations by allowing to store a $\mathcal{O}(\log n)$ long binary label at every vertex such that adjacency between two vertices can be determined by running an algorithm on the two labels. We investigate what graph classes can or cannot be represented in such a way when restricting the computational complexity of the function that determines adjacency, also called label decoder.

Let us call a graph class that has at most $2^{\mathcal{O}(n \log n)}$ graphs on n vertices small. A simple counting argument shows that only small graph classes can have labeling schemes. The first question that springs to mind is whether all small graph classes have a labeling scheme. This is not the case as Spinrad shows by giving a small, non-hereditary graph class as counter-example in [12, Thm. 7]. Now, the question becomes whether all small, hereditary graph classes have a labeling scheme; this is known as implicit graph conjecture(IGC). This question was already posed more than two decades ago in 1992 by Kannan, Naor and Rudich [7] and has been brought up again by Spinrad [12]. But despite being such an old question not much is known in this regard. One such result is: every tiny, hereditary graph class admits a labeling scheme with labels of constant length [10]. Tiny means that there exist $n_0 \in \mathbb{N}$ and $k < \frac{1}{2}$ such that the class has at most $2^{kn \log n}$ labeled graphs on n vertices for all $n \geq n_0$. This follows from the insight that every tiny, hereditary graph class has only a constant number of twin-free graphs, which makes such classes rather uninteresting. On the other hand, small, hereditary graph classes such as planar or circular-arc graphs can have a rich structure. Candidates for the IGC, i.e. small, hereditary graph classes for which no labeling scheme is known, are line segment graphs, (unit) disk graphs, k-dot product graphs and k-sphere graphs [3, 9, 6]. It is interesting to note that the obvious labeling schemes for line segment and disk graphs using their geometrical representation does not work since coordinates and radii can require an exponential number of bits [9] unlike in the case of interval graphs.

A different aspect of labeling schemes that has been extensively studied are lower and upper bounds on the label length, i.e. the constant lurking in $\mathcal{O}(\log n)$, which is related to small universal graphs. A recent result shows that graphs of bounded arboricity k admit a labeling scheme with optimal label length $k \log n + \mathcal{O}(1)$ [1]. Besides, labeling schemes can be generalized in various ways. One variant are distance labeling schemes where one wants to infer the distance between two vertices given their labels [4]. In [8] it was proposed to consider multiple labels instead of only two. Another natural extension is to consider labeling schemes for graph classes that are not small by allowing longer labels while still maintaining the condition of being asymptotically space optimal [12]. However, here we shall investigate the original variant of this concept.

Our results. For a complexity class A let GA denote the class of graph classes that have a labeling scheme where the label decoder can be computed in A (precise definitions follow). In general, we investigate how choosing various complexity classes for A affects the class of graph classes GA that can be represented and how such classes of graph classes can be characterized. In section two we argue that $\mathsf{G}k\mathsf{EXP} \subsetneq \mathsf{G}(k+1)\mathsf{EXP}$ for all $k \geq 1$ by giving a diagonalization argument. A related result for distance labeling schemes can be found in section four of [4]. Additionally, we consider the graph class(es) constructed in the proof as candidate for the implicit graph conjecture. In the third section we show that for every reasonable complexity class A the class of graph classes GA can be exactly characterized in terms of a graph parameter. By graph parameter we mean a graph property which maps to the natural numbers such as clique number or tree width. Given such a characterizing graph parameter λ_{A} for GA the question of whether a graph class lies in GA then is equivalent to asking whether it is bounded by λ_{A}. Another consequence of such a characterization is that if for example determining the existence of a Hamiltonian cycle is fixed-parameter tractable under the parameterization λ_{A} then for every graph class in GA this problem can be decided in polynomial time. This means the existence of a labeling scheme can have algorithmic implications. In the last section we define a class of label decoders FO via first-order logic formulas with arithmetic, i.e. comparing order, addition and multiplication. Our motivation for introducing this class of label decoders is that the Turing machine model seems too strong to obtain lower bounds. We give upper bounds on the expressiveness of GFO and its quantifier-free variant. Even if quantifiers, addition and multiplication are disallowed the resulting class $\mathsf{GFO}_{\mathrm{qf}}(<)$ already contains many interesting graph classes such as forests, planar graphs and k-interval graphs(also known as multiple interval graphs [2]). Lastly, we describe an alternative characterization of $\mathsf{GFO}_{\mathrm{qf}}(<)$ in terms of directed acyclic graphs.

Terminology. Let $[n] = \{1, 2, \dots, n\}$. We write $\log n$ instead of $\lceil \log_2 n \rceil$ and $\exp(n) = 2^n$. Let $\exp^i(n) = \exp(\exp^{i-1}(n))$ for $i \geq 1$ and $\exp^0(n) = n$. The domain and image of a function f are abbreviated by $\mathrm{dom}(f)$ and $\mathrm{Im}(f)$ respectively. We consider only graphs without multiple edges and self-loops and regard undirected graphs as special case of directed ones. For a sequence of graphs $G, G_1, \dots, G_m$ on the same vertex set V let us say G is the edge-union of $G_1, \dots, G_m$ if $E(G) = \cup_{i \in [m]} E(G_i)$. For two graphs G, H we write $G \cong H$ to indicate that they are isomorphic. We speak of G as unlabeled graph to emphasize that we talk about the isomorphism class of G rather than a specific adjacency matrix of G. A graph class is a set of unlabeled graphs, i.e. closed under isomorphism. A graph class is hereditary if it is closed under taking induced subgraphs. Let $\mathcal{G}$ be the class of all graphs and $\mathcal{G}_n$ is the class of all graphs on n vertices. A language is a set of words over the binary alphabet $\{0, 1\}$. We use complexity class as informal term to mean a set of languages defined in terms of computation and assume that it is countable. The deterministic Turing machine (TM) is our model of computation when talking about time as resource bound. Let L denote the complexity class logspace, PH is the polynomial-time hierarchy, R is the class of recursive languages and $k\mathsf{EXP}$ is the class of languages computable in time $\exp^k(n^{\mathcal{O}(1)})$ for $k \geq 0$, e.g. $0\mathsf{EXP} = \mathsf{P}$. Let $\mathsf{ALL} = \mathcal{P}(\{0, 1\}^*)$ be the class of all languages.

▶ **Definition 1** (Labeling scheme). A label decoder F is a binary relation over words, i.e. $F \subseteq \{0, 1\}^* \times \{0, 1\}^*$. A labeling scheme is a tuple $S = (F, c)$ where F is a label decoder and $c \in \mathbb{N}$ is the label length. A graph G on n vertices is in the class of graphs spanned by S, denoted by $G \in \mathrm{gr}(S)$, if there exists a labeling $\ell \colon V(G) \to \{0, 1\}^{c \log n}$ such that for all $u, v \in V(G)$:

$$(u, v) \in E(G) \Leftrightarrow (\ell(u), \ell(v)) \in F$$

We say a graph class $\mathcal{C}$ is represented by (or has) a labeling scheme S if $\mathcal{C} \subseteq \mathrm{gr}(S)$.

▶ **Definition 2.** A language $L \subseteq \{0, 1\}^*$ induces a label decoder F_L where for all $x, y \in \{0, 1\}^*$ with $|x| = |y|$ it holds that $(x, y) \in F_L \Leftrightarrow xy \in L$.

Let A be a set of languages and $k \in \mathbb{N}$. A graph class $\mathcal{C}$ is in $\mathsf{G}_k\mathsf{A}$ if there exists a language $L \in \mathsf{A}$ such that $\mathcal{C}$ is represented by (F_L, c) for some $c \leq k$. Analogously, $\mathcal{C}$ is in GA if $\mathcal{C}$ is in $\mathsf{G}_k\mathsf{A}$ for some $k \in \mathbb{N}$.

A class of the form $\mathsf{G}\cdot$ is trivially closed under taking subsets, i.e. if $\mathcal{C} \subseteq \mathcal{C}'$ and $\mathcal{C}' \in \mathsf{G}\cdot$ then $\mathcal{C} \in \mathsf{G}\cdot$. It follows that $\mathsf{G}\cdot$ is closed under intersection as well. However, no $\mathsf{G}\cdot$ is closed under complement since the complement of a small graph class is not small. For many complexity classes such as L and P it is also not hard to show that the classes GL and GP are closed under union.

Here is an example of a language L whose label decoder F_L represents interval graphs: $x_1x_2y_1y_2 \in L$ iff x_1, x_2, y_1, y_2 are binary strings of equal length and neither $x_2 < y_1$ nor $y_2 < x_1$ holds where $<$ denotes the lexicographical order. Then the labeling scheme $S = (F_L, 2)$ represents interval graphs. Since L can be computed in logspace it follows that interval graphs are in $\mathsf{G}_2\mathsf{L}$.

Using our terminology the implicit graph conjecture can be rephrased as:

▶ **Conjecture 3** (IGC,[7]). *Let* H *denote the set of all small, hereditary graph classes.*

$$\mathsf{GP} \cap \mathrm{H} = \mathsf{GALL} \cap \mathrm{H} = \mathrm{H}$$

As of now it is far from clear whether even the second equality holds, i.e. can every small, hereditary graph class be represented by some labeling scheme, leaving computability issues aside? This is a graph-theoretic question dealing with the existence of polynomial-sized universal graphs that should be addressed before one can expect to prove the implicit graph conjecture.

2 Hierarchy of Implicit Representations

In the previous section we have seen that every language L can be interpreted as label decoder F_L. Therefore a set of languages A can be understood as set of label decoders and GA denotes the set of graph classes that can be represented by a labeling scheme (F, c) with $F \in \mathsf{A}$ and $c \in \mathbb{N}$. Inclusion carries over to this setting meaning $\mathsf{A} \subseteq \mathsf{B}$ implies $\mathsf{GA} \subseteq \mathsf{GB}$. For separations, however, this is not true, i.e. there exist A, B with $\mathsf{A} \subsetneq \mathsf{B}$ and $\mathsf{GA} = \mathsf{GB}$. Spinrad remarks that it is not known whether restricting the label decoder to be computable in polynomial time versus requiring it to be simply computable makes a difference in terms of the graph classes that can be represented [12, p. 22]. We resolve this question by applying diagonalization, which yields many of the separations known in the classical setting. For the sake of clarity we prove the following class of separations which we deem most interesting with respect to the IGC since it yields the smallest class($\mathsf{G2EXP}$) that can be separated from GP by this argument:

▶ **Theorem 4.** $\mathsf{G}k\mathsf{EXP} \subsetneq \mathsf{G}(k+1)\mathsf{EXP}$ *for all* $k \geq 1$.

The basic idea behind the proof of this statement is the following diagonalization argument. Let $\mathsf{A} = \{F_1, F_2, \dots\}$ be a set of label decoders. Then a labeling scheme in GA can be seen as pair of natural numbers, one for the label decoder and one for the label length. Let $\tau : \mathbb{N} \to \mathbb{N}^2$ be a surjective function and $S_{\tau(x)} = (F_y, z)$ with $\tau(x) = (y, z)$. It follows that for every labeling scheme S in GA there exists an $x \in \mathbb{N}$ such that $S = S_{\tau(x)}$. The following graph class cannot be in GA:

$$G \in \mathcal{C}_{\mathsf{A}} \Leftrightarrow G \text{ is the smallest graph on } n = |V(G)| \text{ vertices s.t. } G \notin \mathrm{gr}(S_{\tau(n)})$$

where smallest is meant w.r.t. some order such as the lexicographical one. Note that the order must be for unlabeled graphs. However, an order for labeled graphs can be easily adopted to unlabeled ones. Assume $\mathcal{C}_{\mathsf{A}}$ is in GA via the labeling scheme S. There exists an $n \in \mathbb{N}$ such that $S = S_{\tau(n)}$ and it follows that $\mathcal{C}_{\mathsf{A}}$ contains a graph on n vertices that cannot be in S per definition, contradiction. Then it remains to show that $\mathcal{C}_{\mathsf{A}}$ is in the class that we wish to separate from GA.

For the remainder of this section we formalize this idea in three steps. First, we state the requirements for a pairing function τ and show that such a function exists. We continue by arguing that the diagonalization graph class $\mathcal{C}_{\mathsf{A}}$ is not contained GA. In the last step we construct a label decoder for $\mathcal{C}_{k\mathsf{EXP}}$ and show that it can be computed in $(k+1)\mathsf{EXP}$.

▶ **Definition 5.** A surjective function $\tau : \mathbb{N} \to \mathbb{N}^2$ is an admissible pairing if

1. $|\tau^{-1}(y, z)|$ is infinite for all $y, z \in \mathbb{N}$,
2. $\tau_y(x), \tau_z(x) \in \mathcal{O}(\log x)$ with $\tau(x) = (\tau_y(x), \tau_z(x))$,
3. $\tau(x)$ is undefined if x is not a power of two, and
4. τ is computable in polynomial time given its input in unary.

Note, that a graph on n vertices gets assigned labels of the same length as a graph on m vertices whenever $\log n = \log m$ (rounded up). The third condition prevents this from happening, i.e. for all $G \neq H \in \mathcal{C}_{\mathsf{A}}$ it holds that their vertices must have labels of different length.

▶ **Lemma 6.** *There exists an admissible pairing function.*

Proof. Consider the function $\tau(x) = (y,z)$ iff $x = 2^{2^y \cdot 3^z \cdot 5^w}$ for some $w \geq 0$. ◀

▶ **Definition 7.** Let A be a set of languages, $\prec$ an order on unlabeled graphs and τ an admissible pairing. The diagonalization graph class of A is defined as:

$$\mathcal{C}_{\mathsf{A}} = \bigcup_{n \in \operatorname{dom}(\tau)} \left\{ G \in \mathcal{G}_n \;\middle|\; G \text{ is the smallest graph w.r.t. } \prec \text{ not in } \operatorname{gr}(S_{\tau(n)}) \right\}$$

When we consider the diagonalization graph class of a set of languages we assume the lexicographical order for $\prec$ and the function given in the proof of Lemma 6 for τ.

▶ **Lemma 8.** *For every countable set of languages* A *it holds that* $\mathcal{C}_{\mathsf{A}} \notin \mathsf{GA}$.

Proof. As argued in the paragraph after Theorem 4 it holds that for any labeling scheme S in GA there exists a graph G that is in $\mathcal{C}_{\mathsf{A}}$ but not in $\operatorname{gr}(S)$ and thus this lemma holds. Since the labeling scheme S is in GA there exists an $n \in \mathbb{N}$ such that $S = S_{\tau(n)}$ where $S_{\tau(n)} = (F_y, z)$, $\tau(n) = (y,z)$ and $\mathsf{A} = \{F_1, F_2, \dots\}$. Due to the fact that $|\tau^{-1}(y,z)|$ is infinite it follows that there exists an arbitrarily large $n \in \mathbb{N}$ such that $S = S_{\tau(n)}$. For $\mathcal{C}_{\mathsf{A}} \setminus \operatorname{gr}(S)$ to be non-empty it must hold that $\operatorname{gr}(S_{\tau(n)})$ does not contain all graphs on n vertices. By choosing n to be sufficiently large this is guaranteed since $\operatorname{gr}(S_{\tau(n)})$ is a small graph class. ◀

To show that $\mathcal{C}_{\mathsf{A}}$ is in some class GB we need to define a labeling scheme $S_{\mathsf{A}} = (F_{\mathsf{A}}, 1)$ that represents $\mathcal{C}_{\mathsf{A}}$ and consider the complexity of computing its label decoder.

▶ **Definition 9.** Let A be a set of languages. For $G \in \mathcal{C}_{\mathsf{A}}$ let G_0 denote the smallest labeled graph with $G_0 \cong G$. We define the label decoder F_{A} as follows. For every $m \in \mathbb{N}$ such that there exists $G \in \mathcal{C}_{\mathsf{A}}$ with $|V(G)| = 2^m$ and for all $x, y \in \{0,1\}^m$ let

$$(x,y) \in F_{\mathsf{A}} \Leftrightarrow (x,y) \in E(G_0)$$

It can be assumed that G_0 has $\{0,1\}^m$ as vertex set. Also, note that $\mathcal{C}_{\mathsf{A}}$ has at most one graph on n vertices for any n. Therefore the label decoder F_{A} is well-defined. It is easy to see that $(F_{\mathsf{A}}, 1)$ represents $\mathcal{C}_{\mathsf{A}}$, i.e. $\mathcal{C}_{\mathsf{A}} \subseteq \operatorname{gr}(F_{\mathsf{A}}, 1)$.

Up to this point the exact correspondence between $y \in \mathbb{N}$ and the label decoder F_y was not important. In fact, we only required the set of label decoders A to be countable. To show that the label decoder $F_{k\mathsf{EXP}}$ can be computed in $(k+1)\mathsf{EXP}$ it is important that given y the label decoder F_y from $k\mathsf{EXP}$ can be effectively computed. The following lemma grants this.

▶ **Lemma 10.** *For every* $k \geq 0$ *there exists a mapping* $f \colon \mathbb{N} \to \mathsf{ALL}$ *such that* $\operatorname{Im}(f) = k\mathsf{EXP}$ *and on input* $x \in \mathbb{N}$ *in binary and* $w \in \{0,1\}^*$ *the question* $w \in f(x)$ *can be decided in* $\exp^{k+1}(n^{\mathcal{O}(1)})$ *time with* $n = |w| + \log x$.

Proof. The lemma essentially states that all TMs running in $k\mathsf{EXP}$ can be simulated in $(k+1)\mathsf{EXP}$. Given the Gödelization of such a TM M and a word w as input the question whether M accepts x can be decided by a TM in $(k+1)\mathsf{EXP}$. Fix a reasonable encoding of TMs as natural numbers, i.e. given $z \in \mathbb{N}$ then M_z is a TM. Let $f(x) = (y,z) \Leftrightarrow x = 2^y 3^z$. It holds that $y \leq \log x$ for every $z \geq 0$. On input $x \in \mathbb{N}$ and $w \in \{0,1\}^*$ the reference input length is $n = |w| + \log x$. Compute $f(x) = (y,z)$ and then simulate M_z on w for $\exp^k(y|w|^y) \leq \exp^k(n^{n+1}) \in \mathcal{O}(\exp^{k+1}(n^2))$ steps. ◀

▶ **Lemma 11.** $F_{k\mathsf{EXP}} \in (k+1)\mathsf{EXP}$ *for every* $k \geq 1$.

Proof. On input xy with $x, y \in \{0,1\}^m$ and $m \geq 1$ compute $\tau(2^m) = (y, z)$. If it is undefined then reject. Otherwise there is a labeling scheme $S_{\tau(2^m)} = (F_y, z)$ and we need to compute the smallest graph G_0 on 2^m vertices such that $G_0 \notin \mathrm{gr}(S_{\tau(2^m)})$. If G_0 exists we assume that its vertex set is $\{0,1\}^m$ and accept iff $(x, y) \in E(G_0)$. If it does not exist then reject.

The graph G_0 can be computed as follows. Iterate over all labeled graphs H with 2^m vertices in order and over all bijections $\ell\colon V(H) \to \{0,1\}^{zm}$. Check if $H \in \mathrm{gr}(S_{\tau(2^m)})$ by checking for every pair of vertices $u, v \in V(H)$ if $(\ell(u), \ell(v)) \in F_y \Leftrightarrow (u, v) \in E(H)$. If this condition fails then $G_0 = H$. To query the label decoder F_y the previous lemma can be applied, i.e. y can be interpreted as encoding of a TM in kEXP that can be simulated.

Let us consider the time requirement w.r.t. m. To compute $\tau(2^m)$ we write down 2^m in unary and compute τ in polynomial time w.r.t. 2^m which is in the order $2^{\mathcal{O}(m)}$. To compute G_0 there are four nested loops. The first one goes over all labeled graphs on 2^m vertices which is bounded by $\exp^2(2m)$. The second loop considers all possible labelings ℓ of which there can be at most $\exp(zm)^{\exp(m)} = \exp(\exp(m)zm) \leq \exp^2(zm^2) \in \exp^2(m^{\mathcal{O}(1)})$; recall that z is polynomially bounded by m due to Definition 5. The other two loops go over all vertices of H meaning 2^m. By applying Lemma 10 the time required to compute $(\ell(u), \ell(v)) \in F_y$ is $\exp^{k+1}(n_0^{\mathcal{O}(1)})$ with $n_0 := 2zm + \log y$. Since $n_0 \in m^{\mathcal{O}(1)}$ this operation can be computed in $(k+1)$-exponential time. In summary, the runtime order of this algorithm is $\exp^{k+1}(m^{\mathcal{O}(1)})$. ◀

Now, Lemma 8 states that $\mathcal{C}_{k\mathsf{EXP}} \notin \mathsf{G}k\mathsf{EXP}$ and from Lemma 11 it follows that $\mathcal{C}_{k\mathsf{EXP}} \in \mathsf{G}(k+1)\mathsf{EXP}$ therefore proving Theorem 4. Notice, that this argument fails to show that $\mathsf{GP} \subsetneq \mathsf{GEXP}$ because the runtime to compute the label decoder F_{P} is at least double exponential due to the first two loops mentioned in the proof of Lemma 11. Can this argument be modified to separate these two classes as well? This seems rather unlikely. Nonetheless, we now know that there exist graph classes that have an implicit representation but a polynomial-time computable label decoder does not suffice to capture them.

▶ **Fact 12.** *If there exists a small, hereditary graph class $\mathcal{C}$ with $\mathcal{C}_{\mathsf{P}} \subseteq \mathcal{C}$ then the implicit graph conjecture is false.*

For two graph classes $\mathcal{C}$ and $\mathcal{D}$ let us call $\mathcal{D}$ the hereditary closure of $\mathcal{C}$ if $G \in \mathcal{D}$ iff G occurs as induced subgraph of some graph in $\mathcal{C}$. If the hereditary closure of $\mathcal{C}_{\mathsf{P}}$ is not a small graph class then it follows that the premise of Fact 12 is unsatisfiable. Recall that $\mathcal{C}_{\mathsf{P}}$ is not an unambiguous graph class but depends on the chosen order $\prec$ and pairing τ, which makes it difficult to analyze what kind of graphs are contained in such a class.

3 Parameter Characterization

We consider a graph parameter to be a total function $\lambda\colon \mathcal{G} \to \mathbb{N}$ and call it natural if the cardinality of its image is infinite. Examples of natural graph parameters are the chromatic number or the diameter. A graph class $\mathcal{C}$ is bounded by a graph parameter λ if there exists a $c \in \mathbb{N}$ such that for all $G \in \mathcal{C}$ it holds that $\lambda(G) \leq c$. We show that for every complexity class A such that GA is closed under union there exists a graph parameter that characterizes GA. One interesting aspect of such a characterization is that it might reveal algorithmic implications for graph classes that have a labeling scheme of certain complexity.

▶ **Definition 13.** Let $\mathbb{C}$ be a set of graph classes and λ is a graph parameter. We say λ characterizes $\mathbb{C}$ if for every graph class $\mathcal{C}$ it holds that $\mathcal{C} \in \mathbb{C}$ iff $\mathcal{C}$ is bounded by λ.

Let us say a set of graph classes $\mathbb{C}$ is complete if for every graph G there exists a $\mathcal{C} \in \mathbb{C}$ such that $G \in \mathcal{C}$.

▶ **Theorem 14.** *Let $\mathbb{C}$ be a complete set of graph classes closed under union and subsets with $\mathcal{G} \notin \mathbb{C}$. If there exists a countable subset of $\mathbb{C}$ such that its closure under subsets equals $\mathbb{C}$ then there exists a natural graph parameter that characterizes $\mathbb{C}$.*

Proof. Let $\mathbb{C}$ be a set of graph classes that satisfies the above premises and $\mathbb{C}' = \{\mathcal{C}_1, \mathcal{C}_2, \dots\}$ is the needed countable subset of $\mathbb{C}$. Let $\lambda(G)$ be the minimal $i \geq 1$ such that $G \in \mathcal{C}_i$. Since $\mathbb{C}$ is complete it follows that $\mathbb{C}'$ is complete and thus λ is total. Let us define $\mathcal{C}^{\lambda}_{\leq i}$ as $\{G \in \mathcal{G} \mid \lambda(G) \leq i\}$ and similarly $\mathcal{C}^{\lambda}_{=i}$. It follows that a class $\mathcal{C}$ is bounded by λ iff $\mathcal{C} \subseteq \mathcal{C}^{\lambda}_{\leq i}$ for some $i \in \mathbb{N}$. We now argue that λ characterizes $\mathbb{C}$.

If $\mathcal{C} \in \mathbb{C}$ then there exists an $i \in \mathbb{N}$ such that $\mathcal{C} \subseteq \mathcal{C}_i$. It follows that $\mathcal{C} \subseteq \mathcal{C}^{\lambda}_{\leq i}$. We show the other direction by induction: if $\mathcal{C} \subseteq \mathcal{C}^{\lambda}_{\leq i}$ then $\mathcal{C} \in \mathbb{C}$ for all $i \in \mathbb{N}$. For $i = 1$ it holds that $\mathcal{C} \subseteq \mathcal{C}^{\lambda}_{\leq 1} = \mathcal{C}^{\lambda}_{=1} = \mathcal{C}_1$. Since $\mathbb{C}$ is closed under subsets it follows that $\mathcal{C} \in \mathbb{C}$. For $i+1$ it holds that $\mathcal{C} \subseteq \mathcal{C}^{\lambda}_{\leq i+1}$ and $\mathcal{C}^{\lambda}_{\leq i+1} = \mathcal{C}^{\lambda}_{\leq i} \cup \mathcal{C}^{\lambda}_{=i+1}$. By induction hypothesis it follows that $\mathcal{C}^{\lambda}_{\leq i} \in \mathbb{C}$. Since $\mathbb{C}$ is closed under union it remains to argue that $\mathcal{C}^{\lambda}_{=i+1}$ is in $\mathbb{C}$. This follows by the observation $\mathcal{C}^{\lambda}_{=i+1} \subseteq \mathcal{C}_{i+1}$ and $\mathcal{C}_{i+1} \in \mathbb{C}$. ◀

Let us examine the premises of Theorem 14 with respect to the class of graph classes that we consider. Every class of the form G· is closed under subsets and for a lot of complexity classes A it also holds that GA is closed under union. For completeness a lookup table can be constructed for every singleton graph class. The required countable subset is given by the languages of A. In fact, every class of the form G· mentioned in this paper satisfies these premises and therefore has a parameter characterization with the only exception being the class GALL, which provably has no parameter characterization. Assume λ is a characterizing parameter for GALL and let $A = \{\mathcal{C}^{\lambda}_{\leq i} \mid i \in \mathbb{N}\}$. It must hold that for every graph class $\mathcal{C} \in$ GALL that it is a subset of some graph class in A. However, the diagonalization graph class $\mathcal{C}_A$ of A cannot be a subset of any graph class in A but has a labeling scheme and thus is in GALL, contradiction.

Consider the algorithmic relevance of such characterizations. Let $P \colon \mathcal{G} \to \{0,1\}$ be a graph property such as having a Hamiltonian cycle and λ is a graph parameter that characterizes the class GA. Assume that P can be decided in time $n^{f(k)}$ on input G with $k = \lambda(G)$ for some computable function $f \colon \mathbb{N} \to \mathbb{N}$. This can also be stated as P parameterized by λ being in the complexity class XP. Then it follows that the property P can be decided in polynomial time on every graph class in GA. The contra-position of this argument can be used to show that a graph class $\mathcal{C}$ is probably not in GA: if it is NP-hard to decide the property P on a graph class $\mathcal{C}$ then this implies that $\mathcal{C}$ cannot be in GA unless P = NP.

Of course, the characterizing parameter derived from the proof of Theorem 14 is not suitable for direct analysis but guarantees existence of such a characterization. However, there is room for different parameter characterizations of the same class as the following equivalence notion shows. For two graph parameters λ_1, λ_2 let us say that λ_2 bounds λ_1, in symbols $\lambda_1 \leq \lambda_2$, if every graph class $\mathcal{C}$ that is bounded by λ_1 is also bounded by λ_2. If $\lambda_1 \leq \lambda_2$ and $\lambda_2 \leq \lambda_1$ we say λ_1 and λ_2 are equivalent. For example, the maximum degree is bounded by clique number but not vice versa.

▶ **Fact 15.** *Let $\mathbb{C}_1, \mathbb{C}_2$ be two classes of graph classes and λ_1, λ_2 are respective characterizing graph parameters. $\mathbb{C}_1 \subseteq \mathbb{C}_2$ iff $\lambda_1 \leq \lambda_2$.*

It follows that two graph parameters are equivalent iff they characterize the same class of graph classes. For a complexity class A let λ_{A} be a characterizing graph parameter thereof. Hence, comparing the containment relation of two classes GA and GB is the same as examining whether λ_{A} bounds λ_{B} or vice versa. The interval number $\lambda_{\text{Intv}}(G)$ of a graph G is the smallest number $k \in \mathbb{N}$ such that G is a k-interval graph, see [2]. From this perspective some of our results can be stated as:

$$\lambda_{\text{Intv}} \lneq \lambda_{\mathsf{FO}_{\text{qf}}(<)} \leq \lambda_{\mathsf{L}} \leq \lambda_{\mathsf{P}} \leq \lambda_{\mathsf{EXP}} \lneq \lambda_{\mathsf{2EXP}} \lneq \cdots \lneq \lambda_{\mathsf{R}}$$

where $\lambda \lneq \lambda'$ means strict containment, i.e. $\lambda \leq \lambda'$ holds and $\lambda' \leq \lambda$ does not hold. The class $\mathsf{FO}_{\text{qf}}(<)$ is introduced in the next section.

4 First-Order Definable Label Decoders

For a given small, hereditary graph class there is no obvious way of showing that this class is not contained in GP or even GL as the fact that the IGC still stands open has shown. As a consequence, it is reasonable to look at a more restrictive model of computation for label decoders. From a complexity-theoretic view the circuit class AC^0 is probably among the first candidates. In this case uniformity issues have to be considered, i.e. the complexity of an algorithm computing the circuits for each input length. The strongest uniformity condition, which is the most suitable for lower bounds, leads to the class FO_{D} from descriptive complexity defined in terms of first-order logic [5]. However, the domain of discourse in this setting would be the positions of the labels, which is arguably not the most natural choice. Instead we propose the domain to be polynomially many natural numbers and a label consists of a constant number of elements of this domain. In this setting the labeling scheme for interval graphs can be stated as the formula $\varphi(x_1, x_2, y_1, y_2) = \neg(x_2 < y_1 \vee y_2 < x_1)$; compare this with the example given in the first section. It is also possible to describe k-interval graphs or any hereditary graph class with linearly many edges such as bounded arboricity graphs with such formulas.

For $n \geq 1$ let $\mathcal{N}_n$ be the structure that has $[n]$ as universe, the order relation $<$ on $[n]$ and addition as well as multiplication defined as functions:

$$+(x,y) = \begin{cases} x+y & \text{, if } x+y \leq n \\ 1 & \text{, if } x+y > n \end{cases} \quad , \quad \times(x,y) = \begin{cases} xy & \text{, if } xy \leq n \\ 1 & \text{, if } xy > n \end{cases}$$

For $\sigma \subseteq \{<, +, \times\}$ let $\mathsf{FO}_k(\sigma)$ be the set of first-order formulas with boolean connectives $\neg, \vee, \wedge$, quantifiers $\exists, \forall$ and k free variables using only equality and the relation and function symbols from σ. For $\sigma = \{<, +, \times\}$ we simply write FO_k. Let $\text{Vars}(\varphi)$ be the set of free variables in φ. Given $\varphi \in \mathsf{FO}_k(\sigma)$, $\text{Vars}(\varphi) = (x_1, \ldots, x_k)$ and an assignment $a_1, \ldots, a_k \in [n]$ we write $\mathcal{N}_n, (a_1, \ldots, a_k) \models \varphi$ if the interpretation $\mathcal{N}_n, (a_1, \ldots, a_k)$ satisfies φ under the usual semantics of first-order logic.

▶ **Definition 16.** A (quantifier-free) logical labeling scheme is a tuple $S = (\varphi, c)$ with a (quantifier-free) formula $\varphi \in \mathsf{FO}_{2k}$ and $c, k \in \mathbb{N}$. A (c,k)-labeling for a set V is a function $\ell \colon V \to [n^c]^k$ and induces the graph G_S^ℓ with vertex set V and edges (u,v) if $\mathcal{N}_{n^c}, (\ell(u), \ell(v)) \models \varphi$. Then a graph G is in $\text{gr}(S)$ if there exists a (c,k)-labeling ℓ for $V(G)$ such that $G = G_S^\ell$.

▶ **Definition 17.** Let $\sigma \subseteq \{<, +, \times\}$, $c, k \in \mathbb{N}$. A graph class $\mathcal{C}$ is in $\mathsf{G}_{c,k}\mathsf{FO}(\sigma)$ if there exists a logical labeling scheme (φ, c) with $\varphi \in \mathsf{FO}_{2k}(\sigma)$ such that $\mathcal{C} \subseteq \text{gr}(\varphi, c)$. And $\mathsf{GFO}(\sigma) = \cup_{c,k \in \mathbb{N}} \mathsf{G}_{c,k}\mathsf{FO}(\sigma)$. Let $\mathsf{GFO}_{\text{qf}}(\sigma)$ denote the quantifier-free analogue.

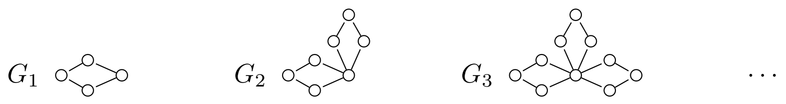


Figure 1 A family of graphs with unbounded interval number.

Notice, k numbers in $[n^c]$ can be encoded as string of length $ck \log n$. A logical labeling scheme can for instance express a system of polynomial inequalities on $2k$ variables and adjacency is determined by whether this system is satisfied when plugging in the values for two vertices. By disallowing multiplication these systems become linear. Quantified variables can be used to incorporate unknowns. For example, $\varphi(x, y) = \exists z : x \times z^2 = y$ means that there is an edge from u to v with labels x_u, y_v if y_v can be written as product of x_u and a square number.

▶ **Theorem 18.** $\mathsf{GFO} \subseteq \mathsf{GPH}$ *and* $\mathsf{GFO}_{\mathrm{qf}} \subseteq \mathsf{GL}$.

Proof sketch. It is known that the circuit class $\mathsf{TC}^0 \subseteq \mathsf{L}$ (assuming logspace-uniformity or stronger) and therefore $\mathsf{GTC}^0 \subseteq \mathsf{GL}$ [13]. We argue that $\mathsf{GFO}_{\mathrm{qf}} \subseteq \mathsf{GTC}^0$. Given a logical labeling scheme (φ, c) with $\varphi \in \mathsf{FO}_{2k}$ the label length in a graph with n vertices is $ck \log n$. The TC^0-circuit has $2ck \log n$ input bits and every block of $c \log n$ bits corresponds to the value of a free variable in φ. Every term in φ can be evaluated by implementing its syntax tree as part of the circuit since addition and multiplication can be computed in TC^0. The overflow condition, i.e. if the result is larger than n^c, has to be checked. Then for every atomic formula in φ it remains to test for equality or less than of the input terms. After replacing every atomic formula in φ by its truth value the formula becomes a propositional formula that can be seen as circuit since it is quantifier-free. If φ contains quantifiers assume that it is in prenex normal form, i.e. $\varphi = Q_1 z_1 \dots Q_q z_q \psi(x_1, \dots, x_{2k}, z_1, \dots, z_k)$ where $Q_i \in \{\exists, \forall\}$ and ψ is a quantifier-free formula. The values for $x_1, \dots, x_{2k}$ are determined by the input string and the value of a variable z_i corresponds to a binary word of length $k \log n$, which is linear in the size of the input string. Using the non-determinism of the polynomial-time hierarchy the values of the z_i's can be "guessed" and then evaluated using the TC^0-circuit described before, which can be simulated in polynomial time. ◀

Indeed, all of the graph classes mentioned in the beginning of this section are already contained in $\mathsf{GFO}_{\mathrm{qf}}(<)$. Therefore let us consider this class more closely.

▶ **Fact 19.** *The interval number λ_{Intv} is strictly bounded by a graph parameter that characterizes* $\mathsf{GFO}_{\mathrm{qf}}(<)$.

Proof. This statement is equivalent to saying that k-interval graphs are contained in $\mathsf{GFO}_{\mathrm{qf}}(<)$ and there exists a graph class $\mathcal{C} \in \mathsf{GFO}_{\mathrm{qf}}(<)$ that is no subclass of k-interval graphs for all $k \geq 1$. The containment of k-interval graphs in $\mathsf{GFO}_{\mathrm{qf}}(<)$ for every k follows by translating its geometrical representation into a logical labeling scheme as we have done for interval graphs previously. Consider the family of graphs shown in Figure 1 where G_{i+1} is obtained by appending a new 4-cycle to G_i.

Then the class $\{G_i \mid i \in \mathbb{N}\}$ lies in $\mathsf{GFO}_{\mathrm{qf}}(<)$ but can be verified to have unbounded interval number. This follows from the observation that the vertex with maximal degree in G_i cannot be represented with $i - 1$ intervals. ◀

A natural question is how do c and k affect the expressiveness of $\mathsf{G}_{c,k}\mathsf{FO}_{\mathrm{qf}}(<)$. Non-surprisingly, increasing k strictly enhances the graph classes that can be represented as we will see in a moment. The parameter c determines how large a number stored in a label can be, i.e. at most n^c. In fact, c is degenerate in the sense that it can be bounded in terms of k. It would be surprising if the same holds in the presence of addition.

▶ **Lemma 20.** $\mathsf{G}_{c,k}\mathsf{FO}_{\mathrm{qf}}(<) \subseteq \mathsf{G}_{k,k}\mathsf{FO}_{\mathrm{qf}}(<)$ *for all* $c, k \geq 1$.

Proof. Consider why it suffices for an interval graph on n vertices to use only numbers between 1 and $2n$ to represent the intervals. For the same reason it makes no difference for a quantifier-free formula $\varphi \in \mathsf{FO}_{2k}(<)$ to be evaluated on a universe larger than kn in the sense that a labeling $\ell\colon V(G) \to \mathbb{N}^k$ can be converted to a labeling $\ell'\colon V(G) \to [kn]^k$ such that adjacency is preserved. More precisely, a (c,k)-labeling ℓ for a vertex set V can be transformed into a (k,k)-labeling ℓ' such that $G^{\ell}_{(\varphi,c)} = G^{\ell'}_{(\varphi,k)}$ holds for every quantifier-free formula $\varphi \in \mathsf{FO}_{2k}(<)$. Let $n = |V|$ be the number of vertices. Since k numbers are assigned to each vertex there are at most kn numbers in $A = \{x_i \mid u \in V, \ell(u) = (x_1,\dots,x_k), i \in [k]\}$. For an $a \in A$ let $\mathrm{ord}(a) = |\{b \in A \mid b < a\}| + 1$, i.e. the number of numbers in A that are smaller than a plus one. For $u \in V(G)$ we define $\ell'(u)$ as follows. Let $\ell(u) = (x_1,\dots,x_k)$. Then $\ell'(u) = (\mathrm{ord}(x_1),\dots,\mathrm{ord}(x_l))$. Notice that the maximal value for a component of $\ell'(u)$ is kn. It remains to check that the truth value of φ is invariant under this modified labeling, which follows from the fact that $x < y \Leftrightarrow \mathrm{ord}(x) < \mathrm{ord}(y)$. ◀

A consequence of this is that a logical labeling scheme in $\mathsf{GFO}_{\mathrm{qf}}(<)$ is solely determined by its formula φ. Therefore we consider a quantifier-free formula $\varphi \in \mathsf{FO}_{2k}(<)$ to be the logical labeling scheme (φ, k) as well. To check whether a graph G is in $\mathrm{gr}(\varphi)$ it suffices to find a labeling $\ell\colon V(G) \to \mathbb{N}^k$ with $2k = |\mathrm{Vars}(\varphi)|$ which can be regarded as (c,k)-labeling for a sufficiently large c. Stated differently, one does not need to worry about the numbers being polynomially bounded.

Also, it implies that for every k there exists a $k' > k$ such that $\mathsf{G}_{k,k}\mathsf{FO}_{\mathrm{qf}}(<) \subsetneq \mathsf{G}_{k',k'}\mathsf{FO}_{\mathrm{qf}}(<)$. Assume the opposite, then $\mathsf{GFO}_{\mathrm{qf}}(<)$ collapses to $\mathsf{G}_{k,k}\mathsf{FO}_{\mathrm{qf}}(<)$. It follows that every graph class in $\mathsf{GFO}_{\mathrm{qf}}(<)$ can be represented using $k^2 \log n$ bits and therefore has at most $\exp(k^2 \log n)$ graphs on n vertices, which obviously cannot be the case for any $k \in \mathbb{N}$.

▶ **Lemma 21.** *The graph class that is the union of every graph class in* $\mathsf{G}_{k,k}\mathsf{FO}_{\mathrm{qf}}(<)$ *is contained in* $\mathsf{GFO}_{\mathrm{qf}}(<)$ *for all* $k \in \mathbb{N}$.

Proof. We argue that $\mathsf{GFO}_{\mathrm{qf}}(<)$ is closed under finite union and that there exists only a finite number of labeling schemes in $\mathsf{G}_{k,k}\mathsf{FO}_{\mathrm{qf}}(<)$ such that they represent different graph classes. For closure under union consider two labeling schemes given by their quantifier-free formulas $\varphi, \psi \in \mathsf{FO}_{2k}(<)$. Then the graph class given by the following formula with $2k+2$ variables contains the union of $\mathrm{gr}(\varphi)$ and $\mathrm{gr}(\psi)$:

$$\big(x_{k+1} = x_{2k+2} \Rightarrow \varphi(x_1,\dots,x_k,x_{k+2},\dots,x_{2k+1})\big) \wedge$$
$$\big(x_{k+1} \neq x_{2k+2} \Rightarrow \psi(x_1,\dots,x_k,x_{k+2},\dots,x_{2k+1})\big)$$

The second claim follows from the fact that there are only finitely many semantically different quantifier-free formulas in $\mathsf{FO}_k(<)$ for every k. More precisely, there are at most $2k^2$ different atomic formulas ('$<$' and '$=$') on k variables and therefore at most $\exp^2(2k^2)$ semantically different formulas, which is the number of boolean functions on $2k^2$ variables. ◀

▶ **Definition 22.** For a graph G and $k \in \mathbb{N}$ we define the graph parameter $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}$ such that $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}(G) = k$ if k is the minimal number with $\{G\} \in \mathsf{G}_{k,k}\mathsf{FO}_{\mathrm{qf}}(<)$.

▶ **Fact 23.** *The graph parameter* $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}(G)$ *characterizes* $\mathsf{GFO}_{\mathrm{qf}}(<)$.

Proof. One direction is trivial: if $\mathcal{C}$ is in $\mathsf{GFO}_{\mathrm{qf}}(<)$ then it is bounded by $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}$. For the other direction let $\mathcal{C}$ be bounded by $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}$ meaning that there exists a k such that for every $G \in \mathcal{C}$ it holds that $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}(G) \leq k$. Therefore $\mathcal{C}$ is a subset of the union of all graph classes in $\mathsf{G}_{k,k}\mathsf{FO}_{\mathrm{qf}}(<)$ which is in $\mathsf{GFO}_{\mathrm{qf}}(<)$ by Lemma 21. ◀

We remark that a similar construction using the label length does not yield a characterizing parameter for GP or GL. More specifically, the parameter defined by $\lambda(G)$ = minimal k such that $\{G\} \in \mathsf{G}_k\mathsf{P}$ does not characterize GP simply because the union of all graph classes in $\mathsf{G}_1\mathsf{P}$ already contains all graphs(the analogon of Lemma 21 fails).

4.1 Directed Acyclic Graph Characterization

The semantics of a logical labeling scheme given by a quantifier-free formula $\varphi \in \mathsf{FO}_{2k}(<)$ can be alternatively characterized by directed acyclic graphs (DAGs). Intuitively, an edge in the DAG corresponds to an atomic formula using '$<$'. The atomic formulas involving equality can be modeled by grouping variables together. This means the DAG has not the variables of φ as vertex set but rather a partition of these variables.

▶ **Definition 24.** Let $k \in \mathbb{N}$. We call a DAG $D = (X, E_D)$ a k-DAG if its vertex set X partitions $[2k]$. A k-labeling of a vertex set V is a function $\ell \colon V \to \mathbb{N}^k$. A k-DAG D and a k-labeling ℓ of a vertex set V define the graph G^ℓ_D on vertex set V with the following edges. For $u, v \in V$ let $(\ell(u), \ell(v)) = (x_1, \ldots, x_{2k})$. There is an edge (u, v) in G^ℓ_D if the following two conditions are satisfied:

1. For all $i, j \in [2k]$ it holds that $x_i = x_j$ whenever i, j are in the same part of X,
2. For all edges $(A, B) \in E_D$ it holds that $x_i < x_j$ for all $i \in A$ and $j \in B$.

▶ **Definition 25.** A graph $G = (V, E)$ is k-expressible for a $k \in \mathbb{N}$ if there exists a finite sequence of k-DAGs $D_1, \ldots, D_r$ and a k-labeling ℓ of V such that G is the edge-union of $G^\ell_{D_1}, \ldots, G^\ell_{D_r}$.

▶ **Theorem 26.** *For a graph G and $k \in \mathbb{N}$ it holds that $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}(G) = k$ iff k is the minimal number such that G is k-expressible.*

Proof. We show that there is a one-to-one correspondence between the semantics of a quantifier-free formula $\varphi \in \mathsf{FO}_{2k}(<)$ and k-DAGs. We can assume that φ contains no negation. To see that this can be done without loss of generality let φ be in negation normal form. Then $\neg x = y$ can be replaced by $x < y \vee y < x$ and $\neg x < y$ by $y < x \vee x = y$. Next, we assume that φ is in disjunctive normal form, i.e. $\varphi = C_1 \vee \cdots \vee C_p$ where C_i consists of atomic formulas linked by conjunction. Given a (c, k)-labeling ℓ for a vertex set V the formula φ induces the graph G^ℓ_S with $S = (\varphi, k)$ as described in Definition 16. Due to the observation given after the proof of Lemma 20 it is okay to consider a less restrictive k-labeling $\ell \colon V \to \mathbb{N}^k$ instead and additionally we write G^ℓ_φ instead of G^ℓ_S. Since every clause C_i is a formula as well it can be seen as logical labeling scheme, which induces the graph $G^\ell_{C_i}$. Then the correspondence between the graphs induced by φ and its clauses $C_1, \ldots, C_p$ is that G^ℓ_φ is the edge-union of $G^\ell_{C_1}, \ldots, G^\ell_{C_p}$. If a clause is unsatisfiable then its induced graph is the empty graph and thus removing this clause does not affect G^ℓ_S. Therefore we assume that every clause is satisfiable.

We now argue how to convert a clause C from φ into a k-DAG $D = (X, E_D)$ such that $G^\ell_C = G^\ell_D$ for every k-labeling ℓ. Consider the undirected graph H which has the variables of φ as vertices and two vertices x_i, x_j are adjacent iff the clause C contains $x_i = x_j$ or $x_j = x_i$. It follows that the connected components of H partition the variables of φ; let X be this partition. Now, consider the directed graph F which has the variables of φ as vertices again and there is an edge (x_i, x_j) in F iff C contains the atomic formula $x_i < x_j$. Since we can assume C to be satisfiable it follows that for every part A in the partition X (A is a subset of the variables of φ) the induced subgraph of F on the vertex set A yields the independent graph. Assume the opposite, then there exist two variables x_i, x_j in the same part of X such that (x_i, x_j) is an edge in F. This means that C contains the atomic formulas $x_i = x_j$ and $x_i < x_j$, which contradicts satisfiability of C. Let us define the operation of merging a set of vertices S in a graph G such that the resulting graph G' is the same as G except that all vertices in S are replaced by a single vertex v_s and there is an edge (u, v_s) in G' if there is a vertex $v \in S$ such that (u, v) is an edge in the old graph G; analogously for edges (v_s, u). Now, let F' be the graph obtained from F by merging each part of X. Then there is a natural one-to-one correspondence between the partition X and the vertex set of F'. We define D to have the same edges as F' via this correspondence. It remains to check that for this construction $G^\ell_C = G^\ell_D$ holds indeed. To prove the other direction a k-DAG can be converted into a conjunctive clause in a similar way. ◀

We conclude with the following two observations. By adding edge weights $w\colon E \to \mathbb{N}$ to the k-DAGs and adjusting the second condition of Definition 24 such that for all edges $(A, B) \in E_D$ it holds that $x_j - x_i \geq w(x_i, x_j)$ for all $x_i \in A, x_j \in B$ the semantics of existential quantifiers can be mimicked. Besides, given two k-DAGs D_1 and D_2 with identical vertex sets $V(D_1) = V(D_2)$ it holds that $G^\ell_{D_1} = G^\ell_{D_2}$ for every k-labeling ℓ whenever the transitive closures of D_1 and D_2 coincide.

5 Conclusions and Future Research

We have seen that limiting the computational resources for label decoders does indeed affect the class of graph classes that can be represented. Unfortunately, for a specific graph class the diagonalization argument from the second section does not help us determine whether it lies in GP. However, as of now it is not even clear whether any candidate of the IGC admits a labeling scheme at all as mentioned at the end of the first section. Therefore trying to place any of these classes in GP seems elusive. On the other side, proving lower bounds against GP or GL for small, hereditary graph classes might be just as futile given the lack of a suitable reduction notion. To counter this grim situation we have introduced a logical framework in the previous section that is much more restrictive than the TM model in its quantifier-free variant but still expressive enough to capture many of the implicit representations that we know. It appears to be a realistic goal to prove impossibility results in this setting, or more concretely refute the following weaker version of the IGC:

▶ **Conjecture 27** (Weak IGC). *Every small, hereditary graph class is in* $\mathsf{GFO}_{\mathrm{qf}}$.

As a first step in this direction we have investigated the fragment $\mathsf{GFO}_{\mathrm{qf}}(<)$ and made some structural observations. With the concept of parameter characterizations we have shown that the question of whether a certain graph class lies in $\mathsf{GFO}_{\mathrm{qf}}(<)$ can be answered by considering the k-expressibility property of every graph in this class independently. The directed acyclic graph characterization gives an alternative view on $\mathsf{GFO}_{\mathrm{qf}}(<)$, which is independent of the logical formalism. This could be a useful tool for proving lower bounds against this class. But even this small fragment seems to be surprisingly expressive as the following task shows. Give an example of a family of graphs that is not bounded by $\lambda_{\mathsf{FO}_{\mathrm{qf}}(<)}$. Recall that for the interval number this was quite simple, see Figure 1. Another interesting question is whether adding quantifiers enhances the expressiveness, i.e. $\mathsf{GFO}_{\mathrm{qf}}(<) = \mathsf{GFO}(<)$?

References

1 Alstrup, S., Dahlgaard, S., Knudsen, M.: Optimal Induced Universal Graphs and Adjacency Labeling for Trees. Foundations of Computer Science (2015)

2 Fellows, M., Hermelin, D., Rosamond, F., Vialette, S.: On the parameterized complexity of multiple-interval graph problems. Theoretical Computer Science, Volume 410 (2009)

3 Fiduccia, C., Scheinerman, E., Trenk, A., Zito, J.: Dot product representations of graphs. Discrete Mathematics 181 (1998)

4 Gavoille, C., Peleg, D., Pérennes, S., Raz, R.: Distance labeling in graphs. Journal of Algorithms 53 (2004)

5 Immerman, N.: Descriptive complexity. Springer-Verlag New York, Inc. (1999)

6 Kang, R., Müller, T.: Sphere and Dot Product Representations of Graphs. Discrete & Computational Geometry (2012)

7 Kannan, S., Naor, M., Rudich, S.: Implicit Representations of Graphs. SIAM Journal Disc. Math. (1992)

8 Korman, A., Kutten, S.: A note on models for graph representations. Theoretical Computer Science 410 (2009)

9 McDiarmid, C., Müller, T.: Integer realizations of disk and segment graphs. Journal of Combinatorial Theory, Series B 103 (2013)

10 Scheinerman, E.: Local representations using very short labels. Journal of Discrete Mathematics 203 (1999)

11 Scheinerman, E., Zito, J.: On the Size of Hereditary Classes of Graphs. Journal of Combinatorial Theory, Series B 61 (1994)

12 Spinrad, J.: Efficient Graph Representations. Fields Institute Monographs, American Mathematical Soc. (2003)

13 Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer-Verlag New York, Inc. (1999)

Nested Weighted Limit-Average Automata of Bounded Width[*]

Krishnendu Chatterjee[1], Thomas A. Henzinger[2], and Jan Otop[3]

1 **IST Austria**
`krish.chat@ist.ac.at`
2 **IST Austria**
`tah@ist.ac.at`
3 **University of Wrocław**
`jotop@cs.uni.wroc.pl`

Abstract

While weighted automata provide a natural framework to express quantitative properties, many basic properties like average response time cannot be expressed with weighted automata. Nested weighted automata extend weighted automata and consist of a master automaton and a set of slave automata that are invoked by the master automaton. Nested weighted automata are strictly more expressive than weighted automata (e.g., average response time can be expressed with nested weighted automata), but the basic decision questions have higher complexity (e.g., for deterministic automata, the emptiness question for nested weighted automata is PSpace-hard, whereas the corresponding complexity for weighted automata is PTime). We consider a natural subclass of nested weighted automata where at any point at most a bounded number k of slave automata can be active. We focus on automata whose master value function is the limit average. We show that these nested weighted automata with bounded width are strictly more expressive than weighted automata (e.g., average response time with no overlapping requests can be expressed with bound $k = 1$, but not with non-nested weighted automata). We show that the complexity of the basic decision problems (i.e., emptiness and universality) for the subclass with k constant matches the complexity for weighted automata. Moreover, when k is part of the input given in unary we establish PSpace-completeness.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases weighted automata; nested weighted automata; complexity; mean-payoff

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.24

1 Introduction

Traditional to quantitative verification. In contrast to the traditional view of formal verification that focuses on Boolean properties of systems, such as "every request is eventually granted", quantitative specifications consider properties like "the long-run average success rate of an operation is at least one half" or "the long-run average response time is below a threshold." Such properties are crucial for performance related properties, for resource-constrained systems, such as embedded systems, and significant attention has been devoted to them [21, 14, 13, 22, 2].

[*] This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award), ERC Start grant (279307: Graph Games), Vienna Science and Technology Fund (WWTF) through project ICT15-003 and by the National Science Centre (NCN), Poland under grant 2014/15/D/ST6/04543.

Weighted automata. A classical model to express quantitative properties is *weighted automata* that extends finite automata where every transition is assigned a rational number called a *weight*. Each run results in a sequence of weights, and a *value function* aggregates the sequence into a single value. For non-deterministic weighted automata, the value of a word is the infimum value of all runs over the word. Weighted automata provide a natural and flexible framework to express quantitative[1] properties [14]. Weighted automata have been studied over finite words with weights from a semiring [21], and extended to infinite words with limit averaging or supremum as a value function [14, 13, 12]. While weighted automata over semirings can express several quantitative properties [27], they cannot express long-run average properties that weighted automata with limit averaging can [14]. However, even weighted automata with limit averaging cannot express the basic quantitative property of average response time [16, Example 5].

Nested weighted automata. To express properties like average response time, weighted automata were extended to *nested weighted automata (NWA)* [16]. An NWA consists of a master automaton and a set of slave automata. The master automaton runs over infinite input words. At every transition the master automaton can invoke a slave automaton that runs over a finite subword of the infinite word, starting at the position where the slave automaton is invoked. Each slave automaton terminates after a finite number of steps and returns a value to the master automaton. Each slave automaton is equipped with a value function for finite words, and the master automaton aggregates the returned values from slave automata using a value function for infinite words. For Boolean finite automata, nested automata are as expressive as the non-nested counterpart, whereas NWA are strictly more expressive than non-nested weighted automata [16]. It has been shown in [16] that NWA provide a specification framework where many basic quantitative properties, which cannot be expressed by weighted automata, can be expressed easily, and they provide a natural framework to study quantitative run-time verification.

The basic decision questions. We consider the basic automata-theoretic decision questions of emptiness and universality. The importance of these basic questions in the weighted automata setting is as follows: (1) Consider a system modeled by a finite-automaton recognizing traces of the system and a quantitative property given as a weighted automaton or an NWA. Then whether the worst-case (resp., best-case) behavior has the value at least λ is the emptiness (resp., universality) question on the product. (2) Problems related to model measuring (that generalizes model checking) and model repair also reduce to the emptiness problem [25, 16].

Complexity gap. In this work we focus on the following classical value functions: $\textsc{LimAvg}$ for infinite words, which is the long-run average property; and $\textsc{Sum}, \textsc{Sum}^+$ (where $\textsc{Sum}^+$ is the sum of absolute values) for finite words. While NWA are strictly more expressive than weighted automata, the complexity of the decision questions are either unknown or considerably higher. Table 1 (non-bold-faced results) summarizes the existing results for weighted automata [14] and NWA [16], for example, for NWA for $\textsc{Sum}^+$ the known bounds are $\textsc{ExpSpace}$ and $\textsc{PSpace}$-hard, and for $\textsc{Sum}$ even the decidability of the basic decision

[1] We use the term "quantitative" in a non-probabilistic sense, which assigns a quantitative value to each infinite run of a system, representing long-run average or maximal response time, or power consumption, or the like, rather than taking a probabilistic average over different runs.

Table 1 Decidability and complexity of emptiness and universality for weighted and nested weighted automata with LimAvg value function and Sum and Sum$^+$ value function for slave automata. Our results are bold faced. Moreover all PTime results become NLogSpace-complete when the weights are specified in unary.

	Deterministic (Emptiness/Universality)	Nondeterministic Emptiness	Nondeterministic Universality
Weighted aut.	PTime		Undecidable
NWA (LimAvg, Sum$^+$)	ExpSpace, PSpace-hard **PTime (width k is constant)** **PSpace-c. (bounded width)**		Undecidable
NWA (LimAvg, Sum)	Open **PTime (width k is constant)** **PSpace-c. (bounded width)**		Undecidable

questions is open (or undecidable). Thus, a fundamental question is whether there exist sub-classes of NWA that are strictly more expressive than weighted automata and yet have better complexity than general NWA. We address this question in this paper.

Nested weighted automata with bounded width. For NWA, let the maximum number of slave automata that can be active at any point be the *width* of the automaton. In this work we consider a natural special class of NWA, namely, NWA with bounded width, i.e., NWA where at any point at most k slave automata can be active. For example, the average response time with bounded number of requests pending at any point can be expressed by an NWA with bounded width, but not with a weighted automaton. Moreover, the class of NWA with bounded width is equivalent to automata with monitor counters [18], which are automata equipped with counters, where at each transition, a counter can be started, terminated, or the value of the counter can be increased or decreased. The transitions do not depend on the counter values, and hence they are referred to as monitor counters. The values of the counters when they are terminated give rise to the sequence of weights, which is aggregated into a single value with the LimAvg value function (see [18]). Automata with monitor counters are similar in spirit with the class of cost register automata of [2].

Our contributions. Our contributions are as follows (summarized as bold-faced results in Table 1):

1. *Constant width.* We show that the emptiness problem (resp., the emptiness and the universality problems) for non-deterministic (resp., deterministic) NWA with constant width (i.e., k is constant) can be solved in polynomial time and is NLogSpace-complete when the weights are specified in unary. Thus we achieve the same complexity as weighted automata for a much more expressive class of quantitative properties.
2. *Bounded width.* We show that the emptiness problem (resp., the emptiness and the universality problems) for non-deterministic (resp., deterministic) NWA with bounded width (i.e., k is a part of input given in unary) is PSpace-complete. Thus we establish precise complexity when k is a part of input given in unary.
3. *Deciding width.* We show that checking whether a given NWA has width k can be solved in polynomial time for constant k and in PSpace if k is given in the input (Theorem 6).

Technical contributions. Our main technical contributions for deterministic (LimAvg; Sum)-automata are as follows.

1. *Infinite infimum.* We first identify a necessary and sufficient condition for the infimum value over all words to be $-\infty$, and show that this condition can be checked efficiently.
2. *Lasso-approximation.* We show that if the above condition does not hold, then the infimum over all words can be approximated by lasso words, i.e., words of the form vu^ω. Moreover, we show that the infimum value is achieved with words where the slave automata run for short length relative to the point of the invocation, and hence the partial averages converge.
3. *Reduction to width* 1. Using the lasso-approximation we reduce the emptiness problem of width bounded by k to the corresponding problem of width 1. We show that the case of width 1 can be solved using standard techniques.

In the paper we present the key intuitions of the proofs, and due to space restrictions the technical details are in the full version [17].

Related works. Weighted automata over finite words have been extensively studied, the book [21] provides an excellent collection of results. Weighted automata on infinite words have been studied in [14, 13, 22]. The extension to weighted automata with monitor counters over finite words has been considered as cost register automata in [2]. A version of nested weighted automata over finite words has been studied in [6], and nested weighted automata over infinite words has been studied in [16]. Several quantitative logics have also been studied, such as [5, 7, 1]. In this work we consider a subclass of nested weighted automata which is strictly more expressive than weighted automata yet achieve the same complexity for the basic decision questions. Probabilistic models (such as Markov decision processes) with quantitative properties (such as limit-average or discounted-sum) have also been extensively studied for single objectives [23, 28], and for multiple objectives and their combinations [20, 10, 15, 8, 19, 9, 24, 11, 3, 4]. While NWA with bounded width have been studied under probabilistic semantics [18], the basic automata theoretic decision problems have not been studied for them.

2 Preliminaries

2.1 Words and automata

Words. We consider a finite *alphabet* of letters Σ. A *word* over Σ is a (finite or infinite) sequence of letters from Σ. We denote the i-th letter of a word w by $w[i]$, and for $i < j$ we have $w[i,j]$ is the word $w[i]w[i+1]\ldots w[j]$. The length of a finite word w is denoted by $|w|$; and the length of an infinite word w is $|w| = \infty$. For an infinite word w, thus $w[i,\infty]$ is the suffix of the word with first $i-1$ letters removed.

Labeled automata. For a set X, an *X-labeled automaton* $\mathcal{A}$ is a tuple $\langle \Sigma, Q, Q_0, \delta, F, C \rangle$, where (1) Σ is the alphabet, (2) Q is a finite set of states, (3) $Q_0 \subseteq Q$ is the set of initial states, (4) $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, (5) F is a set of accepting states, and (6) $C : \delta \mapsto X$ is a labeling function. A labeled automaton $\langle \Sigma, Q, \{q_0\}, \delta, F, C \rangle$ is *deterministic* if and only if δ is a function from $Q \times \Sigma$ into Q and Q_0 is a singleton. For deterministic labeled automata, we omit curly brackets for Q_0 and write $\langle \Sigma, Q, q_0, \delta, F, C \rangle$.

Semantics of (labeled) automata. A *run* π of a (labeled) automaton $\mathcal{A}$ on a word w is a sequence of states of $\mathcal{A}$ of length $|w|+1$ such that $\pi[0]$ belongs to the initial states of $\mathcal{A}$ and for every $0 \leq i \leq |w|-1$ we have $(\pi[i], w[i+1], \pi[i+1])$ is a transition of $\mathcal{A}$. A run π on a finite word w is *accepting* iff the last state $\pi[|w|]$ of the run is an accepting state of

$\mathcal{A}$. A run π on an infinite word w is *accepting* iff some accepting state of $\mathcal{A}$ occurs infinitely often in π. For an automaton $\mathcal{A}$ and a word w, we define $\mathsf{Acc}(w)$ as the set of accepting runs on w. Note that for deterministic automata, every word w has at most one accepting run ($|\mathsf{Acc}(w)| \leq 1$).

Weighted automata. A *weighted automaton* is a $\mathbb{Z}$-labeled automaton, where $\mathbb{Z}$ is the set of integers. The labels are called *weights*.

Semantics of weighted automata. We define the semantics of weighted automata in two steps. First, we define the value of a run. Second, we define the value of a word based on the values of its runs. To define values of runs, we will consider *value functions* f that assign real numbers to sequences of integers. Given a non-empty word w, every run π of $\mathcal{A}$ on w defines a sequence of weights of successive transitions of $\mathcal{A}$, i.e., $C(\pi) = (C(\pi[i-1], w[i], \pi[i]))_{1 \leq i \leq |w|}$; and the value $f(\pi)$ of the run π is defined as $f(C(\pi))$. We denote by $(C(\pi))[i]$ the weight of the i-th transition, i.e., $C(\pi[i-1], w[i], \pi[i])$. The value of a non-empty word w assigned by the automaton $\mathcal{A}$, denoted by $\mathcal{L}_{\mathcal{A}}(w)$, is the infimum of the set of values of all *accepting* runs; i.e., $\inf_{\pi \in \mathsf{Acc}(w)} f(\pi)$, and we have the usual semantics that infimum of an empty set is infinite, i.e., the value of a word that has no accepting run is infinite. Every run π on the empty word has length 1 and the sequence $C(\pi)$ is empty, hence we define the value $f(\pi)$ as an external (not a real number) value $\bot$. Thus, the value of the empty word is either $\bot$, if the empty word is accepted by $\mathcal{A}$, or ∞ otherwise. To indicate a particular value function f that defines the semantics, we will call a weighted automaton $\mathcal{A}$ an f-automaton.

Value functions. For finite runs we consider the following classical value functions: for runs of length $n+1$ we have

- *Sum, absolute sum:* the sum function $\textsc{Sum}(\pi) = \sum_{i=1}^{n} (C(\pi))[i]$, the absolute sum $\textsc{Sum}^{+}(\pi) = \sum_{i=1}^{n} \mathsf{Abs}((C(\pi))[i])$, where $\mathsf{Abs}(x)$ is the absolute value of x,

For infinite runs we consider:

- *Limit average:* $\textsc{LimAvg}(\pi) = \liminf_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k} (C(\pi))[i]$.

Silent moves. Consider a $(\mathbb{Z} \cup \{\bot\})$-labeled automaton. We can consider such an automaton as an extension of a weighted automaton in which transitions labeled by $\bot$ are *silent*, i.e., they do not contribute to the value of a run. Formally, for every function $f \in \mathsf{InfVal}$ we define $\mathsf{sil}(f)$ as the value function that applies f on sequences after removing $\bot$ symbols. The significance of silent moves is as follows: it allows to ignore transitions, and thus provide robustness where properties could be specified based on desired events rather than steps.

2.2 Nested weighted automata

In this section we describe nested weighted automata introduced in [16], and closely follow the description of [16]. For more details and illustration of such automata we refer the reader to [16]. We start with an informal description.

Informal description. A *nested weighted automaton* (NWA) consists of a labeled automaton over infinite words, called the *master automaton*, a value function f for infinite words, and a set of weighted automata over finite words, called *slave automata*. A nested weighted automaton can be viewed as follows: given a word, we consider the run of the master automaton on the word, but the weight of each transition is determined by dynamically running slave automata; and then the value of a run is obtained using the value function f. That is, the master automaton proceeds on an input word as an usual automaton, except that before it takes a transition, it starts a slave automaton corresponding to the label of the current transition. The slave automaton starts at the current position of the word of the master automaton and works on some finite part of the input word. Once a slave automaton finishes, it returns its value to the master automaton, which treats the returned value as the weight of the current transition that is being executed. The slave automaton might immediately accept and return value $\bot$, which corresponds to *silent* transitions. If one of slave automata rejects, the nested weighted automaton rejects. We first present an example and then the formal definition.

▶ **Example 1** (Average response time). Consider an alphabet Σ consisting of requests r, grants g, and null instructions #. The average response time (ART) property asks for the average number of instructions between any request and the following grant. This property cannot be expressed by a non-nested automaton: a quantitative property is a function from words to reals, and as a function the range of non-nested LimAvg-automata is bounded, whereas the ART can have unbounded values (for details see [16]).

Nested weighted automata. A *nested weighted automaton* (NWA) is a tuple $\langle \mathcal{A}_{mas}; f; \mathfrak{B}_1, \ldots, \mathfrak{B}_l \rangle$, where (1) $\mathcal{A}_{mas}$, called the *master automaton*, is a $\{1, \ldots, l\}$-labeled automaton over infinite words (the labels are the indexes of automata $\mathfrak{B}_1, \ldots, \mathfrak{B}_l$), (2) f is a value function on infinite words, called the *master value function*, and (3) $\mathfrak{B}_1, \ldots, \mathfrak{B}_l$ are weighted automata over finite words called *slave automata*. Intuitively, an NWA can be regarded as an f-automaton whose weights are dynamically computed at every step by the corresponding slave automaton. We define an $(f; g)$*-automaton* as an NWA where the master value function is f and all slave automata are g-automata.

Semantics: runs and values. A *run* of $\mathbb{A}$ on an infinite word w is an infinite sequence $(\Pi, \pi_1, \pi_2, \ldots)$ such that (1) Π is a run of $\mathcal{A}_{mas}$ on w; (2) for every $i > 0$ we have π_i is a run of the automaton $\mathfrak{B}_{C(\Pi[i-1], w[i], \Pi[i])}$, referenced by the label $C(\Pi[i-1], w[i], \Pi[i])$ of the master automaton, on some finite word of $w[i, j]$. The run $(\Pi, \pi_1, \pi_2, \ldots)$ is *accepting* if all runs $\Pi, \pi_1, \pi_2, \ldots$ are accepting (i.e., Π satisfies its acceptance condition and each $\pi_1, \pi_2, \ldots$ ends in an accepting state) and infinitely many runs of slave automata have length greater than 1 (the master automaton takes infinitely many non-silent transitions). The value of the run $(\Pi, \pi_1, \pi_2, \ldots)$ is defined as $\mathsf{sil}(f)(v(\pi_1)v(\pi_2)\ldots)$, where $v(\pi_i)$ is the value of the run π_i in the corresponding slave automaton. The value of a word w assigned by the automaton $\mathbb{A}$, denoted by $\mathcal{L}_{\mathbb{A}}(w)$, is the infimum of the set of values of all *accepting* runs. We require accepting runs to contain infinitely many non-silent transitions as f is a value function over infinite sequences, hence the sequence $v(\pi_1)v(\pi_2)\ldots$ with $\bot$ removed must be infinite.

Deterministic nested weighted automata. An NWA $\mathbb{A}$ is *deterministic* if (1) the master automaton and all slave automata are deterministic, and (2) slave automata recognize prefix-free languages, i.e., languages $\mathcal{L}$ such that if $w \in \mathcal{L}$, then no proper extension of w belongs to $\mathcal{L}$. Condition (2) implies that no accepting run of a slave automaton visits an accepting state twice. Intuitively, slave automata have to accept the first time they encounter an accepting state as they will not visit an accepting state again.

▶ **Definition 2** (Width of NWA). An NWA has *width k* if and only if in every run at every position at most k slave automata are active.

▶ **Example 3** (Non-overlapping ART). We consider a variant of the ART property, called the 1-ART property, where after a request till it is granted additional requests are not considered. Formally, we consider the ART property over the language $\mathcal{L}_1$ defined by $(r\#^*g\#^*)^\omega$ (equivalently, given a request, the automata can check if the slave automaton is not active, and only then invoke it). An NWA $\mathbb{A}_1$ computing the ART property over $\mathcal{L}_1$ is obtained from the NWA for the ART property (see [16]) by taking the product of the master automaton $\mathcal{A}_{mas}$ with an automaton recognizing the language $\mathcal{L}_1$. The automaton $\mathbb{A}_1$ is a deterministic (LimAvg; Sum$^+$)-automaton. Indeed, the master automaton and the slave automata are deterministic and the slave automata recognize prefix-free languages. Moreover, in any (infinite) run at most one slave automaton is active, i.e., $\mathbb{A}_1$ has width 1. The dummy slave automata do not increase the width as they immediately accept, and hence they are not considered as active even at the position they are invoked. Finally, observe that the 1-ART property can return unbounded values, which implies that there exists no (non-nested) LimAvg-automaton expressing it. Also see Example 3 of the full version [17].

Decision problems. The classical questions in automata theory are language *emptiness* and *universality*. These problems have their counterparts in the quantitative setting of weighted automata and NWA. The (quantitative) emptiness and universality problems are defined in the same way for weighted automata and NWA; in the following definition the automaton $\mathcal{A}$ can be either a weighted automaton or an NWA.

- **Emptiness and universality**: Given an automaton $\mathcal{A}$ and a threshold λ, the *emptiness* (resp. *universality*) problem asks whether there exists a word w with $\mathcal{L}_{\mathcal{A}}(w) \leq \lambda$ (resp., for every word w we have $\mathcal{L}_{\mathcal{A}}(w) \leq \lambda$).

▶ Remark. In this work we focus on value functions Sum and Sum$^+$ for finite words, and LimAvg for infinite words. There are other value functions for finite words, such as Max, Min and bounded sum. However, it was shown in [16] that for these value functions, there is a reduction to non-nested weighted automata. Also for infinite words, there are other value functions such as Sup, LimSup, where the complexity and decidability results have been established in [16]. Hence in this work we focus on the most conceptually interesting case of LimAvg function for master automaton and Sum and Sum$^+$ value functions for the slave automata.

3 Examples

We present several examples of properties that can be specified with NWA of bounded width.

▶ **Example 4** (Variants of ART). Recall the ART property (Example 1) and its variant 1-ART property (Example 3). We present two variants of the ART property.

First, we extend Example 3 and consider the k-ART property over languages L_k defined by $(\#^*r(\#^*r\#^*)^{\leq k-1}g\#^*)^\omega$, i.e., the language where there are at most k-pending requests before each grant. As Example 3, an NWA $\mathbb{A}_k$ computing the k-ART property can be constructed from the NWA from Example. 3 by taking the product of the master automaton $\mathcal{A}_{mas}$ of the ART property with an automaton recognizing $\mathcal{L}_k$. The NWA $\mathbb{A}_k$ has width k.

Second, we consider the 1-ART[k] property, where $\Sigma = \{r_i, g_i : i \in \{1, \ldots, k\}\} \cup \{\#\}$, i.e., there are k-different types of "request-grant" pairs. The 1-ART[k] property asks for the average number of instructions between any request and the following grant of the corresponding type. Moreover, we consider as for 1-ART property that for every i, between a request r_i and the following grant of the corresponding type g_i, there is no request r_i of the same type. The 1-ART$[k]$ can be expressed with an $(\textsc{LimAvg}; \textsc{Sum}^+)$-automaton $\mathbb{A}_1^{[k]}$ of width bounded by k, which is similar to $\mathbb{A}_1$ from Example 3. Basically, the NWA $\mathbb{A}_1^{[k]}$ has k slave automata; for $i \in \{1, \ldots, k\}$ the slave automaton $\mathfrak{B}_i$ is invoked on letters r_i and it counts the number of steps to the following grant g_i. Additionally, the master automaton checks that for every i, between any two grants g_i, there is at most one request r_i.

In Examples 1, 3, and 4 we presented properties that can be expressed with $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata. The following property of *average excess* can be expressed with slave automata with $\textsc{Sum}$ value functions that have both positive and negative weights, i.e., it can be expressed by an $(\textsc{LimAvg}; \textsc{Sum})$-automaton, but not $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata.

▶ **Example 5** (Average excess). Consider the alphabet $\{r, g, \#\}$ from Example 1 with an additional letter \$. The *average excess* (AE) property asks for the average difference between requests and grants over blocks separated by \$. For example, for $\$(rr\#g\$)^\omega$ the average excess is 1. The AE property can be expressed by $(\textsc{LimAvg}; \textsc{Sum})$-automaton $\mathbb{A}_{\mathrm{AE}}$ of width 1 (presented below), but it cannot be expressed with $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata; $(\textsc{LimAvg}; \textsc{Sum}^+)$-automata return values form the interval $[0, \infty)$, while AE ranges from $(-\infty, \infty)$. The automaton $\mathbb{A}_{\mathrm{AE}}$ invokes a slave automaton $\mathfrak{B}_1$ at positions of letter \$ and a dummy automaton $\mathfrak{B}_2$ on the remaining positions. The slave automaton $\mathfrak{B}_1$ runs until it sees \$ letter; it computes the difference between the number of r and g letters by taking transitions of weights $1, -1, 0$ respectively on letters $r, g, \#$. The master automaton as well as the slave automata of $\mathbb{A}_{\mathrm{AE}}$ are deterministic and the slave automata recognize prefix-free languages. Therefore, the NWA $\mathbb{A}_{\mathrm{AE}}$ is deterministic and has width 1.

4 Our Results

In this section we establish our main results. We first discuss complexity of checking whether a given NWA has width k. Next, we comment the results we need to prove.

Configurations. Let $\mathbb{A}$ be a non-deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton of width k. We define a *configuration* of $\mathbb{A}$ as a tuple $(q; q_1, \ldots, q_k)$ where q is a state of the master automaton and each $q_1, \ldots, q_k$ is either a state of a slave automaton of $\mathbb{A}$ or $\bot$. In the sequence $q_1, \ldots, q_k$ each state corresponds to one slave automaton, and the states are ordered w.r.t. the position when the corresponding slave automaton has been invoked, i.e., q_1 correspond to the least recently invoked slave automaton. If there are less than k slave automata active, then $\bot$ symbols follow the actual states (denoting there is no slave automata invoked). We define $\textsc{Conf}(\mathbb{A})$ as the number of configurations of $\mathbb{A}$.

Key ideas. NWA without weights are equivalent to Büchi automata [16]. The property of having width k is independent from weights. It can be decided with a construction of a (non-weighted) Büchi automaton, which tracks configurations $(q; q_1, \ldots, q_k)$ of a given NWA (assuming width k) and accepts only if the width-k condition is at some point violated.

▶ **Theorem 6.** *(1) Fix $k > 0$. We can check in polynomial time whether a given NWA has width k. (2) Given an NWA and a number k given in unary we can check in polynomial space whether the NWA has width k.*

Comment. We first note that for deterministic automata, emptiness and universality questions are similar. Hence we focus on the emptiness problem for non-deterministic automata (which subsumes the emptiness problem for deterministic automata) to establish the new results of Table 1. Moreover, the $\textsc{Sum}^+$ value function is a special case of the $\textsc{Sum}$ value function with only positive weights. Since our main results are algorithms to establish upper bounds, we will only present the result for the emptiness problem for non-deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automata. However, as a first step we show that without loss of generality, we can focus on the case of deterministic automata.

▶ **Lemma 7.** *Let $k > 0$. Given a non-deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton $\mathbb{A}$ over alphabet Σ of width k, a deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton $\mathbb{A}_d$ of width k over an alphabet $\Sigma \times \Gamma$ such that $\inf_{u \in \Sigma^\omega} \mathbb{A}(u) = \inf_{u' \in (\Sigma\times\Gamma)^\omega} \mathbb{A}_d(u')$ can be constructed in time exponential in k and polynomial in $|\mathbb{A}|$. Moreover, $\textsc{Conf}(\mathbb{A}_d)$ is polynomial in $\textsc{Conf}(\mathbb{A})$ and k and only the alphabet of $\mathbb{A}_d$ is exponential (in k) as compared to the alphabet of $\mathbb{A}$.*

Proof sketch. The main idea is that the part Γ of the alphabet encodes the possible non-deterministic choices, and the possible non-deterministic choices basically correspond to transitions between configurations.

Proof overview. We present our proof overview for the emptiness of deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automata. The proof consists of the following four key steps.

1. First, we identify a condition, and show in Lemma 9 that it is sufficient to ensure that the infimum value among all words is $-\infty$ (i.e., the least value possible). Moreover we show that the condition can be decided in $\textsc{PTime}$ if k is constant (even $\textsc{NLogSpace}$ if additionally the weights are in unary) and in $\textsc{PSpace}$ if k is given in unary.
2. Second, we show that if the above condition does not hold, then there is a family of lasso words (i.e., a finite prefix followed by an infinite repetition of another finite word) that approximates the infimum value among all words. This shows that the above condition is both necessary and sufficient. Moreover, we consider *dense* words, in which an i-th invoked slave automaton runs for at most for $O(\log(i))$ steps. We show that the infimum is achieved by a dense word. These results are established in Lemma 11.
3. Third, we show using the above result, that the problem for bounded width can be reduced to the problem of width 1, and the reduction is polynomial in the size of the original automaton, and only exponential in k. Thus if k is constant, the reduction is polynomial. This is established in Lemma 12.
4. Finally, we show that for automata with width 1, the emptiness problem can be solved in $\textsc{NLogSpace}$ if weights are in unary and otherwise in $\textsc{PTime}$ (Lemma 13).

The above four steps give our main result (Theorem 14). We start with the first item.

Intuition for the condition. We first illustrate with an example that for very similar automata, which just differ in order of invoking slave automata, the infima over the values are very different. For one automaton the infimum value is $-\infty$ and for the other it is 0. This example provides the intuition for the need of the condition to identify when the infimum value is $-\infty$.

▶ **Example 8.** Consider two deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automata $\mathbb{A}_1, \mathbb{A}_2$ defined as follows. The master automaton $\mathcal{A}_{mas}$ of $\mathbb{A}_1$ accepts the language $(12a^*\#)^\omega$. At letter 1 (resp., 2) it invokes an automaton $\mathfrak{B}_1$ (resp., $\mathfrak{B}_2$). The slave automaton $\mathfrak{B}_1$ increments its value at every a letter and it terminates once it reads $\#$. The slave automaton $\mathfrak{B}_2$ works as $\mathfrak{B}_1$ except that it decrements its value at a letters. NWA $\mathbb{A}_2$ is similar to $\mathbb{A}_1$ except that it accepts the language $(21a^*\#)^\omega$. It invokes the same slave automata as $\mathbb{A}_1$. Thus the two automata only differ in the order of invocation of the slave automata. Observe that the infimum over values of all words in $\mathbb{A}_1$ is 0. Basically, the values of slave automata are always the opposite, therefore the average of the values of slave automata is 0 infinitely often. However, the infimum over values of all words in $\mathbb{A}_2$ is $-\infty$. Indeed, consider a word $21a^1\# \ldots 21a^{2^i} \ldots$. At positions proceeding $1a^{2^i}$, the automaton $\mathfrak{B}_2$ returns the value -2^i and the average of all previous $2 \cdot i$ values is 0. Thus, the average at this position equals $-\frac{2^i}{2 \cdot i}$ (recall that the average is over the number of invocations of slave automata). Hence, the limit infimum of averages is $-\infty$.

Condition for infinite infimum. Let $k > 0$ and $\mathbb{A}$ be a deterministic ($\textsc{LimAvg}; \textsc{Sum}$)-automaton of width k. Let C be the minimal weight of slave automata of $\mathbb{A}$. Condition (*):

(*) $C < 0$ and there exists a word w accepted by $\mathbb{A}$ and infinitely many positions b such that the sum of weights, which automata active at position b accumulate while running on $w[b, \infty]$, is less than $C \cdot k^2 \cdot \textsc{Conf}(\mathbb{A})$.

Intuitively, condition (*) implies that there is a subword u which can be repeated so that the values of slave automata invoked before position b can be decreased arbitrarily. Note that pumping that word may not decrease the total average of the word. However, with $\textsc{LimAvg}$ value function, we need to ensure only the existence of a subsequence of positions at which the averages tend to $-\infty$, i.e., we only need to decrease the values of slave automata invoked before position b (for infinitely many positions).

Illustration of condition on example. Consider automata $\mathbb{A}_1, \mathbb{A}_2$ from Example 8. The automaton $\mathbb{A}_2$ satisfies condition (*), whereas $\mathbb{A}_1$ does not. In the word $21a^1\# \ldots 21a^{2^i} \ldots$, consider positions b, where $\mathfrak{B}_2$ is invoked by $\mathbb{A}_2$. The automaton $\mathfrak{B}_2$ works on the subword $21a^{2^i}$, where both automata $\mathfrak{B}_1, \mathfrak{B}_2$ are active and the sum of their values past any position is 0. However, the only slave automaton active at position b is $\mathfrak{B}_2$. These automaton accumulates the value -2^i past position b. Therefore, past some position N, all such positions b satisfy the statement from condition (*), and hence $\mathbb{A}_2$ satisfies condition (*). Now, for $\mathbb{A}_1$, at every position at which $\mathfrak{B}_2$ is active, $\mathfrak{B}_1$ is active as well, hence for any position b, the values accumulated by slave automaton active past this position is non-negative. Hence, $\mathbb{A}_1$ does not satisfy condition (*). We now present our lemma about the condition.

▶ **Lemma 9.** *Let $k > 0$ and $\mathbb{A}$ be a deterministic* ($\textsc{LimAvg}; \textsc{Sum}$)*-automaton of width k.*

1. *If condition (*) holds for $\mathbb{A}$, then* $\inf_{u \in \Sigma^\omega} \mathbb{A}(u) = -\infty$.
2. *Condition (*) can be checked in* NLogSpace *for constant width and weights in unary,* PTime *for constant width, and in* PSpace *if the width is given in unary.*

Key intuitions. For (1), we show that the word from condition (*) can be pumped at some positions to achieve a word u' with $\mathbb{A}(u') = -\infty$. For (2), we show that condition (*) holds if and only if there exists a cycle in the graph of configurations of $\mathbb{A}$, which (a) can be visited infinitely often, and (b) for some $j \geq 1$, the sum of weights in this cycle of the j least recently invoked slave automata is negative. Recall that the order of invocation of slave automaton is encoded in the configuration, i.e., in $(q; q_1, \ldots q_j, q_{j+1}, \ldots, q_k)$ slave automata that correspond to the states $q_1, \ldots, q_j$ are the j least recently invoked.

▶ **Definition 10.** Let $\mathbb{A}$ be a deterministic ($\textsc{LimAvg}; \textsc{Sum}$)-automaton of width k. A word w is *dense* (w.r.t. $\mathbb{A}$) if in the run of $\mathbb{A}$ on w, for every $i > 0$, the i-th invoked slave automaton takes at most $O(\log(i))$ steps.

Intuitive explanation of dense words. In a deterministic ($\textsc{LimAvg}; \textsc{Sum}$)-automaton, the average is over the number of invoked slave automata, but in general, the returned values of

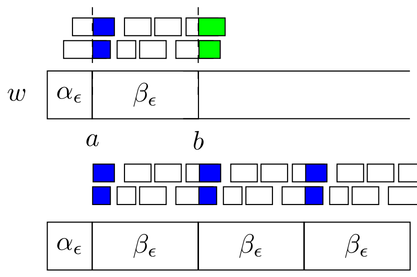


Figure 1 Explanation to Lemma 11; the blue part corresponds to H, while the green part corresponds to T.

the slave automata can be arbitrarily large as compared to the number of invocations, and hence the partial averages need not converge. Intuitively, in dense words, slave automata are invoked and terminated relatively densely, i.e., the length of their runs depends on the number of slave automata invoked till this position. In consequence, the value they can accumulate is small w.r.t. the average, i.e., their absolute contribution to the sum of the first n elements is $O(\log(n))$, and hence the contribution of the value a single slave automaton converges to 0 and the partial averages converge on dense words.

Illustration on example. Consider an automaton $\mathbb{A}_1$ from Example 8. We discuss the definition of density on an example of word $w = 12a^1\#12a^3\#\dots 12a^{2\cdot i+1}\#\dots$, which is not dense (w.r.t. $\mathbb{A}_1$). Observe that at the position of subword $12a^{2\cdot i+1}$, the partial average is 0. Once $\mathfrak{B}_1$ is invoked it returns value $2\cdot i+1$ and it is $(2\cdot i+1)$-th invocation of a slave automaton. Hence, the average increases to 1 only to be decreased to 0 after invocation of $\mathfrak{B}_2$. Now, word $w' = 12a^1\#(12a^2\#)^3\dots(12a^{2\cdot i+1}\#)^{2^i}\dots$ is dense. Indeed, before the slave automaton invoked at subword $12a^{2\cdot i+1}\#$ there are at least $\sum_{j=1}^{i-1} 2^j = 2^i - 1$ invoked slave automata. Therefore, the value $2\cdot i+1$ returned by $\mathfrak{B}_1$ invoked on $12a^{2\cdot i+1}$ is logarithmic in the number of invoked slave automata $2^i - 1$ and it changes the average by at most $\frac{2\cdot i+1}{2^i}$; as previously invoking $\mathfrak{B}_2$ in the next step bring the average back to 0. Thus, the sequence of partial averages of values returned by slave automata converges to 0.

▶ **Lemma 11.** *Let $k > 0$ and $\mathbb{A}$ be a deterministic* (LimAvg; Sum)*-automaton of width k. Assume that condition (*) does not hold. Then the following assertions hold:*
1. *For every $\epsilon > 0$ there exist finite words $\alpha_\epsilon, \beta_\epsilon$ such that $|\inf_{u\in\Sigma^\omega} \mathbb{A}(u) - \mathbb{A}(\alpha_\epsilon(\beta_\epsilon)^\omega)| < \epsilon$.*
2. *The value $\inf_{u\in\Sigma^\omega} \mathbb{A}(u)$ is greater than $-\infty$.*
3. *There exists a dense word w_d such that $\inf_{u\in\Sigma^\omega} \mathbb{A}(u) = \mathbb{A}(w_d)$.*

Proof sketch: We present the key ideas for each item (detailed proof in [17]). Assume that condition (*) fails.
1. We consider $\epsilon > 0$ and a word w_ϵ, which is $\frac{\epsilon}{4}$-close to the infimum over all values of $\mathbb{A}$. We show that w_ϵ contains a subword β_ϵ on which (a) the automaton $\mathbb{A}$ starts and ends with the same configuration, and (b) the average of the values returned by the slave automata is at most $\mathbb{A}(w_\epsilon) + \frac{\epsilon}{4}$. The existence of such a word follows from the fact that the partial averages are infinitely often $\frac{\epsilon}{4}$-close to the value of w_ϵ. We then show that β_ϵ together with α_ϵ, the prefix preceding β_ϵ, satisfy $|\inf_{u\in\Sigma^\omega} \mathbb{A}(u) - \mathbb{A}(\alpha_\epsilon(\beta_\epsilon)^\omega)| < \epsilon$. If we consider the sequence of values returned by slave automata on the word β_ϵ^ω, then it differs from the sequence of values returned when we consider the corresponding suffix in w_ϵ: this is because the values of slave automata in β_ϵ as a subword of w_ϵ depend on the following letters. The difference of partial averages can be bounded with an estimate

of $H - T$ which is defined below. Consider the subword $\alpha_\epsilon \cdot \beta_\epsilon$. Let X be the set of slave automata that are active when β_ϵ is invoked (i.e., after α_ϵ). Let H be the sum of weights of the active slave automata in X accumulated in the part of their respective runs on β_ϵ. Let Y be the set of active slave automata after $\alpha_\epsilon \cdot \beta_\epsilon$. Let T be the sum of weights of the active slave automata in Y accumulated in the part of their respective runs on w_ϵ past $\alpha_\epsilon \cdot \beta_\epsilon$. See Fig 1 for an illustration. We establish an estimate on $H - T$ using the fact that (*) does not hold.

2. Almost all slave automata invoked in the run of $\mathbb{A}$ on a word of the form $\alpha(\beta)^\omega$ take at most $|\beta|$ steps. Only slave automata invoked at α can take more steps without looping. Thus, the value $\mathbb{A}(\alpha(\beta)^\omega)$ is finite. Therefore, (1) implies that $\inf_{u \in \Sigma^\omega} \mathbb{A}(u)$ is finite (given some words are accepted and condition(*) fails.)
3. We construct w_d from a word $\beta_1^{k[1]}\beta_{\frac{1}{2}}^{k[2]}\beta_{\frac{1}{3}}^{k[3]}\ldots$ by choosing the sequence $k[0], k[1], \ldots$ to increase sufficiently fast. By repeating $k[n]$ times word $\beta_{\frac{1}{n}}$, we increase the number of invoked slave automata at least by $k[n]$, so the number of steps of slave automaton invoked in $\beta_{\frac{1}{n+1}}$, which is bounded by $|\beta_{\frac{1}{n+1}}|$, can be made arbitrarily small w.r.t. $k[n]$.

▶ Remark. Lemma 9 together with (2) of Lemma 11 imply that for a deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton $\mathbb{A}$ of width k condition (*) is both necessary and sufficient for the infimum over all values equal to $-\infty$. Moreover, this condition can be checked efficiently.

Lemma 12 reduces the emptiness problem for deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automata of width k to the same problem with automata of width 1.

▶ **Lemma 12.** *Let $k > 0$ and $\mathbb{A}$ be a deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton of width k. Assume that condition (*) does not hold. Then, there exists a deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton $\mathbb{A}_1$ of width 1 over an alphabet Δ such that $\inf_{u \in \Sigma^\omega} \mathbb{A}(u) = \inf_{u \in \Delta^\omega} \mathbb{A}_1(u)$. The size of $\mathbb{A}_1$ is $O(|\mathbb{A}|^k)$ and it can be constructed on-the-fly.*

Key intuitions. Consider a deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton $\mathbb{A}$ of width k. We define the automaton $\mathbb{A}_1$, which uses a single slave automaton to keep track of all k automata of $\mathbb{A}$. This single slave automaton takes transitions whose weight is the sum of weights of transitions of tracked slave automata of $\mathbb{A}$. Therefore, $\mathbb{A}$ and $\mathbb{A}_1$ compute the averages of the same weights. Still, the way these weights are aggregated, i.e., their order in the sequence is different, and hence these automata may return different values on the same word. However, we show that on dense words the values of both automata coincide. This and Lemma 11 stating that there exists a dense word at which the automaton $\mathbb{A}_1$ realizes its infimum implies that the infimum over all values of $\mathbb{A}$ and $\mathbb{A}_1$ coincide.

▶ **Lemma 13.** *The emptiness problem for deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automata of width 1 is in $\textsc{PTime}$ and if the weights are in unary, then it is in $\textsc{NLogSpace}$.*

Key intuitions. We show that every transition of $\mathcal{A}_{mas}$, the master automaton of $\mathbb{A}$, at which a slave automaton is invoked, can be substituted by a transition whose weight is the minimal value the invoked slave automaton can achieve. More precisely, while a slave automaton is running on the input word, the master automaton $\mathcal{A}_{mas}$ is still active. Therefore, we substitute transitions (q, a, q', i) of $\mathcal{A}_{mas}$ by multiple transitions of the form $(q, (q, a, i, q''), q'')$, where (q, a, i, q'') is a new letter, q'' is a state of $\mathcal{A}_{mas}$ and the weight of this transition is the minimal value $\mathfrak{B}_i$ can achieve over words au such that $\mathcal{A}_{mas}$ moves from q to q'' upon reading au. Such a transformation preserves the infimum over all words and it transforms a deterministic $(\textsc{LimAvg}; \textsc{Sum})$-automaton of width 1 to a deterministic

LimAvg-automaton. The emptiness problem for LimAvg-automaton is decidable in PTime and even in NLogSpace provided that weights are given in unary.

We now present the algorithm and lower bound for our main result.

The algorithm. We present an algorithm, which, given a non-deterministic (LimAvg, Sum)-automaton $\mathbb{A}$ of width k and $\lambda \in \mathbb{Q}$, decides whether $\inf_{u \in \Sigma^\omega} \mathbb{A}(u) \leq \lambda$.

1. Transform $\mathbb{A}$ into a deterministic (LimAvg, Sum)-automaton $\mathbb{A}_d$ of the same width such that $\inf_{u \in \Sigma^\omega} \mathbb{A}(u) = \inf_{u \in (\Sigma \times \Gamma)^\omega} \mathbb{A}_d(u)$ (Lemma 7).
2. Check condition (*) for $\mathbb{A}_d$. If it holds, then $\inf_{u \in \Sigma^\omega} \mathbb{A}(u) = -\infty$ and return answer **YES**. Otherwise, continue the algorithm.
3. Transform $\mathbb{A}_d$ into a deterministic (LimAvg, Sum)-automaton $\mathbb{A}_1$ of width 1 such that $\inf_{u \in (\Sigma \times \Gamma)^\omega} \mathbb{A}_d(u) = \inf_{u \in \Delta^\omega} \mathbb{A}_1(u)$ (Lemma 12).
4. Compute $\inf_{u \in \Delta^\omega} \mathbb{A}_1(u)$ (Lemma 13), and return whether $\inf_{u \in \Delta^\omega} \mathbb{A}_1(u) \leq \lambda$.

Transformations in (1) and (3) are polynomial in the size of the automaton and exponential in k. Also, transformation from (1) does not increase k. Therefore, the size of $\mathbb{A}_1$ is polynomial in the size $\mathbb{A}$ and singly exponential in k. Moreover, these transformations can be done on-the-fly, i.e., there is not need to store the whole resulting automaton. Therefore, checks from (2) and (4), can be done in NLogSpace if k is constant and weights are in unary, PTime if k is constant, and PSpace if k is given in unary.

Hardness results. If k is constant, then the reachability problem on directed graphs, which is NLogSpace-complete, can be reduced to language emptiness of a finite automaton, which is a special case the emptiness problem for non-deterministic (LimAvg, Sum)-automata of width 1 with unary weights. If k is given in unary, consider the emptiness problem for the intersection of regular languages, which given k and regular languages $\mathcal{L}_1, \ldots, \mathcal{L}_k$, asks whether $\mathcal{L}_1 \cap \ldots \cap \mathcal{L}_k = \emptyset$. This problem is PSpace-complete [26] and reduces to the emptiness problem for deterministic (LimAvg, Sum)-automata of width given in unary: the PSpace-hardness result for emptiness of NWA given in [16] uses NWA of width $|\mathbb{A}|$.

▶ **Theorem 14.** *The emptiness problem for non-deterministic* (LimAvg, Sum)*-automata is (a)* NLogSpace*-complete in the size of $\mathbb{A}$ for constant width k with weights in unary; (b)* PTime *in the size of $\mathbb{A}$ for constant width k; and (c)* PSpace*-complete when the bounded width k is given as input in unary.*

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. Discounting in LTL. In *TACAS, 2014*, pages 424–439, 2014.
2 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular functions and cost register automata. In *LICS 2013*, pages 13–22, 2013.
3 Christel Baier, Clemens Dubslaff, and Sascha Klüppelholz. Trade-off analysis meets probabilistic model checking. In *CSL-LICS 2014*, pages 1:1–1:10, 2014.
4 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Weight monitoring with linear temporal logic: complexity and decidability. In *CSL-LICS 2014*, pages 11:1–11:10, 2014.
5 Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. *ACM TOCL*, 15(4):27:1–27:25, 2014.
6 Benedikt Bollig, Paul Gastin, Benjamin Monmege, and Marc Zeitoun. Pebble weighted automata and transitive closure logics. In *ICALP 2010, Part II*, pages 587–598. Springer, 2010.

7 Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *CONCUR 2014*, pages 266–280, 2014.

8 Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Two views on multiple mean-payoff objectives in Markov decision processes. In *LICS 2011*, pages 33–42, 2011.

9 Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Multigain: A controller synthesis tool for MDPs with multiple mean-payoff objectives. In *TACAS 2015*, pages 181–187, 2015.

10 Krishnendu Chatterjee. Markov decision processes with multiple long-run average objectives. In *FSTTCS*, pages 473–484, 2007.

11 Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov Decision Processes. In *MFCS 2011*, pages 206–218, 2011.

12 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Alternating weighted automata. In *FCT'09*, pages 3–13. Springer, 2009.

13 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *LMCS*, 6(3), 2010.

14 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM TOCL*, 11(4):23, 2010.

15 Krishnendu Chatterjee, Vojtech Forejt, and Dominik Wojtczak. Multi-objective discounted reward verification in graphs and MDPs. In *LPAR*, pages 228–242, 2013.

16 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted automata. In *LICS 2015*, pages 725–737, 2015.

17 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted limit-average automata of bounded width. *CoRR*, abs/1606.03598, 2016. A conference version accepted to MFCS 2016. URL: `http://arxiv.org/abs/1606.03598`.

18 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative automata under probabilistic semantics. *CoRR*, abs/1604.06764, 2016. A conference version accepted to LICS 2016. URL: `http://arxiv.org/abs/1604.06764`.

19 Krishnendu Chatterjee, Zuzana Komárková, and Jan Kretínský. Unifying two views on multiple mean-payoff objectives in Markov Decision Processes. In *LICS 2015*, pages 244–256, 2015.

20 Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. Markov Decision Processes with multiple objectives. In *STACS 2006*, pages 325–336, 2006.

21 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

22 Manfred Droste and George Rahonis. Weighted automata and weighted logics on infinite words. In *DLT 2006*, pages 49–58, 2006.

23 Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer, 1996.

24 Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Quantitative multi-objective verification for probabilistic systems. In *TACAS*, pages 112–127, 2011.

25 Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In *CONCUR 2013*, pages 273–287, 2013.

26 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE Computer Society, 1977.

27 Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *J. Aut. Lang. & Comb.*, 7(3):321–350, 2002.

28 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1st edition, 1994.

Conditionally Optimal Algorithms for Generalized Büchi Games*

Krishnendu Chatterjee[1], Wolfgang Dvořák[2], Monika Henzinger[3], and Veronika Loitzenbauer[4]

1 **IST Austria**
2 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
3 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
4 **University of Vienna, Faculty of Computer Science, Vienna, Austria**

Abstract

Games on graphs provide the appropriate framework to study several central problems in computer science, such as verification and synthesis of reactive systems. One of the most basic objectives for games on graphs is the liveness (or Büchi) objective that given a target set of vertices requires that some vertex in the target set is visited infinitely often. We study generalized Büchi objectives (i.e., conjunction of liveness objectives), and implications between two generalized Büchi objectives (known as GR(1) objectives), that arise in numerous applications in computer-aided verification. We present improved algorithms and conditional super-linear lower bounds based on widely believed assumptions about the complexity of (A1) combinatorial Boolean matrix multiplication and (A2) CNF-SAT. We consider graph games with n vertices, m edges, and generalized Büchi objectives with k conjunctions. First, we present an algorithm with running time $O(k \cdot n^2)$, improving the previously known $O(k \cdot n \cdot m)$ and $O(k^2 \cdot n^2)$ worst-case bounds. Our algorithm is optimal for dense graphs under (A1). Second, we show that the basic algorithm for the problem is optimal for sparse graphs when the target sets have constant size under (A2). Finally, we consider GR(1) objectives, with k_1 conjunctions in the antecedent and k_2 conjunctions in the consequent, and present an $O(k_1 \cdot k_2 \cdot n^{2.5})$-time algorithm, improving the previously known $O(k_1 \cdot k_2 \cdot n \cdot m)$-time algorithm for $m > n^{1.5}$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Generalized Büchi objective, GR(1) objective, Conditional lower bounds, Graph games, Graph algorithms, Computer-aided verification

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.25

1 Introduction

Games on graphs. Two-player games on graphs, between player 1 and the adversary player 2, are central in many problems in computer science, specially in formal analysis of reactive systems, where vertices of the graph represent states of the system, edges represent transitions, infinite paths of the graph represent behaviors (or non-terminating executions)

* K. C., M. H., and W. D. are partially supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-003. K. C. is partially supported by the Austrian Science Fund (FWF) NFN Grant No S11407-N23 (RiSE/SHiNE) and an ERC Start grant (279307: Graph Games). For W. D., M. H., and V. L. the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

of the system, and the two players represent the system and the environment, respectively. Games on graphs have been used in many applications related to verification and synthesis of systems, such as, synthesis of systems from specifications and controller-synthesis [30, 54, 55], verification of open systems [8], checking interface compatibility [31], well-formedness of specifications [32], and many others. We will distinguish between results most relevant for *sparse graphs*, where the number of edges m is roughly proportional to the number of vertices n, and *dense graphs* with $m = \Theta(n^2)$. Sparse graphs arise naturally in program verification, as control-flow graphs are sparse [57, 28]. Graphs obtained as synchronous product of several components (where each component makes transitions at each step) [45, 23] can lead to dense graphs.

Objectives. Objectives specify the desired set of behaviors of the system. The most basic objective for reactive systems is the *reachability* objective, and the next basic objective is the *Büchi* (*also called liveness or repeated reachability*) objective that was introduced in the seminal work of Büchi [17, 18, 19] for automata over infinite words. Büchi objectives are specified with a target set T and the objective specifies the set of infinite paths in the graph that visit some vertex in the target set infinitely often. Since for reactive systems there are multiple requirements, a very central objective to study for games on graphs is the conjunction of Büchi objectives, which is known as generalized Büchi objective. Finally, currently a very popular class of objectives to specify behaviors for reactive systems is called the GR(1) (generalized reactivity (1)) objectives [53]. A GR(1) objective is an implication between two generalized Büchi objectives.

We present a brief discussion about the significance of the objectives we consider, for a detailed discussion see [26]. The conjunction of liveness objectives is required to specify progress conditions of mutual exclusion protocols, and deterministic Büchi automata can express many important properties of linear-time temporal logic (LTL) (the de-facto logic to specify properties of reactive systems) [47, 46, 9, 44]. The analysis of reactive systems with such objectives naturally gives rise to graph games with generalized Büchi objectives. Finally, graph games with GR(1) objectives have been used in many applications, such as the industrial example of synthesis of AMBA AHB protocol [14, 36] as well as in robotics applications [35, 21].

Basic problem and conditional lower bounds. In this work we consider games on graphs with generalized Büchi and GR(1) objectives, and the basic algorithmic problem is to compute the *winning set*, i.e., the set of starting vertices where player 1 can ensure the objective irrespective of the way player 2 plays; the way player 1 achieves that is called her *winning strategy.* These are core algorithmic problems in verification and synthesis. For the problems we consider, while polynomial-time algorithms are known, there are no super-linear lower bounds. Since for polynomial-time algorithms unconditional super-linear lower bounds are extremely rare in the whole of computer science, we consider *conditional lower bounds*, which assume that for some well-studied problem the known algorithms are optimal up to some lower-order factors. In this work we consider two such well-studied assumptions: (A1) there is no combinatorial[1] algorithm with running time of $O(n^{3-\varepsilon})$ for any $\varepsilon > 0$ to multiply two $n \times n$ Boolean matrices; or (A2) for all $\varepsilon > 0$ there exists a k such that there is no algorithm for the k-CNF-SAT problem that runs in $O(2^{(1-\varepsilon)\cdot n} \cdot \text{poly}(m))$ time, where n is the number of variables and m the number of clauses. These two assumptions have been used to establish

[1] Combinatorial here means avoiding fast matrix multiplication [48], see also the discussion in [38].

lower bounds for several well-studied problems, such as dynamic graph algorithms [3, 5], measuring the similarity of strings [4, 15, 16, 10, 2], context-free grammar parsing [49, 1], and verifying first-order graph properties [52, 61].

Our results. We consider games on graphs with n vertices, m edges, generalized Büchi objectives with k conjunctions, and target sets of size $b_1, b_2, \ldots, b_k$, and GR(1) objectives with k_1 conjunctions in the assumptions and k_2 conjunctions in the guarantee. Our results are as follows.

- *Generalized Büchi objectives.* The classical algorithm for generalized Büchi objectives requires $O(k \cdot \min_{1 \le i \le k} b_i \cdot m)$ time. Further there exists an $O(k^2 \cdot n^2)$-time algorithm via a reduction to Büchi games [13, 26].
  1. *Dense graphs.* Since $\min_{1 \le i \le k} b_i = O(n)$ and $m = O(n^2)$, the classical algorithm has a worst-case running time of $O(k \cdot n^3)$. First, we present an algorithm with worst-case running time $O(k \cdot n^2)$, which is an improvement for instances with $\min_{1 \le i \le k} b_i \cdot m = \omega(n^2)$. Second, for dense graphs with $m = \Theta(n^2)$ and $k = \Theta(n^c)$ for any $0 < c \le 1$ our algorithm is optimal under (A1); i.e., improving our algorithm for dense graphs would imply a faster (sub-cubic) combinatorial Boolean matrix multiplication algorithm.
  2. *Sparse graphs.* We show that for $k = \Theta(n^c)$ for any $0 < c \le 1$, for target sets of constant size, and sparse graphs with $m = \Theta(n^{1+o(1)})$ the basic algorithm is optimal under (A2). In fact, our conditional lower bound under (A2) holds even when each target set is a singleton. Quite strikingly, our result implies that improving the basic algorithm for sparse graphs even with singleton sets would require a major breakthrough in overcoming the exponential barrier for SAT.

  In summary, for games on graphs, we present an improved algorithm for generalized Büchi objectives for dense graphs that is optimal under (A1); and show that under (A2) the basic algorithm is optimal for sparse graphs and constant size target sets.

  The conditional lower bound for dense graphs means in particular that for unrestricted inputs the dependence of the runtime on n cannot be improved, whereas the bound for sparse graphs makes the same statement for the dependence on m. Moreover, as the graphs in the reductions for our lower bounds can be made acyclic by deleting a single vertex, our lower bounds also apply to a broad range of digraph parameters. For instance let w be the DAG-width [12] of a graph, then there is no $O(f(w) \cdot n^{3-\epsilon})$-time algorithm under (A1) and no $O(f(w) \cdot m^{2-\epsilon})$-time algorithm under (A2).
- *GR(1) objectives.* We present an algorithm for games on graphs with GR(1) objectives that has $O(k_1 \cdot k_2 \cdot n^{2.5})$ running time and improves the previously known $O(k_1 \cdot k_2 \cdot n \cdot m)$-time algorithm [43], for $m > n^{1.5}$. Note that since generalized Büchi objectives are special cases of GR(1) objectives, our conditional lower bounds for generalized Büchi objectives apply to GR(1) objectives as well but are not tight.

All our algorithms can easily be modified to also return the corresponding winning strategies for both players within the same time bounds.

Implications. We discuss the implications of our results.

1. *Comparison with related models.* We compare our results for game graphs to the special case of standard graphs (i.e., games on graphs with only player 1) and the related model of Markov decision processes (MDPs) (with only player 1 and stochastic transitions). First note that for reachability objectives, linear-time algorithms exist for game graphs [11, 39],

whereas for MDPs[2] the best-known algorithm has running time $O(\min(n^2, m^{1.5}))$ [29, 26]. For MDPs with reachability objectives, a linear or even $O(m \log n)$ time algorithm is a major open problem, i.e., there exist problems that seem harder for MDPs than for game graphs. Our conditional lower bound results show that under assumptions (A1) and (A2) the algorithmic problem for generalized Büchi objectives is strictly harder for games on graphs as compared to standard graphs and MDPs. More concretely, for $k = \Theta(n)$, (a) for dense graphs ($m = \Theta(n^2)$) and $\min_{1 \le i \le k} b_i = \Omega(\log n)$, our lower bound for games on graphs under (A2) is $\Omega(n^{3-o(1)})$, whereas both the graph and the MDP problems can be solved in $O(n^2)$ time [25, 26]; and (b) for sparse graphs ($m = \Theta(n^{1+o(1)})$) with $\min_{1 \le i \le k} b_i = O(1)$, our lower bound for games on graphs under (A1) is $\Omega(m^{2-o(1)})$, whereas the graph problem can be solved in $O(m)$ time and the MDP problem in $O(m^{1.5})$ time [7, 24]; respectively.

2. *Relation to SAT.* We present an algorithm for game graphs with generalized Büchi objectives and show that improving the algorithm would imply a better algorithm for SAT, and thereby establish an interesting algorithmic connection for classical objectives in game graphs and the SAT problem.

Due to the lack of space, some technical details are omitted. A full version is available at `http://eprints.cs.univie.ac.at/4708/`.

2 Preliminaries

2.1 Basic definitions for Games on Graphs

Game graphs. A *game graph* $\mathcal{G} = ((V, E), (V_1, V_2))$ is a directed graph $G = (V, E)$ with n vertices V and m edges E and a partition of V into *player 1 vertices* V_1 and *player 2 vertices* V_2. Given such a game graph $\mathcal{G}$, we denote with $\overline{\mathcal{G}}$ the game graph where the player 1 and player 2 vertices of $\mathcal{G}$ are interchanged, i.e, $\overline{\mathcal{G}} = ((V, E), (V_2, V_1))$. We use p to denote a player and $\bar{p}$ to denote its opponent. For a vertex $u \in V$, we write $Out(u) = \{v \in V \mid (u, v) \in E\}$ for the set of successor vertices of u and $In(u) = \{v \in V \mid (v, u) \in E\}$ for the set of predecessor vertices of u. If necessary, we refer to the successor vertices in a specific graph by using, e.g., $Out(G, u)$. We denote by $Outdeg(u) = |Out(u)|$ the number of outgoing edges from u, and by $Indeg(u) = |In(u)|$ the number of incoming edges. We assume for technical convenience $Outdeg(u) \ge 1$ for all $u \in V$.

Plays and strategies. A *play* on a game graph is an infinite sequence $\omega = \langle v_0, v_1, v_2, \ldots \rangle$ of vertices such that $(v_\ell, v_{\ell+1}) \in E$ for all $\ell \ge 0$. The set of all plays is denoted by Ω. Given a finite prefix $\omega \in V^* \cdot V_p$ of a play that ends at a player p vertex v, a *strategy* $\sigma : V^* \cdot V_p \to V$ of player p is a function that chooses a successor vertex $\sigma(\omega)$ among the vertices of $Out(v)$. We denote by Σ and Π the set of all strategies for player 1 and player 2 respectively. The play $\omega(v, \sigma, \pi)$ is uniquely defined by a start vertex v, a player 1 strategy $\sigma \in \Sigma$, and a player 2 strategy $\pi \in \Pi$ as follows: $v_0 = v$ and for all $j \ge 0$, if $v_j \in V_1$, then $v_{j+1} = \sigma(\langle v_1, \ldots, v_j \rangle)$, and if $v_j \in V_2$, then $v_{j+1} = \pi(\langle v_1, \ldots, v_j \rangle)$.

Objectives. An objective ψ is a set of plays that is winning for a player. We consider zero-sum games where for a player-1 objective ψ the complementary objective $\Omega \setminus \psi$ is winning

[2] For MDPs the winning set refers to the almost-sure winning set that requires that the objective is satisfied with probability 1.

for player 2. In this work we consider only *prefix independent objectives*, for which the set of desired plays is determined by the set of vertices $\mathrm{Inf}(\omega)$ that occur *infinitely often* in a play ω. Given a target set $T \subseteq V$, a play ω belongs to the *Büchi objective* Büchi (T) iff $\mathrm{Inf}(\omega) \cap T \neq \emptyset$. For the complementary *co-Büchi objective* we have $\omega \in$ coBüchi (T) iff $\mathrm{Inf}(\omega) \cap T = \emptyset$. A *generalized (or conjunctive) Büchi objective* is specified by a set of k target sets T_ℓ for $1 \leq \ell \leq k$ and is satisfied for a play ω iff $\mathrm{Inf}(\omega) \cap T_\ell \neq \emptyset$ for *all* $1 \leq \ell \leq k$. Its complementary objective is the *disjunctive co-Büchi objective* that is satisfied iff $\mathrm{Inf}(\omega) \cap T_\ell = \emptyset$ for *one of* the k target sets. A *generalized reactivity-1* (GR(1)) objective is specified by two generalized Büchi objectives, $\bigwedge_{t=1}^{k_1}$ Büchi (L_t) and $\bigwedge_{\ell=1}^{k_2}$ Büchi (U_ℓ), and is satisfied if whenever the first generalized Büchi objective holds, then also the second generalized Büchi objective holds; in other words, either $\bigvee_{t=1}^{k_1}$ coBüchi (L_t) holds, or $\bigwedge_{\ell=1}^{k_2}$ Büchi (U_ℓ) holds.

All the games in this paper will be given by a game graph $\mathcal{G}$ and an objective ψ for player 1 (player 2 has the complementary objective $\Omega \setminus \psi$).

Winning strategies and sets. A strategy σ is winning for player p at a start vertex v if the resulting play is winning for player p irrespective of the strategy of his opponent, player $\bar{p}$, i.e., $\omega(v, \sigma, \pi) \in \psi$ for all π. A vertex v belongs to the *winning set* W_p of player p if player p has a winning strategy from v. Every vertex is winning for exactly one of the two players [50]. When required for explicit reference of a specific game graph $\mathcal{G}$ and objective ψ, we use $W_p(\mathcal{G}, \psi)$ to refer to the winning sets.

Closed sets and attractors. A set $U \subseteq V$ is *p-closed* (in $\mathcal{G}$) if for all p-vertices u in U we have $Out(u) \subseteq U$ and for all $\bar{p}$-vertices v in U there exists a vertex $w \in Out(v) \cap U$. Note that player $\bar{p}$ can ensure that a play that currently ends in a p-closed set never leaves the p-closed set against any strategy of player p by choosing an edge (v, w) with $w \in Out(v) \cap U$ whenever the current vertex v is in $U \cap V_{\bar{p}}$ [62]. Given a game graph $\mathcal{G}$ and a p-closed set U, we denote by $\mathcal{G}[U]$ the game graph induced by the set of vertices U. Note that given that in $\mathcal{G}$ each vertex has at least one outgoing edge, the same property holds for $\mathcal{G}[U]$. We further use the shortcut $\mathcal{G} \setminus X$ to denote $\mathcal{G}[V \setminus X]$.

In a game graph $\mathcal{G}$, a *p-attractor* $Attr_p(\mathcal{G}, U)$ of a set $U \subseteq V$ is the set of vertices from which player p has a strategy to reach U against all strategies of player $\bar{p}$ [62]. We have that $U \subseteq Attr_p(\mathcal{G}, U)$. A p-attractor can be constructed inductively as follows: Let $R_0 = U$; and for all $j \geq 0$ let $R_{j+1} = R_j \cup \{v \in V_p \mid Out(v) \cap R_j \neq \emptyset\} \cup \{v \in V_{\bar{p}} \mid Out(v) \subseteq R_j\}$. Then $Attr_p(\mathcal{G}, U) = \bigcup_{j \geq 0} R_j$. The computation of attractors can be done in linear time [11, 39].

Dominions. A set of vertices $D \subseteq V$ is a player-p *dominion* if $D \neq \emptyset$ and player p has a winning strategy from every vertex in D that also ensures only vertices in D are visited. The notion of dominions was introduced by [42]. Note that a player-p dominion is also a $\bar{p}$-closed set and the p-attractor of a player-p dominion is again a player-p dominion.

▶ **Lemma 1.** *The following assertions hold for game graphs $\mathcal{G}$ where each vertex has at least one outgoing edge. The assertions referring to winning sets hold for graph games with prefix independent objectives. Let $X \subseteq V$.*

1. *The set $V \setminus Attr_p(\mathcal{G}, X)$ is p-closed on $\mathcal{G}$ [62, Lemma 4].*
2. *Let X be p-closed on $\mathcal{G}$. Then $W_{\bar{p}}(\mathcal{G}[X]) \subseteq W_{\bar{p}}(\mathcal{G})$ [42, Lemma 4.4].*
3. *Let X be a subset of the winning set $W_p(\mathcal{G})$ of player p and let A be its p-attractor $Attr_p(\mathcal{G}, X)$. Then the winning set $W_p(\mathcal{G})$ of the player p is the union of A and the winning set $W_p(\mathcal{G}[V \setminus A])$, and the winning set $W_{\bar{p}}(\mathcal{G})$ of the opponent $\bar{p}$ is equal to $W_{\bar{p}}(\mathcal{G}[V \setminus A])$ [42, Lemma 4.5].*

2.2 Conjectured Lower Bounds

While classical complexity results are based on complexity-theoretical assumptions about relationships between complexity classes, e.g., $\mathsf{P} \neq \mathsf{NP}$, polynomial lower bounds are often based on widely believed, conjectured lower bounds about well studied algorithmic problems. We next discuss the popular conjectures that will be the basis for our lower bounds.

First, we consider conjectures on Boolean matrix multiplication [58, 3] and triangle detection [3] in graphs, which build the basis for our lower bounds on dense graphs. A triangle in a graph is a triple x, y, z of vertices such that $(x, y), (y, z), (z, x) \in E$.

▶ **Conjecture 2** (Combinatorial Boolean Matrix Multiplication Conjecture (BMM))**.** *There is no $O(n^{3-\varepsilon})$ time combinatorial algorithm for computing the Boolean product of two $n \times n$ matrices for any $\varepsilon > 0$.*

▶ **Conjecture 3** (Strong Triangle Conjecture (STC))**.** *There is no $O(n^{3-\varepsilon})$ time combinatorial algorithm that can detect whether a graph contains a triangle for any $\varepsilon > 0$.*

BMM is equivalent to STC [58]. A weaker assumption, without the restriction to combinatorial algorithms, is that detecting a triangle in a graph takes super-linear time.

Second, we consider the Strong Exponential Time Hypothesis [40, 20] and the Orthogonal Vectors Conjecture [6], the former dealing with satisfiability in propositional logic and the latter with the *Orthogonal Vectors Problem.*

The Orthogonal Vectors Problem (OV). Given two sets S_1, S_2 of d-bit vectors with $|S_i| \leq N$ and $d \in \Theta(\log N)$, are there $u \in S_1$ and $v \in S_2$ such that $\sum_{i=1}^{d} u_i \cdot v_i = 0$?

▶ **Conjecture 4** (Strong Exponential Time Hypothesis (SETH))**.** *For each $\varepsilon > 0$ there is a k such that k-CNF-SAT on n variables and m clauses cannot be solved in time $O(2^{(1-\varepsilon)n} \operatorname{poly}(m))$.*

▶ **Conjecture 5** (Orthogonal Vectors Conjecture (OVC))**.** *There is no $O(N^{2-\varepsilon})$ time algorithm for the Orthogonal Vectors Problem for any $\varepsilon > 0$.*

SETH implies OVC [59], i.e., whenever a problem is hard assuming OVC, it is also hard when assuming SETH. Hence, it is preferable to use OVC for proving lower bounds. Finally, to the best of our knowledge, no such relations between the former two conjectures and the latter two conjectures are known.

▶ Remark. The conjectures that no *polynomial* improvements over the best known running times are possible do not exclude improvements by sub-polynomial factors such as poly-logarithmic factors or factors of, e.g., $2^{\sqrt{\log n}}$ as in [60].

3 Algorithms for Generalized Büchi Games

For generalized Büchi games we first present the basic algorithm that follows from the results of [33, 51, 62]. The basic algorithm runs in time $O(knm)$. We then improve it to an $O(k \cdot n^2)$-time algorithm by exploiting ideas from the $O(n^2)$-time algorithm for Büchi games in [25]. The basic algorithm is fast for instances where one Büchi set, say T_1, is small, i.e., the algorithm runs in time $O(k \cdot b_1 \cdot m)$ time, where $b_1 = |T_1|$. Generalized Büchi games can also be solved via a reduction to Büchi games [13], which yields an $O(k^2 n^2)$ time algorithm when combined with the $O(n^2)$-time Büchi algorithm [25].

Our algorithms iteratively identify sets of vertices that are winning for player 2, i.e., player-2 dominions, and remove them from the graph. We denote the sets in the jth-iteration with superscript j, in particular $\mathcal{G}^1 = \mathcal{G}$, where $\mathcal{G}$ is the input game graph, G^j is the graph of $\mathcal{G}^j$, V^j is the vertex set of G^j, and $T_\ell^j = V^j \cap T_\ell$. We also use $\{T_\ell^j\}$ to denote the list of Büchi sets $(T_1^j, T_2^j, \ldots, T_k^j)$, in particular when updating all the sets in a uniform way.

Algorithm GenBuchiGame: Algorithm for Generalized Büchi Games

```
Input  : Game graph $\mathcal{G} = ((V, E), (V_1, V_2))$ and objective $\bigwedge_{1 \le \ell \le k}$ Büchi $(T_\ell)$
Output : Winning set of player 1
$\mathcal{G}^1 \leftarrow \mathcal{G}$; $\{T_\ell^1\} \leftarrow \{T_\ell\}$; $j \leftarrow 0$
repeat
    $j \leftarrow j + 1$
    for $i \leftarrow 1$ to $\lceil \log_2 n \rceil$ do
        construct $G_i^j$
        $Z_i^j \leftarrow \{v \in V_2 \mid Outdeg(G_i^j, v) = 0\} \cup \{v \in V_1 \mid Outdeg(G_i^j, v) > 2^i\}$
        for $1 \le \ell \le k$ do
            $Y_{\ell,i}^j \leftarrow Attr_1(G_i^j, T_\ell^j \cup Z_i^j)$
            $S^j \leftarrow V^j \setminus Y_{\ell,i}^j$
            if $S^j \neq \emptyset$ then player 2 dominion found, continue with line 11
    $D^j \leftarrow Attr_2(G^j, S^j)$
    $\mathcal{G}^{j+1} \leftarrow \mathcal{G}^j \setminus D^j$;  $\{T_\ell^{j+1}\} \leftarrow \{T_\ell^j \setminus D^j\}$
until $D^j = \emptyset$
return $V^j$
```

Basic Algorithm. For each set U that is closed for player 1 we have that from each vertex $u \in U$ player 2 has a strategy to ensure that the play never leaves U [62]. Thus, if there is a Büchi set T_ℓ with $T_\ell \cap U = \emptyset$, then the set U is a player-2 dominion. Moreover, if U is a player-2 dominion, also the attractor $Attr_2(\mathcal{G}, U)$ of U is a player-2 dominion. The basic algorithm proceeds as follows. It iteratively computes vertex sets S^j closed for player 1 that do not intersect with one of the Büchi sets. If such a player-2 dominion S^j is found, then all vertices of $Attr_2(\mathcal{G}^j, S^j)$ are marked as winning for player 2 and removed from the game graph; the remaining game graph is denoted by $\mathcal{G}^{j+1}$. To find a player-2 dominion S^j, for each $1 \le \ell \le k$ the attractor $Y_\ell^j = Attr_1(\mathcal{G}^j, T_\ell^j)$ of the Büchi set T_ℓ^j is determined. If for some ℓ the complement of Y_ℓ^j is not empty, then we assign $S^j = V^j \setminus Y_\ell^j$ for the smallest such ℓ. The algorithm terminates if in some iteration j for each $1 \le \ell \le k$ the attractor Y_ℓ^j contains all vertices of V^j. In this case the set V^j is returned as the winning set of player 1. The winning strategy of player 1 from these vertices is then a combination of the attractor strategies to the sets T_ℓ^j.

▶ **Theorem 6.** *The basic algorithm for generalized Büchi games computes the winning set for player 1 in $O(k \cdot \min_{1 \le \ell \le k} b_\ell \cdot m)$ time, where $b_\ell = |T_\ell|$, and thus also in $O(knm)$ time.*

Our Improved Algorithm. The $O(k \cdot n^2)$-time Algorithm GenBuchiGame for generalized Büchi games combines the basic algorithm described above with the method used for the $O(n^2)$-time Büchi game algorithm [26], called *hierarchical graph decomposition* [37]. The hierarchical graph decomposition defines for a directed graph $G = (V, E)$ and integers $1 \le i \le \lceil \log_2 n \rceil$ the graphs $G_i = (V, E_i)$. Assume the incoming edges of each vertex in G are given in some fixed order in which first the edges from vertices of V_2 and then the edges from vertices of V_1 are listed. The set of edges E_i contains all the outgoing edges of each $v \in V$ with $Outdeg(G, v) \le 2^i$ and the first 2^i incoming edges of each vertex. Note that $G = G_{\lceil \log_2 n \rceil}$ and $|E_i| \in O(n \cdot 2^i)$. The runtime analysis uses that we can identify small player-2 dominions (i.e., player-1 closed sets that do not intersect one of the target sets) that contain $O(2^i)$ vertices by only looking at G_i. The algorithm first searches for such a set S^j in G_i for $i = 1$ and each target set and then increases i until the search is successful. In this way the time spent for the search is proportional to $k \cdot n$ times the number of vertices in the found dominion, which yields a total runtime bound of $O(k \cdot n^2)$. To obtain the $O(k \cdot n^2)$ running time bound, it is crucial to put the loop over the different Büchi sets as the innermost part of the algorithm. Given a game graph $\mathcal{G} = (G, (V_1, V_2))$, we denote by $\mathcal{G}_i$ the game graph where G was replaced by G_i from the hierarchical graph decomposition, i.e., $\mathcal{G}_i = (G_i, (V_1, V_2))$.

Properties of hierarchical graph decomposition. The following lemma identifies two essential properties of the hierarchical graph decomposition. The first is crucial for correctness: When searching in $\mathcal{G}_i$ for a player-1 closed set that does not contain one of the target sets, we can ensure that such a set is also closed for player 1 in $\mathcal{G}$ by excluding certain vertices that are missing outgoing edges in $\mathcal{G}_i$ from the search. The second is crucial for the runtime: Whenever the basic algorithm would remove (i.e., identify as winning for player 2) a set with at most 2^i vertices, then we can identify this set also by searching in $\mathcal{G}_i$ instead of $\mathcal{G}$.

▶ **Lemma 7.** *Let $\mathcal{G} = (G = (V, E), (V_1, V_2))$ be a game graph and $\{G_i\}$ its hierarchical graph decomposition. For $1 \le i \le \lceil \log_2 n \rceil$ let Z_i be the set consisting of the player 2 vertices that have no outgoing edge in $\mathcal{G}_i$ and the player 1 vertices with $> 2^i$ outgoing edges in $\mathcal{G}$.*
1. *If a set $S \subseteq V \setminus Z_i$ is closed for player 1 in $\mathcal{G}_i$, then S is closed for player 1 in $\mathcal{G}$.*
2. *If a set $S \subseteq V$ is closed for player 1 in $\mathcal{G}$ and $|Attr_2(\mathcal{G}, S)| \le 2^i$, then (i) $\mathcal{G}_i[S] = \mathcal{G}[S]$, (ii) the set S is in $V \setminus Z_i$, and (iii) S is closed for player 1 in $\mathcal{G}_i$.*

With the above lemma we can show that whenever a player-2 dominion is found in $\mathcal{G}_i$ but not in $\mathcal{G}_{i-1}$, then at least $\Omega(2^i)$ vertices are removed from the maintained game graph. Together with a runtime bound of $O(k \cdot 2^i \cdot n)$ for the search, this yields a total runtime of $O(k \cdot n)$ per vertex, i.e., time $O(k \cdot n^2)$ in total.

▶ **Theorem 8.** *Algorithm* GenBuchiGame *computes the winning set of player 1 in a generalized Büchi game in $O(k \cdot n^2)$ time.*

4 Conditional Lower bounds for Generalized Büchi Games

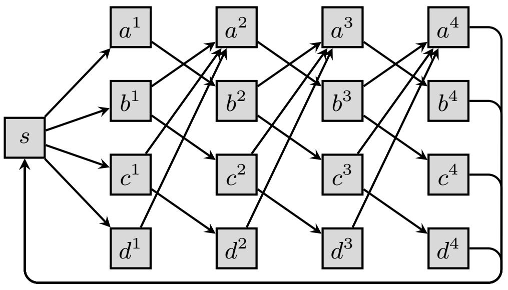
In this section we present two conditional lower bounds, one for dense graphs ($m = \Theta(n^2)$) based on STC & BMM, and one for sparse graphs ($m = \Theta(n^{1+o(1)})$) based on OVC & SETH.

▶ **Theorem 9.** *There is no combinatorial $O(n^{3-\epsilon})$ or $O((k \cdot n^2)^{1-\epsilon})$-time algorithm (for any $\epsilon > 0$) for generalized Büchi games under Conjecture 3 (i.e., unless STC & BMM fail).*

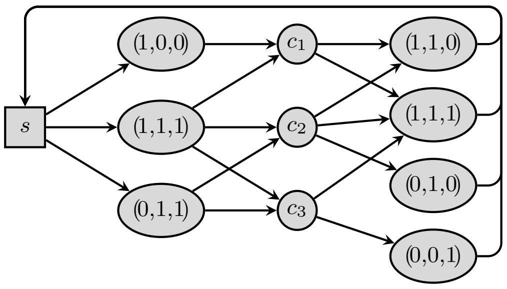
The result can be obtained from a reduction from triangle detection to disjunctive co-Büchi objectives on graphs in [22], and we present the reduction in terms of game graphs below and illustrate it on an example in Figure 1a.

▶ **Reduction 10.** *Given a graph $G = (V, E)$ (for triangle detection), we build a game graph $\mathcal{G}' = (G = (V', E'), (V_1, V_2))$ (for generalized Büchi objectives) as follows. As vertices V' we have four copies V^1, V^2, V^3, V^4 of V and a vertex s. A vertex $v^i \in V^i, i \in \{1, 2, 3\}$ has an edge to a vertex $u^{i+1} \in V^{i+1}$ iff $(v, u) \in E$. Moreover, s has an edge to all vertices of V^1 and all vertices of V^4 have an edge to s. All the vertices are owned by player 2, i.e., $V_1 = \emptyset$ and $V_2 = V$. Finally, we consider the generalized Büchi objective $\bigwedge_{v \in V}$ Büchi(T_v), with $T_v = (V^1 \setminus \{v^1\}) \cup (V^4 \setminus \{v^4\})$.*

We have that there is a triangle in the graph G if and only if the vertex s is winning for player 2 in the generalized Büchi game on $\mathcal{G}'$. Notice that the sets T_v in the above reduction are of linear size but can be reduced to logarithmic size using a construction from [22]. Next we present an $\Omega(m^{2-o(1)})$ lower bound for generalized Büchi objectives.



(a) Reduction 10 applied to $G = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c), (c, a), (c, d), (d, a)\})$.



(b) Reduction 12 applied to $S_1 = \{(1, 0, 0), (1, 1, 1), (0, 1, 1)\}$ and $S_2 = \{(1, 1, 0), (1, 1, 1), (0, 1, 0), (0, 0, 1)\}$.

Figure 1 Illustration of Reductions 10 and 12.

▶ **Theorem 11.** *There is no $O(m^{2-\epsilon})$ or $O(\min_{1\leq i \leq k} b_i \cdot (k \cdot m)^{1-\epsilon})$-time algorithm (for any $\epsilon > 0$) for generalized Büchi games under Conjecture 5 (i.e., unless OVC & SETH fail).*

The above theorem is by a linear time reduction from OV provided below (cf. Figure 1b).

▶ **Reduction 12.** *Given two sets S_1, S_2 of d-dimensional vectors, we build the following game graph. The vertices V of the graph G are given by a start vertex s, vertices S_1 and S_2 representing the sets of vectors, and vertices $\mathcal{C} = \{c_i \mid 1 \leq i \leq d\}$ representing the coordinates. The edges E of G are defined as follows: the start vertex s has an edge to every vertex of S_1 and every vertex of S_2 has an edge to s; further for each $x \in S_1$ there is an edge to $c_i \in \mathcal{C}$ iff $x_i = 1$ and for each $y \in S_2$ there is an edge from $c_i \in \mathcal{C}$ iff $y_i = 1$. The set of vertices V is partitioned into player 1 vertices $V_1 = S_1 \cup S_2 \cup \mathcal{C}$ and player 2 vertices $V_2 = \{s\}$. Finally, the generalized Büchi objective is given by $\bigwedge_{v \in S_2}$ Büchi(T_v) with $T_v = \{v\}$.*

▶ **Lemma 13.** *Given two sets S_1, S_2 of d-dimensional vectors and the corresponding graph game $\mathcal{G}$ given by Reduction 12 with $T_v = \{v\}$ for $v \in S_2$, (1) there exist orthogonal vectors $x \in S_1$ and $y \in S_2$ if and only if (2) $s \notin W_1(\mathcal{G}, \bigwedge_{v \in S_2}$ Büchi$(T_v))$.*

Proof. W.l.o.g. we assume that the 1-vector, i.e., the vector with all coordinates being 1, is contained in S_2 (adding the 1-vector does not change the result of the OV instance), which guarantees that each vertex $c \in \mathcal{C}$ in the construction below has at least 1 outgoing edge. Then a play in the game graph $\mathcal{G}$ proceeds as follows. Starting from s, player 2 chooses a vertex $x \in S_1$; then player 1 first picks a vertex $c \in \mathcal{C}$ and then a vertex $y \in S_2$; then the play goes back to s (at each $y \in S_2$ player 1 has only this choice), starting another cycle of the play. (1)⇒(2): Assume there are orthogonal vectors $x \in S_1$ and $y \in S_2$. Now player 2 can satisfy coBüchi (T_y) by simply going to x whenever the play is in s. Then player 1 can choose some adjacent $c \in \mathcal{C}$ and then some adjacent vertex in S_2, but as x and y are orthogonal, this c is not connected to y. Thus the play will never visit y. (2)⇒(1): By the fact that $W_1 = V \setminus W_2$ [50] we have that (2) is equivalent to $s \in W_2(\mathcal{G}, \bigwedge_{v \in S_2}$ Büchi $(T_v))$. Assume $s \in W_2(\mathcal{G}, \bigwedge_{v \in S_2}$ Büchi $(T_v))$ and consider a corresponding strategy for player 2 that satisfies $\bigvee_{v \in S_2}$ coBüchi (T_v). Notice that the graph is such that player 2 has to visit at least one of the vertices v in S_1 infinitely often. Moreover, for such a vertex v then player 1 can visit all vertices $v' \in S_2$ that correspond to non-orthogonal vectors infinitely often. That is, if v has no orthogonal vector, player 1 can satisfy all the Büchi constraints, a contradiction to our assumption that $s \in W_2(\mathcal{G}, \bigwedge_{v \in S_2}$ Büchi $(T_v))$. Thus there must be a vector $x \in S_1$ such that there exists a vector $y \in S_2$ that is orthogonal to x. ◀

Let $N = \max(|S_1|, |S_2|)$. The number of vertices in the game graph, constructed by Reduction 12, is $O(N)$, the number of edges m is $O(N \log N)$ (recall that $d \in O(\log N)$), we have $k \in \Theta(N)$ target sets, each of size 1, and the construction can be performed in $O(N \log N)$ time. Thus, if we would have an $O(m^{2-\epsilon})$ or $O(\min_{1 \le i \le k} b_i \cdot (k \cdot m)^{1-\epsilon})$ time algorithm for any $\epsilon > 0$, we would immediately get an $O(N^{2-\epsilon})$ algorithm for OV, which contradicts OVC (and thus SETH).

▶ Remark. Notice that the lower bounds apply to instances with $k \in \Theta(n^c)$ for arbitrary $0 < c \le 1$, although the reductions produce graphs with $k \in \Theta(n)$. This is because of the specific type of the constructed instances, for which each $O((k \cdot f(n,m))^{1-\epsilon})$-time algorithm for $k \in \Theta(n^c)$ also implies an $O((k \cdot f(n,m))^{1-\epsilon})$-time algorithm for $k \in \Theta(n)$.

5 Generalized Reactivity-1 Games

GR(1) games deal with an objective of the form $\bigwedge_{t=1}^{k_1} \text{Büchi}(L_t) \to \bigwedge_{\ell=1}^{k_2} \text{Büchi}(U_\ell)$ and can be solved in $O(k_1 k_2 \cdot m \cdot n)$ time [43] with an extension of the progress measure algorithm of [41] and in $O((k_1 k_2 \cdot n)^{2.5})$ time by combining the reduction to one-pair Streett objectives by [13] with the algorithm of [27]. In this section we develop an $O(k_1 k_2 \cdot n^{2.5})$-time algorithm by modifying the algorithm of [43] to compute dominions. We further use our $O(k \cdot n^2)$-time algorithm for generalized Büchi games with $k = k_1$ as a subroutine.

We first describe a basic, direct algorithm for GR(1) games that is based on repeatedly identifying player-2 dominions in generalized Büchi games. We then show how the progress measure algorithm of [43] can be modified to identify player-2 dominions in generalized Büchi games with k_1 Büchi objectives in time proportional to $k_1 \cdot m$ times the size of the dominion. In the $O(k_1 k_2 \cdot n^{2.5})$-time algorithm we use the modified progress measure algorithm in combination with the hierarchical graph decomposition of [26, 27] to identify dominions that contain up to $\sqrt{n}$ vertices and our $O(k_1 \cdot n^2)$-time algorithm for generalized Büchi games to identify dominions with more than $\sqrt{n}$ vertices. Each time we search for a dominion we might have to consider k_2 different subgraphs.

We denote the sets in the jth-iteration of our algorithms with superscript j, in particular $\mathcal{G}^1 = \mathcal{G}$, where $\mathcal{G}$ is the input game graph, G^j is the graph of $\mathcal{G}^j$, V^j is the vertex set of G^j, $V_1^j = V_1 \cap V^j$, $V_2^j = V_2 \cap V^j$, $L_t^j = L_t \cap V^j$, and $U_\ell^j = U_\ell \cap V^j$.

Basic Algorithm. Similar to generalized Büchi games, the basic algorithm for GR(1) games identifies a player-2 dominion S^j, removes the dominion and its player-2 attractor D^j from the graph, and recurses on the remaining game graph $\mathcal{G}^{j+1} = \mathcal{G}^j \setminus D^j$. If no player-2 dominion is found, the remaining set of vertices V^j is returned as the winning set of player 1. Given the set S^j is indeed a player-2 dominion, the correctness of this approach follows from Lemma 1(3). A player-2 dominion in $\mathcal{G}^j$ is identified as follows: For each $1 \le \ell \le k_2$ first the player-1 attractor Y_ℓ^j of U_ℓ^j is temporarily removed from the graph. Then a generalized Büchi game with target sets $L_1^j, \ldots, L_{k_1}^j$ is solved on $\overline{\mathcal{G}^j \setminus Y_\ell^j}$. The generalized Büchi player in this game corresponds to player 2 in the GR(1) game and his winning set to a player-2 dominion in the GR(1) game. Note that $V^j \setminus Y_\ell^j$ is player-1 closed and does not contain U_ℓ^j. Thus if in the game induced by the vertices of $V^j \setminus Y_\ell^j$ player 2 can win w.r.t. the generalized Büchi objective $\bigwedge_{t=1}^{k_1} \text{Büchi}(L_t^j)$, then these vertices form a player-2 dominion in the GR(1) game. Further, we can show that when a player-2 dominion in the GR(1) games on $\mathcal{G}^j$ exists, then for one of the sets U_ℓ^j the winning set of the generalized Büchi game on $\overline{\mathcal{G}^j \setminus Y_\ell^j}$ is non-empty; otherwise we can construct a winning strategy of player 1 for the GR(1) game on $\mathcal{G}^j$. Note that this algorithm computes a player-2 dominion $O(k_2 \cdot n)$ often using our $O(k_1 \cdot n^2)$-time generalized Büchi Algorithm GenBuchiGame.

▶ **Theorem 14.** *The basic algorithm for GR(1) games computes the winning set for player 1 in $O(k_1 \cdot k_2 \cdot n^3)$ time.*

Improved Algorithm. The overall structure of our $O(k_1 k_2 \cdot n^{2.5})$-time algorithm for GR(1) games (see Algorithm GR(1)Game) is the same as for the basic algorithm: We search for a player-2 dominion S^j and if one is found, then its player-2 attractor D^j is determined and removed from the current game graph $\mathcal{G}^j$ (with $\mathcal{G}^1 = \mathcal{G}$) to create the game graph for the next iteration, $\mathcal{G}^{j+1}$. If no player-2 dominion exists, then the remaining vertices are returned as the winning set of player 1. The difference to the basic algorithm lies in the way player-2 dominions are searched. Two different procedures are used for this purpose: First we search for "small" dominions with the subroutine `kGenBüchiDominion`. If no small dominions exist, then we search for player-2 dominions as in the basic algorithm. The guarantee that we find a "large" dominion allows us to bound the number of times the second case can happen.

Progress Measure Algorithm. In the Procedure `kGenBüchiDominion` we use a subroutine that finds in a generalized Büchi game all dominions of the generalized Büchi player that have size at most h (where h is an input parameter). This subroutine is based on a so-called *progress measure* for generalized Büchi objectives which is a special instance of the progress measure for GR(1) objectives presented in [43, Section 3.1], which itself is based on [41]. We modify the progress measure to efficiently identify dominions of size at most h (instead of computing the whole winning set) by restricting the range of allowed values for the progress measure functions similar to [56]. Finally, we give an $O(k \cdot h \cdot m)$-time algorithm for computing the progress measure functions based on [34, 43] (details are provided in the full version).

▶ **Theorem 15.** *For a game graph $\mathcal{G}$ and objective $\psi = \bigwedge_{1 \le \ell \le k} \text{Büchi}(T_\ell)$, there is an $O(k \cdot h \cdot m)$ time procedure* `GenBüchiProgressMeasure`$(\mathcal{G}, \psi, h)$ *that either returns a player-1 dominion or the empty set, and, if there is at least one player-1 dominion of size $\le h$ then returns a player-1 dominion containing all player-1 dominions of size $\le h$.*

Procedure `kGenBüchiDominion`. The procedure `kGenBüchiDominion` searches for player-2 dominions in the GR(1) game, and returns some dominion if there exists a dominion D with $|Attr_2(\mathcal{G}, D)| \le h_{\max}$. To this end we again consider generalized Büchi games with target sets $L_1^j, \dots, L_{k_1}^j$, where the generalized Büchi player corresponds to player 2 in the GR(1) game. We use the same hierarchical graph decomposition as for Algorithm GenBuchiGame: Let the incoming edges of each vertex be ordered such that the edges from vertices of V_2 come first; for a given game graph $\mathcal{G}^j$ the graph G_i^j contains all vertices of $\mathcal{G}^j$, for each vertex its first 2^i incoming edges, and for each vertex with outdegree at most 2^i all its outgoing edges. The set Z_i^j contains all vertices of V_1 with outdegree larger than 2^i and all vertices of V_2 that have no outgoing edge in G_i^j. We start with $i = 1$ and increase i by one as long as no dominion was found. For a given i we perform the following operations for each $1 \le \ell \le k_2$: First the player 1 attractor $Y_{i,\ell}^j$ of $U_\ell^j \cup Z_i^j$ is determined. Then we search for player-1 dominions on $\overline{\mathcal{G}_i^j \setminus Y_{i,\ell}^j}$ w.r.t. the objective $\bigwedge_{t=1}^{k_1} \text{Büchi}(L_t)$ with the generalized Büchi progress measure algorithm and parameter $h = 2^i$, i.e., by Theorem 15 the progress measure algorithm returns all generalized Büchi dominions in $\overline{\mathcal{G}_i^j \setminus Y_{i,\ell}^j}$ of size at most h.

The following lemma shows how the properties of the hierarchical graph decomposition extend to GR(1) games. The first part is crucial for correctness: Every non-empty set found by the progress measure algorithm on $\overline{\mathcal{G}_i^j \setminus Y_{i,\ell}^j}$ for some i and ℓ is indeed a player-2 dominion in the GR(1) game. The second part is crucial for the runtime argument: Whenever the basic algorithm for GR(1) games would identify a player-2 dominion D with $|Attr_2(\mathcal{G}, D)| \le 2^i$, then D is also a generalized Büchi dominion in $\overline{\mathcal{G}_i^j \setminus Y_{i,\ell}^j}$ for some ℓ.

Algorithm GR(1)GAME: Algorithm for GR(1) Games

```
Input  : Game graph $\mathcal{G} = ((V, E), (V_1, V_2))$, Obj. $\bigwedge_{t=1}^{k_1} \text{Büchi}(L_t) \rightarrow \bigwedge_{\ell=1}^{k_2} \text{Büchi}(U_\ell)$
Output : Winning set of player 1
$\mathcal{G}^1 \leftarrow \mathcal{G}$; $\{U_\ell^1\} \leftarrow \{U_\ell\}$; $\{L_t^1\} \leftarrow \{L_t\}$
$j \leftarrow 0$
repeat
    $j \leftarrow j + 1$
    $S^j \leftarrow$ kGenBüchiDominion($\mathcal{G}^j, \{U_\ell^j\}, \{L_t^j\}, \sqrt{n}$)
    if $S^j = \emptyset$ then
        for $1 \le \ell \le k_2$ do
            $Y_\ell^j \leftarrow Attr_1(\mathcal{G}^j, U_\ell^j)$
            $S^j \leftarrow$ GenBüchiGame($\overline{\mathcal{G}^j \setminus Y_\ell^j}, \bigwedge_{\ell=1}^{k_1} \text{Büchi}\left(L_t^j \setminus Y_\ell^j\right)$)
            if $S^j \neq \emptyset$ then break
    $D^j \leftarrow Attr_2(\mathcal{G}^j, S^j)$
    $\mathcal{G}^{j+1} \leftarrow \mathcal{G}^j \setminus D^j$; $\{U_\ell^{j+1}\} \leftarrow \{U_\ell^j \setminus D^j\}$; $\{L_t^{j+1}\} \leftarrow \{L_t^j \setminus D^j\}$
until $D^j = \emptyset$
return $V^j$

Procedure kGenBüchiDominion($\mathcal{G}^j, \{U_\ell^j\}, \{L_t^j\}, h_{max}$)
    for $i \leftarrow 1$ to $\lceil \log_2(h_{max}) \rceil$ do
        construct $G_i^j$
        $Z_i^j \leftarrow \{v \in V_2 \mid Outdeg(G_i^j, v) = 0\} \cup \{v \in V_1 \mid Outdeg(G_i^j, v) > 2^i\}$
        for $1 \le \ell \le k_2$ do
            $Y_{i,\ell}^j \leftarrow Attr_1(\mathcal{G}_i^j, U_\ell^j \cup Z_i^j)$
            $X_{i,\ell}^j \leftarrow$ GenBüchiProgressMeasure($\overline{\mathcal{G}_i^j \setminus Y_{i,\ell}^j}, \bigwedge_{\ell=1}^{k_1} \text{Büchi}\left(L_t^j \setminus Y_{i,\ell}^j\right), 2^i$)
            if $X_{i,\ell}^j \neq \emptyset$ then return $X_{i,\ell}^j$
    return $\emptyset$
```

▶ **Lemma 16.** *Let the notation be as in Algorithm* GR(1)GAME*.*

1. *Every $X_{i,\ell}^j \neq \emptyset$ is a player-2 dominion in the GR(1) game on $\mathcal{G}^j$ with $X_{i,\ell}^j \cap U_\ell^j = \emptyset$.*
2. *If for player 2 there exists in $\mathcal{G}^j$ a dominion D w.r.t. the generalized Büchi objective $\bigwedge_{t=1}^{k_1}$ Büchi(L_t^j) such that $D \cap U_\ell^j = \emptyset$ for some $1 \le \ell \le k_2$ and $|Attr_2(\mathcal{G}^j, D)| \le 2^i$, then D is a dominion w.r.t. the generalized Büchi objective $\bigwedge_{t=1}^{k_1}$ Büchi$(L_t^j \setminus Y_{i,\ell}^j)$ in $\mathcal{G}_i^j \setminus Y_{i,\ell}^j$.*

From this we can draw the following two corollaries: (1) When we had to go up to i^* in the graph decomposition to find a dominion, then its attractor has size at least 2^{i^*-1} and (2) when `kGenBüchiDominion` returns an empty set, then all player-2 dominions in the current game graph have more than $h_{\max} = \sqrt{n}$ vertices. In the second case either no player-2 dominion exists or the subsequent call to `GenBüchiGame` returns one with more than $\sqrt{n}$ vertices, which can happen at most $O(\sqrt{n})$ times. Together with (1), this means we can (a) charge the time spent in `kGenBüchiDominion` to the vertices in the dominion identified in this iteration of the repeat-until loop (except for the last iteration) and (b) bound the number of calls to `GenBüchiGame` with $O(\sqrt{n})$.

▶ **Theorem 17.** *Algorithm* GR(1)GAME *computes the winning set of player 1 in a GR(1) game in $O(k_1 \cdot k_2 \cdot n^{2.5})$ time.*

6 Conclusion

In this work we present improved algorithms for generalized Büchi and GR(1) objectives, and conditional lower bounds for generalized Büchi objectives. The existing upper bounds and our conditional lower bounds are tight for (a) for dense graphs, and (b) sparse graphs with constant size target sets. Two interesting open questions are as follows: (1) For sparse graphs with $\theta(n)$ many target sets of size $\theta(n)$ the upper bounds are cubic, whereas the conditional lower bound is quadratic, and closing the gap is an interesting open question. (2) For GR(1) objectives we obtain the conditional lower bounds from generalized Büchi objectives, which are not tight in this case; whether better (conditional) lower bounds can be established also remains open.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *FOCS*, pages 98–117, 2015.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results For LCS and other Sequence Similarity Measures. In *FOCS*, pages 59–78, 2015.

3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443, 2014.

4 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *ICALP 2014, Proceedings, Part I*, pages 39–51, 2014.

5 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *STOC*, pages 41–50, 2015.

6 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *SODA*, pages 377–391, 2016.

7 Rajeev Alur and Thomas A. Henzinger. Computer-aided verification, 2004. Unpublished, available at `http://www.cis.upenn.edu/group/cis673/`.

8 Rajeev Alur, Thomas. A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.

9 Rajeev Alur and Salvatore La Torre. Deterministic generators and games for ltl fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

10 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *STOC*, pages 51–58, 2015.

11 Catriel Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Transactions on Database Systems*, pages 241–259, 1980.

12 Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. Dag-width and parity games. In *STACS*, pages 524–536, 2006.

13 Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. Robustness in the presence of liveness. In *CAV*, pages 410–424, 2010.

14 Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Interactive presentation: Automatic hardware synthesis from specifications: a case study. In *DATE*, pages 1188–1193, 2007.

15 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *FOCS*, pages 661–670, 2014.

16 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *FOCS*, pages 79–97, 2015.

17 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

18 J. Richard Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Proceedings of the First International Congress on Logic, Methodology, and Philosophy of Science 1960*, pages 1–11. Stanford University Press, 1962.

19 J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.

20 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *IWPEC*, pages 75–85, 2009.

21 Krishnendu Chatterjee, Martin Chmelik, Raghav Gupta, and Ayush Kanodia. Qualitative analysis of pomdps with temporal logic specifications for robotics applications. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 325–330, 2015.

22 Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Veronika Loitzenbauer. Model and objective separation with conditional lower bounds: Disjunction is harder than conjunction. In *LICS*, 2016. To appear, available at `http://arxiv.org/abs/1602.02670`.

23 Krishnendu Chatterjee, Amir Kafshdar Goharshady, Rasmus Ibsen-Jensen, and Andreas Pavlogiannis. Algorithms for algebraic path properties in concurrent systems of constant treewidth components. In *POPL*, pages 733–747, 2016.

24 Krishnendu Chatterjee and Monika Henzinger. Faster and Dynamic Algorithms For Maximal End-Component Decomposition And Related Graph Problems In Probabilistic Verification. In *SODA*, pages 1318–1336, 2011.

25 Krishnendu Chatterjee and Monika Henzinger. An $O(n^2)$ Time Algorithm for Alternating Büchi Games. In *SODA*, pages 1386–1399, 2012.

26 Krishnendu Chatterjee and Monika Henzinger. Efficient and Dynamic Algorithms for Alternating Büchi Games and Maximal End-component Decomposition. *Journal of the ACM*, 61(3):15, 2014.

27 Krishnendu Chatterjee, Monika Henzinger, and Veronika Loitzenbauer. Improved Algorithms for One-Pair and k-Pair Streett Objectives. In *LICS*, pages 269–280, 2015.

28 Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Andreas Pavlogiannis, and Prateesh Goyal. Faster algorithms for algebraic path properties in recursive state machines with constant treewidth. In *POPL*. ACM, 2015.

29 Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger. Simple stochastic parity games. In *CSL*, pages 100–113, 2003.

30 Alonzo Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35. Institut Mittag-Leffler, 1962.

31 Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *FSE'01*, pages 109–120. ACM Press, 2001.

32 David L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits*. The MIT Press, 1989.

33 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS*, pages 368–377, 1991.

34 Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair simulation relations, parity games, and state space reduction for büchi automata. *SIAM J. Comput.*, 34(5):1159–1175, 2005.

35 Georgios E. Fainekos, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for mobile robots. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 2020–2025, 2005.

36 Yashdeep Godhal, Krishnendu Chatterjee, and Thomas A. Henzinger. Synthesis of AMBA AHB from formal specification: A case study. *Journal of Software Tools Technology Transfer*, 2011.

37 Monika Henzinger, Valerie King, and Tandy Warnow. Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Evolutionary Biology. *Algorithmica*, 24(1):1–13, 1999.

38 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30, 2015.

39 Neil Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and System Sciences*, pages 384–406, 1981.

40 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

41 Marcin Jurdziński. Small Progress Measures for Solving Parity Games. In *STACS*, pages 290–301, 2000.

42 Marcin Jurdziński, Mike Paterson, and Uri Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.

43 Sudeep Juvekar and Nir Piterman. Minimizing generalized büchi automata. In *CAV*, pages 45–58, 2006.

44 Sriram C. Krishnan, Anuj Puri, and Robert K. Brayton. Deterministic Ω automata vis-a-vis deterministic buchi automata. In *ISAAC*, pages 378–386, 1994.

45 Wouter Kuijper and Jaco van de Pol. Computing weakest strategies for safety games of imperfect information. In *TACAS*, pages 92–106, 2009.

46 Orna Kupferman and Moshe Y. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *LICS*, pages 81–92, 1998.

47 Orna Kupferman and Moshe Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

48 François Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *ISSAC*, pages 296–303, 2014.

49 Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, January 2002.

50 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

51 Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

52 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *SODA*, pages 1065–1075, 2010.

53 Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, LNCS 3855, Springer, pages 364–380, 2006.

54 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM Press, 1989.

55 P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

56 Sven Schewe. Solving Parity Games in Big Steps. In *FSTTCS*, pages 449–460, 2007.

57 Mikkel Thorup. All Structured Programs Have Small Tree Width and Good Register Allocation. *Information and Computation*, 142(2):159 – 181, 1998.

58 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654, 2010.

59 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. Announced at ICALP'04.

60 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *STOC*, pages 664–673, 2014.

61 Ryan Williams. Faster decision of first-order graph properties. In *CSL-LICS*, pages 80:1–80:6, 2014.

62 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998.

FPT Algorithms for Plane Completion Problems*

Dimitris Chatzidimitriou[1], Archontia C. Giannopoulou[2], Spyridon Maniatis[3], Clément Requilé[4], Dimitrios M. Thilikos[5], and Dimitris Zoros[6]

1 **Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece**
`hatzisdimitris@gmail.com`

2 **Institute of Software Technology and Theoretical Computer Science, Technische Universität Berlin, Germany**
`archontia.giannopoulou@tu-berlin.de`

3 **Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece**
`spyridon.maniatis@gmail.com`

4 **Freie Universität Berlin, Institut für Mathematik und Informatik, Berlin, Germany and**
Department of Mathematics, Universitat Politècnica de Catalunya, Barcelona, Spain
`clement.requile@gmail.com`

5 **Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece and**
AlGCo project team, CNRS, LIRMM, France.
`sedthilk@thilikos.info`

6 **Department of Mathematics, National and Kapodistrian University of Athens, Athens, Greece**
`dzoros@math.uoa.gr`

Abstract

The Plane Subgraph (resp. Topological Minor) Completion problem asks, given a (possibly disconnected) plane (multi)graph Γ and a connected plane (multi)graph Δ, whether it is possible to add edges in Γ without violating the planarity of its embedding so that it contains some subgraph (resp. topological minor) that is topologically isomorphic to Δ. We give FPT algorithms that solve both problems in $f(|E(\Delta)|) \cdot |E(\Gamma)|^2$ steps. Moreover, for the Plane Subgraph Completion problem we show that $f(k) = 2^{\mathcal{O}(k \log k)}$.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases Completion problems, FPT, Plane graphs, Topological Isomorphism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.26

* The first, the third, the fifth, and the last authors were supported by the E.U. (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: "Thales. Investing in knowledge society through the European Social Fund". The second author's research has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No 648527) and by the Warsaw Center of Mathematics and Computer Science. The fourth author was supported by the FP7-PEOPLE-2013-CIG project CountGraph (ref. 630749), and the Berlin Mathematical School.

1 Introduction

Completion problems on graphs are defined as follows: Consider a graph class $\mathcal{P}$ and ask whether we may add edges to a given graph G in order to obtain a graph G^+, where $G^+ \in \mathcal{P}$. Numerous results have appeared for the case where the objective is to minimize the number of edges added in G [13, 9, 11, 8, 3].

In this paper, we consider the Plane Subgraph (resp. Topological Minor) Completion (PSC) (resp. PTMC) problem which, given a (possibly disconnected) plane graph Γ, called the *host graph*, and a connected plane graph Δ, called the *pattern graph*, asks whether it is possible to add edges in Γ such that the resulting graph remains plane and contains some subgraph (resp. topological minor) that is topologically isomorphic to Δ. Both Γ and Δ are allowed to have multiple edges but not loops. When the input graph Γ is planar triangulated, both PSC and PTMC are NP-complete. Indeed, let G be any planar triangulated graph. Note here, that as any planar triangulated graph is 3-connected, G is 3-connected and from Whitney's Theorem [12] admits a unique embedding on the plane (up to equivalence), say Γ. Let also Δ be the cycle on $n = |V(G)|$ vertices. Then Δ also has unique embedding on the plane (up to equivalence). Since Γ is triangulated no edge can be added to it while preserving its planarity. Thus, both PSC and PTMC become equivalent to the Hamilton Cycle Problem which is NP-complete on planar triangulated graphs [4] (see also [7]). This observation further implies that PSC and PTMC parameterized by the number of added edges k, and in particular even for $k = 0$, are NP-complete. Thus, PSC and PTMC are not FPT when parameterized by the number of added edges unless P $=$ NP. Thus, in order to obtain a tractable algorithm, we need to find an alternative way to parameterize these problems. In particular, we will consider $|E(\Delta)|$ as our parameter. Our two main results are the following.

▶ **Theorem.** PSC *parameterized by the number of edges of the pattern graph Δ, say k, can be solved in $2^{\mathcal{O}(k \log k)} \cdot m^2$ time, where $m = |E(\Gamma)|$.*

▶ **Theorem.** PTMC *parameterized by the number of edges of the pattern graph Δ, say k, can be solved in $f(k) \cdot m^2$ time, where $m = |E(\Gamma)|$ and f is a computable function.*

For the PTMC algorithm our approach is the following. Let Γ and Δ be an input of the problem as above. We first apply a series of transformations on our input graph Γ that turn it into a combinatorial structure $\mathbf{G}$ whose treewidth is bounded by a function of $|E(\Delta)|$. Then, we apply a series of transformations on our input graph Δ that also turn it into a combinatorial structure $\mathbf{D}$. Finally, we show that (Δ, Γ) is a yes-instance of our problem if and only if an MSO-expressible relation holds for $\mathbf{G}$ and $\mathbf{D}$, thus translating our problem into a purely combinatorial one. Then by employing Courcelle's Theorem we prove our algorithm. We remark here that a similar approach could also solve the Plane Subgraph Completion problem. However, with a more careful analysis we are able to derive an algorithm with much better bounds on the dependence on the parameter.

Our approach towards solving PSC is the following. Let Γ and Δ be an input of PSC, where $|E(\Delta)| = k$ for some positive integer k. We construct a family $\mathcal{G}$ consisting of $\mathcal{O}(n)$ combinatorial structures depending only on Γ whose underlying graphs have treewidth $\mathcal{O}(k)$. We also construct a family $\mathcal{H}$ consisting of $2^{\mathcal{O}(k \log k)}$ combinatorial structures depending only on Δ, again by applying series of appropriate transformations on them (different than the transformations for PTMC). For the graphs Γ and Δ and the families $\mathcal{G}$ and $\mathcal{H}$, it holds that (Δ, Γ) is a yes-instance if and only if some structure $\mathbf{D} \in \mathcal{H}$ is contained as a *contraction* in a structure $\mathbf{G} \in \mathcal{G}$, denoted $\mathbf{D} \leq_c \mathbf{G}$. Therefore, we again translate our problem into one of combinatorial nature. Finally, for a fixed pair of structures $(\mathbf{D}, \mathbf{G}) \in \mathcal{H} \times \mathcal{G}$ with the above properties, we can decide in $2^{\mathcal{O}(k \log k)} \cdot m$ time whether $\mathbf{D} \leq_c \mathbf{G}$. Therefore, by testing for all pairs $(\mathbf{D}, \mathbf{G}) \in \mathcal{H} \times \mathcal{G}$ whether $\mathbf{D} \leq_c \mathbf{G}$, we decide in $2^{\mathcal{O}(k \log k)} \cdot m^2$ steps whether (Δ, Γ) is a yes-instance.

The paper is organized as follows. In Section 2 we give the necessary definitions. In Section 3 we present the algorithm for the PSC problem and in Section 4 we present the algorithm for the PTMC problem. In the concluding Section 5 we discuss about other completion problems that can be solved by modifying our results, such as the Plane Induced Subgraph Completion, the Plane Minor Completion, the Planar Rooted Topological Minor, and the Planar Disjoint Paths Completion problems. The lemmas whose proofs have been omitted due to lack of space are marked with $(\star)$.

2 Definitions

For a positive integer n, we denote $[n] = \{1, 2, \ldots, n\}$. Given a set S, a near-partition of S is a family of sets S_1, S_2, … , S_k, where $S_i \cap S_j = \emptyset$, for every $i \neq j$, and $\cup_{i \in [k]} S_i = S$ (note that by the definition it is possible that $S_i = \emptyset$ for some $i \in [k]$). Unless stated otherwise, the graphs considered do not have loops but may have multiple edges. In a graph G we will denote by $V(G)$ the set of its vertices and $E(G)$ the set of its edges. We denote by $\mathbf{dist}_G(u, v)$ the distance of two vertices u and v in the graph G. Also, given a graph G, a vertex $u \in V(G)$, and $V_0 \subseteq V(G)$, we denote by $N_G(u)$ the neighborhood of u in G and by $N_G(V_0) := \bigcup_{v \in V_0} N_G(v) \setminus V_0$. Given a vertex v with exactly two neighbors v_1 and v_2, the dissolution of v is the operation where we delete v and add an edge $\{v_1, v_2\}$ (even if one existed already).

Let G be a graph. A subset S of its vertices is a *separator* of G if the graph $G - S := (V(G) \setminus S, E[V(G) \setminus S])$ is not connected. The size of a separator S is equal to $|S|$. The vertex contained in a separator of size 1 will be called a *cut-vertex*, while the vertices of a separator of size 2 will be called a *cut-pair*. For every integer $k > 1$, a graph G with at least $k+1$ vertices is *k-connected* if G has no separators of size less than k. For definitions not explicitly stated on the paper as well as more details on general graphs, see [6]

We say that a graph is *plane* when it is embedded without crossings between its edges on the sphere $\Sigma = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 = 1\}$. We treat a plane graph as its embedding in Σ and we simply refer to it as *plane graph*. That is, we do not distinguish between a vertex of the graph and the point of the sphere used in the drawing to represent the vertex or between an edge and the open line segment representing it. We often use the term "general graph" in order to stress that a graph is treated as a combinatorial structure and not as a topological (i.e., embedded) one. Also, given a plane graph Γ we use the term *general graph* of Γ to refer to Γ as a combinatorial structure. We use capital greek letters for plane graphs and capital latin letters for general graphs.

We denote by $\subseteq$, $\subseteq_{sp}$, $\subseteq_{in}$, $\leq_m$, and $\simeq$ the usual subgraph, spanning subgraph, induced subgraph, minor, and isomorphism relation between two graphs, respectively. Given a graph G and $V_0 \subseteq V(G)$, we denote by $G[V_0]$ the subgraph of G induced by V_0. We call V_0 *connected* if $G[V_0]$ is connected.

Let Γ be a plane graph and $u \in V(\Gamma)$. Then a tuple $(u_1, \ldots, u_k)$, with possible repetitions, will be called a *cyclic neighborhood* of u, and denoted by $\mathcal{N}_\Gamma(u)$, if $\{u, u_1\}, \ldots, \{u, u_k\}$ are exactly the edges incident to u, as we meet them starting from (u, u_1) and proceeding clockwise.

Let A be a subset of $\mathbb{R}^n$. We define $\mathbf{int}(A)$ to be the interior of A, $\mathbf{cl}(A)$ its closure and $\mathbf{bd}(A) = \mathbf{cl}(A) \setminus \mathbf{int}(A)$ its border. Given a plane graph Γ we denote its *faces* by $F(\Gamma)$, i.e.,

$F(\Gamma)$ is the set of the connected components of $\Sigma \setminus \Gamma$ (in the operation $\Sigma \setminus \Gamma$ we treat Γ as the set of points of Σ corresponding to its vertices and its edges). Given a graph G we denote by $\mathcal{C}(G)$ the set of the connected components of G. For every $f \in F(\Gamma)$ we denote by $B_\Gamma(f)$ the graph induced by the vertices and edges of Γ whose embeddings are subsets of $\mathbf{bd}(f)$ and we call it the *boundary* of f. We also denote by $V(B_\Gamma(f))$ and $E(B_\Gamma(f))$ the vertices and the edges of $B_\Gamma(f)$, respectively.

We define a *closed walk* of a graph G to be a cyclic ordering $w = (v_1, \dots, v_l, v_1)$ of vertices of $V(G)$ such that for any two consecutive vertices, say v_i, v_{i+1}, there is an edge between them in G, i.e., $\{v_i, v_{i+1}\} \in E(G)$. Note here that there may exist two distinct indices i, j such that $v_i, v_j \in w$ and $v_i = v_j$ (the walk can revisit a vertex). We will denote by $\ell_w = l$ the length of the respective closed walk w. We say that a walk w of a plane graph Γ is *facial* if there exists $f_i \in F(\Gamma)$ and $\Theta_j \in \mathcal{C}(B_\Gamma(f_i))$ such that the vertices of w are the vertices of $V(\Theta_j)$ and the cyclic ordering of w indicates the way these vertices are met when making a closed walk along Θ_j while always keeping f_i on the same side of the walk.

Given that Γ is a plane graph and $\mathbf{w} = \{w_1, \dots, w_p\}$ is a non-empty set of closed walks of Γ, we say that $\mathbf{w}$ is a *facial mapping* if there exists some face f of Γ such that $\mathcal{C}(B_\Gamma(f)) = \{\Theta_1, \Theta_2, \dots, \Theta_p\}$ and w_j is a facial walk of Θ_j, $j \in [p]$. We define the length of the facial mapping $\mathbf{w}$ to be $\ell_{\mathbf{w}} = \sum_{i=1}^{p} \ell_{w_i}$. Given a plane graph Γ and $f \in F(\Gamma)$, we define $\mathbf{w}(f)$ as the facial mapping of Γ corresponding to f and define its length ℓ_f to be the length $\ell_{\mathbf{w}}(f)$ of its corresponding facial mapping. Observe that for every face $f \in \Gamma(F)$, its facial mapping $\mathbf{w}(f)$ is unique (up to permutations). Let C_1, C_2 be two disjoint closed curves of Σ. Let also D_i be the open disk of $\Sigma \setminus C_i$ that does not contain points of C_{3-i}, $i \in [2]$. The *annulus between* C_1 and C_2 is the set $\Sigma \setminus (D_1 \cup D_2)$ and we denote it by $A[C_1, C_2]$. Notice that $A[C_1, C_2]$ is a closed set.

Let Γ and Δ be two plane graphs. We say that Γ and Δ are *topologically isomorphic* if they are isomorphic via a bijection $g : V(\Gamma) \to V(\Delta)$ and there exists a function $h : F(\Gamma) \to F(\Delta)$, such that for every $f \in F(\Gamma)$, $g(\mathbf{w}(f)) = \mathbf{w}(h(f))$ (where $g(\mathbf{w}(f))$ is the result of applying g to every vertex of every closed walk in $\mathbf{w}$). We call the function $\alpha : V(\Gamma) \cup F(\Gamma) \to V(\Delta) \cup F(\Delta)$ such that $\alpha = g \cup h$, a *topological isomorphism* between Γ and Δ.

We say that a general graph G is *uniquely embeddable* if any two plane graphs Γ and Γ' that are embeddings of G in the sphere, are topologically isomorphic. We say that a plane graph Γ is *uniquely embedded* if its general graph G is uniquely embeddable, i.e., Γ is the unique embedding of G, up to topological isomorphism. Given two plane graphs Γ_1 and Γ_2 we say that they are the *same* graph if they are topologically isomorphic (and not just isomorphic).

Let Γ and Δ be two plane graphs and let $Z \subseteq V(\Gamma)$. We say that Δ is a *Z-embedded subgraph* of Γ, and write $\Delta \leq_{es}^{Z} \Gamma$, if Δ is topologically isomorphic to some subgraph of $\Gamma \setminus Z$. When $Z = \emptyset$, we say that Δ is an *embedded subgraph* of Γ and write $\Delta \leq_{es} \Gamma$.

Let Γ and Δ be two plane graphs and let $Z \subseteq V(\Gamma)$. We say that Δ is a *Z-embedded topological minor* of Γ, and write $\Delta \leq_{etm}^{Z} \Gamma$ if there exist a function $\rho_1 : V(\Delta) \to V(\Gamma)$ and a function $\rho_2 : E(\Delta) \to \mathcal{P}(\Gamma)$, where $\mathcal{P}(\Gamma)$ denotes the set of all paths of Γ such that

1. For every $v \in V(\Delta)$, $\rho_1(v) \notin Z$.
2. For every $e = \{u, v\} \in E(\Delta)$, the path $\rho_2(e)$ of Γ has $\rho_1(u)$ and $\rho_1(v)$ as its endpoints and if $e_1 \neq e_2$, then $\rho_2(e_1)$ and $\rho_2(e_2)$ are internally vertex-disjoint.
3. If $\Gamma\langle\rho_2\rangle$ is the graph obtained by the union of all paths in $\rho_2(E(\Delta))$ after we dissolve all vertices that are not vertices in $\rho_1(V(\Delta))$, then there is a topological isomorphism $\alpha : V(\Delta) \cup F(\Delta) \to V(\Gamma\langle\rho_2\rangle) \cup F(\Gamma\langle\rho_2\rangle)$ between Δ and $\Gamma\langle\rho_2\rangle$ where $\alpha|_{V(\Delta)} = \rho_1$.

When $Z = \emptyset$, we just write $\Delta \leq_{etm} \Gamma$.

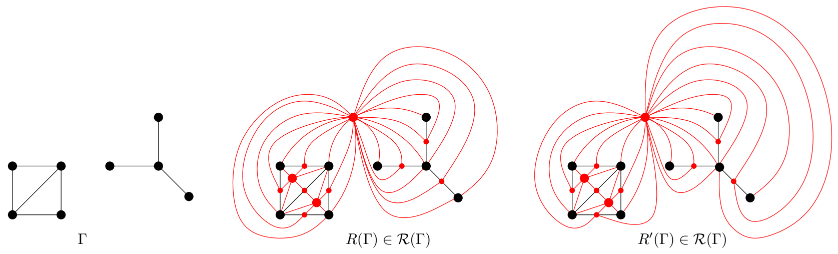

Γ
$R(\Gamma) \in \mathcal{R}(\Gamma)$
$R'(\Gamma) \in \mathcal{R}(\Gamma)$

Figure 1 A disconnected plane graph Γ and two members of $\mathcal{R}_\Gamma$.

If in the 3rd condition of the above definition we replace topological isomorphism by isomorphism and consider general graphs, say H and G, we define the relation of H being a *Z-topological minor* of (G, Z).

For definitions not explicitly stated on the paper as well as more details on plane graphs, see [10].

2.1 Radial Enhancements

Let Γ be a plane graph. A *subdivided radial enhancement* of Γ is defined as a plane graph that can be constructed as follows: consider Γ, subdivide every edge once, add a vertex v_f inside each face f of Γ. Consider a permutation $(H_1, H_2, \ldots, H_s)$ of the connected components of $B_\Gamma(f)$ and a facial walk of each connected component. Then add edges connecting v_f with the vertices incident to $B_\Gamma(f)$ in such a way that the first vertices in the cyclic neighborhood of v_f are the vertices of H_1 and appear in the order of the fixed facial walk. Then the vertices of H_2 follow, etc. Observe that in the resulting embedding, every face that is incident to an edge of $E(\Gamma)$ is (planar) triangulated. This triangulation may have multiple edges unless the boundary of each face of Γ is a cycle, as can be seen in the two distinct examples of a subdivided radial enhancement of a disconnected plane graph Γ in Figure 1.

Notice that the vertices of the resulting plane graph can be partitioned into three independent sets: the *original vertices* of Γ denoted by $V_o(\Gamma)$, the *subdivision vertices* denoted by $V_s(\Gamma)$, which are the ones that were introduced after subdividing the edges, and the *radial vertices* denoted by $V_r(\Gamma)$, which are the ones that were added inside each face. Notice also that the edges of the resulting plane graph can be partitioned into two independent sets: the *subdivision edges* denoted by $E_s(\Gamma)$ and the *radial edges*, denoted by $E_r(\Gamma)$, that were introduced after adding the radial vertices.

We denote by $\mathcal{R}_\Gamma$ the set of all different (in terms of topological isomorphism) subdivided radial enhancements of Γ. Observe that if Γ is connected, then the boundary of each face of Γ is connected and we obtain the following.

▶ **Observation 1.** *For every connected plane graph Γ, $R(\Gamma)$ is uniquely defined and thus $\mathcal{R}_\Gamma$ contains only one member.*

From the subdivided radial enhancement's construction we obtain the following.

▶ **Observation 2.** *For every plane graph Γ and every $R(\Gamma) \in \mathcal{R}_\Gamma$ it holds that $|E(R(\Gamma)| = \mathcal{O}(|E(\Gamma)|)$.*

Given a plane graph Γ and a graph $R(\Gamma) \in \mathcal{R}_\Gamma$, for every integer $i > 1$, we denote by $R^i(\Gamma)$ the graph $R(R^{i-1}(\Gamma))$, where $R^1(\Gamma) = R(\Gamma)$. We define then $V_o^i(\Gamma) = V(R^{i-1}(\Gamma))$, $V_s^i = V_s(R^{i-1}(\Gamma))$, and $V_r^i = V_r(R^{i-1}(\Gamma))$. For notation consistency, we will denote by

$V_s^1(\Gamma) = V_s(\Gamma)$, $V_r^1(\Gamma) = V_r(\Gamma)$ and $V_o^1(\Gamma) = V_o(\Gamma)$. We also define the sets of edges $E_s^i(\Gamma)$ which are the edges obtained in $R^i(\Gamma)$ after subdiving the edges $E_s^{i-1}(\Gamma)$ and $E_r^i(\Gamma) = E(R^i(\Gamma)) \setminus E_s^i(\Gamma)$.

▶ **Lemma 2.1** (⋆). *For every plane graph Γ (with possibly multiple edges) every member of $\mathcal{R}_\Gamma$ is connected. Moreover, if Γ is i-connected, then $R(\Gamma)$ is $(i+1)$-connected, for $i \in [2]$.*

▶ Remark. If Γ is 2-connected then $R(\Gamma)$ can also be shown to be 4-connected. However, 3-connectivity is sufficient for our purposes.

2.2 Graph Structures

A key-concept in our algorithms is the notion of the vertex and the edge structure which is formally defined as follows. Let G be a simple planar graph, $k, l \in \mathbb{N}$, $(S_1, S_2, \dots, S_k)$ be a near-partition of $V(G)$ and $E_1, E_2, \dots, E_l$ be a near-partition of $E(G)$. A *vertex structure* $\mathbf{G}$ is a tuple $(G, S_1, S_2, \dots, S_k)$ and an edge structure $\mathbf{G}'$ is a tuple $(G, E_1, E_2, \dots, E_l)$.

Let $\mathbf{G} = (G, A, X_1, \dots, X_l)$ and $\mathbf{D} = (D, B, Y_1, \dots, Y_l)$ be vertex structures, where $l \in \mathbb{N}$. We say that $\mathbf{D}$ is a *contraction* of $\mathbf{G}$, denoted by $\mathbf{D} \leq_c \mathbf{G}$, if and only if there exists a function $\sigma : V(G) \to V(D)$ satisfying the following *contraction properties*:

1. if $u, v \in V(D)$, $u \neq v \Leftrightarrow \sigma^{-1}(u) \cap \sigma^{-1}(v) = \emptyset$,
2. for every $u \in V(D)$, $G[\sigma^{-1}(u)]$ is connected,
3. $\{u, v\} \in E(D) \Leftrightarrow G[\sigma^{-1}(u) \cup \sigma^{-1}(v)]$ is connected,
4. $\sigma(A) \subseteq B$, and
5. for every $i \in [l]$ and every $x \in Y_i$ it holds that $|\sigma^{-1}(x)| = 1$ and $\sigma^{-1}(x) \in X_i$.

In particular, a graph D is a *contraction* of a graph G if $(D, V(D)) \leq_c (G, V(G))$ and we write $D \leq_c G$. Notice that $\leq_c$ defined for graphs is the usual contraction relation where only conditions 1, 2, and 3 apply. Observe that for any two vertex structures $\mathbf{G}$ and $\mathbf{D}$, where G and D respectively are their associated planar graphs, $\mathbf{D} \leq_c \mathbf{G}$ implies that $D \leq_c G$.

We will also need the following proposition, which follows from the results in [1].

▶ **Proposition 1.** *There exists an algorithm that receives as input a vertex structure $\mathbf{G}$, whose graph has m edges and treewidth at most h, and a vertex structure $\mathbf{D}$, whose graph is connected and has k edges, and outputs whether $\mathbf{D} \leq_c \mathbf{G}$ in $2^{\mathcal{O}(k+h+k\log h)} \cdot m$ steps.*

Let $\mathbf{G} = (G, S_1, \dots, S_l)$ be a vertex structure on a planar graph G, where $l \in \mathbb{N}$. Given a possibly empty $Q \subseteq V(G)$, notice that the tuple $(Q, S_1 \setminus Q, \dots, S_l \setminus Q)$ also forms a near-partition of $V(G)$. Then, we can define the following operator on vertex structures:

$$\mathbf{d}(\mathbf{G}, Q) := (G, Q, S_1 \setminus Q, \dots, S_l \setminus Q).$$

Obviously, $\mathbf{d}(\mathbf{G}, Q)$ is also a vertex structure on G.

Let Γ be a plane graph and consider an $R(\Gamma) \in \mathcal{R}_\Gamma$. By Lemma 2.1 and Observation 1, the graph $R^3(\Gamma)$ is uniquely defined according to $R(\Gamma)$. The following operators on $(\Gamma, R(\Gamma))$ uniquely define a vertex and an edge structure:

$$\begin{aligned}\mathbf{p}(\Gamma, R(\Gamma)) &:= (R^3(\Gamma), V(\Gamma), V_s^1(\Gamma), V_r^1(\Gamma), V_s^2(\Gamma), V_r^2(\Gamma), V_s^3(\Gamma), V_r^3(\Gamma)) \\ \mathbf{e}(\Gamma, R(\Gamma)) &:= (R^3(\Gamma), E_s^3(\Gamma), E_r^3(\Gamma)).\end{aligned}$$

The underlying graph of the above structure is the general graph of $R^3(\Gamma)$ and the vertex sets that form the partition of $V(R^3(\Gamma))$ are the original vertices $V(\Gamma)$, followed by the sets of the subdivision and the radial vertices of each of the three subdivided radial enhancements.

Moreover, the edges are separated to those that have been obtain in $R^3(\Gamma)$ only by subdividing original edges of the graph and those that where obtained after adding radial vertices and edges and subdividing those edges.

Throughout the rest of the paper we will only use structures defined by those three operators. The main purpose is to associate three subdivided radial enhancements to a given plane graph so that (i) the resulting graph is 3-connected and therefore uniquely embeddable, so we can disregard the embedding and treat it as a combinatorial object, and (ii) the vertices and edges of the original graph and each subdivided radial enhancement are distinguishable. In addition, both in PSC and the PTMC problems we try to match the faces of the pattern graph to faces, or parts of faces, of the host graph, the radial enhancements and the corresponding structures seem to be the appropriate tool to use, since we actually only need to match the radial vertices that are added inside each face.

Given a graph G and a non-negative integer k, we define the *ball* around a vertex v of G as the subgraph $B^k_G(v)$ of G induced by the set of vertices at distance at most k from v. Consider now the subgraph $\tilde{G}$ of G induced by the set of vertices that lay outside a given ball $B^k_G(v)$, i.e., $\tilde{G} = G \setminus B^k_G(v)$, and consider the set $\mathcal{C}(\tilde{G})$ of all its connected components. Then by contracting all the edges of every $C \in \mathcal{C}(\tilde{G})$ to a single vertex in G, denoted v_C, we obtain the *k-contracted graph* around v, that will be denoted by G_v. Given a vertex structure $\mathbf{G} = (G, \emptyset, S_1, \ldots, S_l)$ and a non-negative integer k, we define the *k-contracted vertex structure* around a vertex v of the graph G as $\mathbf{G}^{(k)}_v := (G_v, \{v_C \mid C \in \mathcal{C}(\tilde{G})\}, S'_1, \ldots, S'_l)$, where $S'_i = S_i \cap B^k_G(v)$ for every $i \in [l]$.

3 An FPT algorithm for the PSC problem

Given a plane graph Γ we define *the set of non-edges of* Γ: $\overline{E(\Gamma)} = \binom{V(\Gamma)}{2} \setminus E(\Gamma)$. A set of non-edges $S \subseteq \overline{E(\Gamma)}$ will be called *insertable* if there is a way to add the edges to Γ such that no two edges of $E(\Gamma) \cup S$ intersect (apart from any common endpoints). Finally, we define the following relation between two plane graphs Γ and Δ. We say that $\Delta \preceq \Gamma$ if there exists a set $S \subseteq \overline{E(\Gamma)}$ of insertable edges of Γ such that $\Delta \leq_{es} \Gamma'$, where Γ' is obtained from Γ after adding S. Then PSC asks, given two plane graphs Γ and Δ, whether $\Delta \preceq \Gamma$.

The main idea of our algorithm is to create two families of vertex structures, one from the host graph Γ and the other from the pattern graph Δ, such that $\Delta \preceq \Gamma$ if and only if there are two structures $\mathbf{D}$ and $\mathbf{G}$ from each of the above families such that $\mathbf{D} \leq_c \mathbf{G}$. Then, we bound the size of these families and use the algorithm from Proposition 1 to check all pairs of their members for the required property. From now on, in this section, whenever we refer to a structure we will assume that it is a vertex structure.

We define the first family of structures based on the host graph. Given a plane graph Γ, a subdivided radial enhancement of it, $R(\Gamma)$, and a positive integer k, we define the following family of structures:

$$\mathcal{G}_{\Gamma,R(\Gamma),k} := \{\mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset)^{(k)}_v | v \in V(\Gamma)\}.$$

Obviously, $|\mathcal{G}_{\Gamma,R(\Gamma),k}| = |V(\Gamma)|$, regardless of the choice of $R(\Gamma)$ and k.

▶ **Lemma 3.1** ($\star$). *Let Γ be a plane graph, $R(\Gamma)$ a subdivided radial enhancement of Γ, $k \in \mathbb{N}$, $v \in V(\Gamma)$, and $\mathbf{G}_v := \mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset)^{(k)}_v \in \mathcal{G}_{\Gamma,R(\Gamma),k}$. Then the underlying graph G_v of the structure $\mathbf{G}_v$ has treewidth at most $3(k+1)$ and size $\mathcal{O}(|E(\Gamma)|)$.*

In order to define the second family of structures based on the pattern graph we need the following two definitions.

A *facial extension* of a connected plane graph Δ is a connected plane graph Δ^+ satisfying the following properties:

1. $\Delta \subseteq \Delta^+$,
2. $V(\Delta^+) \setminus V(\Delta)$ is an independent set in Δ^+, and
3. for every distinct $x, y \in V(\Delta^+) \setminus V(\Delta)$, $N_{\Delta^+}(x) \not\subseteq N_{\Delta^+}(y)$.

We will denote by $\mathcal{F}_\Delta$ the family of all facial extensions of the graph Δ.

Given a connected plane graph Δ and a subset $L \subseteq E(\Delta)$ of its edges, we denote by $\mathsf{span}(\Delta, L)$ the set of all spanning subgraphs of Δ that contain all the edges in $E(\Delta) \setminus L$. Note that such subgraphs could also contain some edges in L. A *pattern-guess* of a connected plane graph Δ is an element Δ^* of $\mathsf{span}(\Delta^+, E(\Delta))$, for $\Delta^+ \in \mathcal{F}_\Delta$. That is, a spanning subgraph of a facial extension Δ^+ of Δ containing at least all the edges in $E(\Delta^+) \setminus E(\Delta)$. The family of all possible pattern-guesses Δ^* of Δ will be denoted by $\mathcal{PG}_\Delta$.

Now, given a connected plane graph Δ we define the following family of structures: $\mathcal{H}_\Delta := \{\mathbf{d}(\mathbf{p}(\Delta^*, R(\Delta^*)), V(\Delta^*) \setminus V(\Delta)) | \Delta^* \in \mathcal{PG}_\Delta,\ R(\Delta^*) \in \mathcal{R}_{\Delta^*}\}$.

▶ **Lemma 3.2** ($\star$). *If Δ is a connected plane graph then $|\mathcal{H}_\Delta| = 2^{\mathcal{O}(|E(\Delta)| \cdot \log |E(\Delta)|)}$ and, for any structure $\mathbf{D} \in \mathcal{H}_\Delta$, the underlying graph D of $\mathbf{D}$ has size and diameter bounded by $\mathcal{O}(|E(\Delta)|)$.*

▶ **Lemma 3.3** ($\star$). *Let $\mathbf{G} = (G, \emptyset, S_1, \ldots, S_l)$ and $\mathbf{D} = (D, B, Z_1, \ldots, Z_l)$ be two structures, where B is an independent set and $l \in \mathbb{N}$. Then $\mathbf{D} \leq_c \mathbf{G}$ if and only if there exists some $v \in V(G)$ such that $\mathbf{D} \leq_c \mathbf{G}_v^{(k)}$, where $k := \mathbf{diam}(D)$.*

The next theorem ensures the correctness of our algorithm.

▶ **Theorem 3.4.** *Let Γ be a plane graph and Δ be a connected plane graph. It holds that $\Delta \preceq \Gamma$ if and only if for every $R(\Gamma) \in \mathcal{R}_\Gamma$ there exist two structures $\mathbf{G} \in \mathcal{G}_{\Gamma, R(\Gamma), c}$ and $\mathbf{D} \in \mathcal{H}_\Delta$, such that $\mathbf{D} \leq_c \mathbf{G}$, where c is a constant such that $\max\limits_{\Delta^*}\{\mathbf{diam}(R^3(\Delta^*))\} \leq c$.*

Proof. First of all, we know that such a constant c exists from Lemma 3.2 and that in fact $c = \mathcal{O}(|E(\Delta)|)$. Let us first assume that $\Delta \preceq \Gamma$. Then there exists an insertable set of non-edges $S \subseteq \overline{E(\Gamma)}$ and two plane graphs $\Gamma' = (V(\Gamma), E(\Gamma) \cup S)$ and Γ_0, such that $\Gamma_0 \subseteq \Gamma'$ and $\Delta \simeq_{tp} \Gamma_0$. Without loss of generality we may assume that all edges of S are also edges of Γ_0. Let then $\alpha : V(\Gamma_0) \cup F(\Gamma_0) \to V(\Delta) \cup F(\Delta)$ be a topological isomorphism between Γ_0 and Δ. For every edge $e = \{u, v\}$ of S let $e_\alpha = \{\alpha(u), \alpha(v)\}$. We define the sets $S_\alpha = \{e_\alpha \mid e \in S\}$, $S_\alpha^\Delta = S_\alpha \cap E(\Delta)$, and $S_\alpha^\Gamma = S_\alpha \setminus S_\alpha^\Delta$.

We first construct a graph $\Delta^+ \in \mathcal{F}_\Delta$. For this, we add a set of vertices and edges embedded inside some of the faces of Δ in such a way that edges intersect only at their common endpoints. In particular, for each face $f \in F(\Delta)$ with facial mapping $\mathbf{w}(f)$ do the following:

- For each edge $e = \{u, v\}$ that lies inside the region enclosed by $\alpha^{-1}(\mathbf{w}(f))$ in Γ and whose endpoints belong to Γ', add the edge $\{\alpha(u), \alpha(v)\}$ in the interior of f in Δ in such a way that (i) edges intersect only at their common endpoints and (ii) after we extend the mapping α so that it takes into account those edges of Γ that were added in Δ, the following must hold: for any connected component that was inside f and, after the addition of the edges, is in a face f', the preimages of the vertices of that connected component in Γ_0 are inside the region enclosed by $\alpha^{-1}(\mathbf{w}(f'))$.
- Consider the faces $f_1, f_2, \ldots, f_j$ that form the partition of f after the addition of the new edges. For every such face f_i let p_i be the region enclosed by $\alpha^{-1}(\mathbf{w}(f_i))$ in Γ'. Notice that since Δ^+ is connected, the boundary of $\alpha^{-1}(\mathbf{w}(f_i))$ is connected. For every $i \in [j]$

let $\mathcal{C}_{p_i}$ be the set of all connected components that lie entirely in the region enclosed by $\alpha^{-1}(\mathbf{w}(f_i))$ in Γ'. Let $\mathcal{C}_{p_i}^{\emptyset}$ denote the set of all connected components in $\mathcal{C}_{p_i}$ that do not have any neighbors in $B_{\Gamma'}(f_i)$. For every $C_w \in \mathcal{C}_{p_i}$, let S_w be its neighborhood in $B_\Gamma(f_i)$. Consider the Hasse diagram defined by the sets S_w and without loss of generality, let S_1, $S_2, \ldots, S_q$ be its maximal elements. Let then $O_t = \{C_l \in \mathcal{C}_{p_i} \setminus \mathcal{C}_{p_i}^{\emptyset} \mid S_l \subseteq S_t\}$, $t \in [q]$. For every $t \in [q]$, add a vertex u_t in f_i and make it adjacent to the vertices in $\alpha(S_t)$ (notice that since the boundary is again connected there is a unique way to construct the cyclic neighborhood of u_t up to cyclic permutations). We call O_t the *origin* of u_t.

The resulting graph Δ^+ is, by definition, a member of $\mathcal{F}_\Delta$.

To construct Δ^* from Δ^+, for every edge $\{u, v\} \in S$, we remove the edge $\{\alpha(u), \alpha(v)\}$ from Δ^+. Since $\{\alpha(u), \alpha(v)\} \in E(\Delta)$, it follows that

$$\Delta^* \in \mathsf{span}(\Delta^+, E(\Delta)).$$

We now define a function $g_0 : E(\Delta^*) \cup F(\Delta^*) \mapsto E(\Gamma) \cup F(\Gamma)$. Let $f \in F(\Delta^*)$ with facial mapping $\mathbf{w}(f)$. Observe that there is at least one face $f' \in F(\Gamma)$ with facial mapping $\mathbf{w}(f')$, such that for every facial walk $w = (u_1, \ldots, u_k) \in \mathbf{w}(f)$ there is a facial walk $w' \in \mathbf{w}(f')$ of length at least k and a subsequence $(v_1, \ldots, v_k)$ of w' (up to cyclic permutations) with the following properties: $v_i = \alpha(u_i)$ if $v_i \in \Delta$ and $v_i \in V(C)$, for some C in the origin of u_i, if $u_i \in V(\Delta^*) \setminus V(\Delta)$.

Notice that due to planarity the regions defined by those walks (unless the walks are trivial) are mutually nested. Of all such faces (if there are multiple), let f' be the one whose region contains all other regions. Then, $g_0(f) = f'$. We will call the connected component whose vertices belong to that walk the *outermost connected component*.

Recall that, by construction, the new vertices of $V(\Delta^*) \setminus V(\Delta)$ form an independent set. Thus, for each edge $e = \{u, v\} \in E(\Delta^*)$ at most one of its endpoints belongs in $V(\Delta^*) \setminus V(\Delta)$. If both endpoints u, v of e belong to $V(\Delta)$, then we define $g_0(e) = \{\alpha^{-1}(u), \alpha^{-1}(v)\} \in E(\Gamma)$. Otherwise exactly one of u and v, say v, belongs to $V(\Delta^*) \setminus V(\Delta)$. In this case, we define $g_0(e) = \{\alpha^{-1}(u), v'\} \in E(\Gamma)$, where v' is a neighbor of $\alpha^{-1}(u)$ in the outermost connected component in the origin of v.

Let now $R(\Gamma)$ be an arbitrary subdivided radial enhancement of Γ. In order to construct a subdivided radial enhancement $R(\Delta^*)$ of Δ recall that we first subdivide all edges of $R(\Delta^*)$ and then add a radial vertex u_f inside each face $f \in F(\Delta^*)$. For every f let $r_{g_0}(f)$ be the radial vertex of $R(\Gamma)$ that was added in $g_0(f)$. Consider the cyclic neighborhood of $r_{g_0}(f)$ in $R(\Gamma)$. Notice that it can be broken down in $s_1, s_2, \ldots, s_l$ segments where s_i is a facial walk w_i of $\mathbf{w}(g_0(f))$. Let w_i' be the subsequence of the walk that corresponds to a walk z_i in $\mathbf{w}(f)$. Add edges between the u_f and the vertices of the boundary of u_f in such a way that the cyclic neighborhood of u_f is $(z_1, z_2, \ldots, z_l)$. Notice that for every subdivision vertex x of $R(\Delta^*)$ that appears between u_i and u_{i+1} in the facial walk of w, there is a subdivision vertex v_x appearing between v_i and v_{i+1} in the walk w of $\mathbf{w}(f)$. We add an edge $\{u_f, v_x\}$ so that v_x appears between u_i and u_{i+1} in the cyclic neighborhood of u_f (this can be done in a unique way). We extend the mapping g_0 restricted to $E(\Delta^*)$ to the mapping g_1 by mapping every edge $\{u_f, u_i\}$ to the edge $\{r_{g_0}(f), v_i\}$. We also map the edges $\{u_f, x\}$ to the edges $\{r_{g_0(f)}, v_x\}$. Notice that g_1 can be extended to $F(R(\Delta^*))$ similarly to g_0. In the same fashion we extend g_1 to the function g_2 on the graphs $R^2(\Gamma)$ and $R^2(\Delta^*)$ and then to g_3 on the graphs $R^3(\Gamma)$ and $R^3(\Delta^*)$. Recall that

$$\mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset) = (R^3(\Gamma), \emptyset, V(\Gamma), V_s^1(\Gamma), V_r^1(\Gamma), \ldots, V_s^3(\Gamma), V_r^3(\Gamma)),$$

and that

$$\mathbf{d}(\mathbf{p}(\Delta^*, R(\Delta^*)), B) =$$
$$= (R^3(\Delta^*), V(\Delta^*) \setminus V(\Delta), V(\Delta), V_s^1(\Delta^*), V_r^1(\Delta^*), \ldots, V_s^3(\Delta^*), V_r^3(\Delta^*)).$$

Let now $\sigma : V(R^3(\Gamma)) \to V(R^3(\Delta^*))$ such that:

$$\sigma(v) = \begin{cases} u & \text{if } v \in V(\Gamma), u \in V(\Delta), \text{ and } \alpha^{-1}(u) = v \in V(\Gamma) \\ z & \text{if } v \in V_s^i(\Gamma) \text{ and there exists } u \in V_s^i(\Delta^*) \text{ with } g^i(e) = e', i \in [3], \\ & \text{where } z \text{ (resp. } v\text{) is the subdivision vertex of the edge } e \text{ (resp. } e'\text{)} \\ w & \text{if } v \in V_r^i(\Gamma) \text{ and there exists } u \in V_r^i(\Delta^*) \text{ with } g^i(f) = f', i \in [3], \\ & \text{where } w \text{ (resp. } v\text{) is the radial vertex added in face } f \text{ (resp. } f'\text{)} \\ x & \text{where } x \in B \text{ such that the distance between } v \text{ and the vertices in} \\ & O_x \text{ in } R^3(\Gamma) \text{ is minimized} \end{cases}$$

It is quite straightforward to verify that σ satisfies the five required contraction properties and thus $\mathbf{d}(\mathbf{p}(\Delta^*, R(\Delta^*)), B) \leq_c \mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset)$. Therefore, since these two structures satisfy the conditions of Lemma 3.3, we conclude that there exists some $v \in V(\Gamma)$ such that $\mathbf{d}(\mathbf{p}(\Delta^*, R(\Delta^*)), B) \leq_c \mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset)_v^{\mathbf{diam}(R^3(\Delta^*))}$. Notice now that $\mathbf{d}(\mathbf{p}(\Delta^*, R(\Delta^*)), B) \in \mathcal{H}_\Delta$ and that $\mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset)_v^{\mathbf{diam}(R^3(\Delta^*))}$ is a minor of $\mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset)_v^c \in \mathcal{G}_{\Gamma, R(\Gamma), c}$ and we have proven the first direction.

Suppose now that for every $R(\Gamma) \in \mathcal{R}_\Gamma$ there exist two structures $\mathbf{G} \in \mathcal{G}_{\Gamma,R(\Gamma),c}$ and $\mathbf{D} \in \mathcal{H}_\Delta$, such that $\mathbf{D} \leq_c \mathbf{G}$. This is the same as saying that for every $R(\Gamma) \in \mathcal{R}_\Gamma$ there exist a $\Delta^+ \in \mathcal{F}_\Delta$, a $\Delta^* \in \mathsf{span}(\Delta^+, E(\Delta))$, and an $R(\Delta^*) \in \mathcal{R}_{\Delta^*}$ such that $\mathbf{d}(\mathbf{p}(\Delta^*, R(\Delta^*)), B) \leq_c \mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), \emptyset)$. Let then Γ' be the plane graph that results from $R^3(\Gamma)$ if we contract all connected components of $R^3(\Gamma)[\sigma^{-1}(B)]$. It follows immediately that $\Gamma' \simeq_{tp} R^3(\Delta^*)$. Let

$$\alpha : V(\Gamma') \cup F(\Gamma') \to V(R^3(\Delta^*)) \cup F(R^3(\Delta^*))$$

be a topological isomorphism between Γ' and $R^3(\Delta^*)$. Then, for each edge $\{u, v\} \in E(\Delta) \setminus E(\Delta^*)$ there is a face $f \in F(\Gamma)$ such that both $\alpha^{-1}(u)$ and $\alpha^{-1}(v)$ belong to a member of the facial mapping of f. Hence, the set $S = \{\{\alpha^{-1}(u), \alpha^{-1}(v)\} \mid \{u, v\} \in E(\Delta) \setminus E(\Delta^*)\}$ is insertable in Γ. Hence, $\Delta \preceq \Gamma$. ◀

▶ **Theorem 3.5** (⋆). *There exists an algorithm that, given as input an n-edge plane graph Γ and a connected k-edge plane graph Δ, decides whether $\Delta \preceq \Gamma$ in $2^{\mathcal{O}(k \log k)} \cdot n^2$ steps.*

4 An FPT algorithm for the PTMC problem

We need the following definitions and results before we are ready to prove the main result of this section.

Given a plane graph Γ and a non-negative integer k, we say that a graph Γ' is a k-face completion of Γ if it can be obtained from Γ in the following way; for every $f \in F(\Gamma)$ we add a set E_f of at most k edges to Γ such that the endpoints of the edges in E_f are vertices of Γ that belong to the boundary of f, all the edges E_f lie inside f, they do not intersect Γ in any points other than their endpoints, and finally they do not intersect each other.

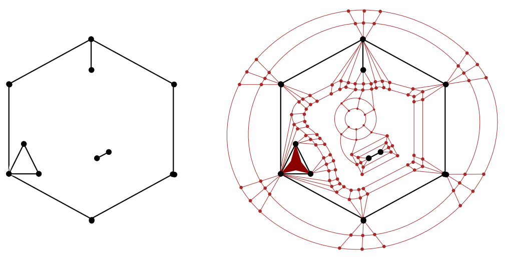


Figure 2 This figure depicts the construction of $\Gamma_{3,2}$ from Γ (Γ is the graph on the left).

Let r and q be integers such that $r \in \mathbb{N}_{\geq 3}, q \in \mathbb{N}_{\geq 1}$. A (r, q)-*cylinder*, denoted by $C_{r,q}$, is the Cartesian product of a cycle on r vertices and a path on q vertices. We will refer to r as the *length* and q as the *width* of $C_{r,q}$. Note here that $C_{r,q}$ is a 3-connected graph and thus, by Whitney's Theorem, it is uniquely embeddable (up to homeomorphism) in the sphere. Furthermore, $C_{r,q}$ has exactly two non-square faces f_1 and f_2 that are incident only with vertices of degree 3. We call one of the faces f_1 and f_2 the *interior* of $C_{r,q}$ and the other the *exterior* of $C_{r,q}$. We call the vertices incident to the interior (exterior) of $C_{r,q}$ *base* (*roof*) of $C_{r,q}$.

Let Γ be a plane graph. We give the definition of the graph $\Gamma_{r,q}$ for $r \in \mathbb{N}_{\geq 3}$ and $q \in \mathbb{N}_{\geq 3}$. Let $f_i \in F(\Gamma)$ and let $\Theta^i_1, \ldots, \Theta^i_{\rho_i}$ be the connected components of $B_\Gamma(f_i)$. For each Θ^i_j, we denote by $\sigma_{j,i}$ the length of a facial walk of Θ^i_j. We then add a copy C^i_j of $(\sigma_{j,i} \cdot r, q)$-cylinder in the embedding of Γ such that Θ^i_j is contained in the interior of C^i_j and all $\Theta^i_1, \ldots, \Theta^i_{j-1}, \ldots, \Theta^i_{j+1} \ldots, \Theta^i_{\rho_i}$ are contained in the exterior of C^i_j. Then we partition the base of C^i_j into $\sigma_{j,i}$ parts Q_l, $l \in \sigma_{j,i}$ each consisting of r consecutive base vertices. Let $(u^1_{j,i}, u^2_{j,i}, \ldots, u^{\sigma_{j,i}}_{j,i}, u^1_{j,i})$ be a facial walk of $\Theta_{j,i}$. We join by r edges the vertex $u^x_{j,i}$ to all the vertices of the set Q_l, $l \in \sigma_{j,i}$. We apply this enhancement for each connected component of the boundary of each face of Γ and we denote the resulting graph by $\widehat{\Gamma}_{r,q}$.

We call a face f_i of $\widehat{\Gamma}_{r,q}$ non-trivial if $B_{\widehat{\Gamma}_{r,q}}(f_i)$ has more than one connected components $\Theta^i_1, \ldots, \Theta^i_{\rho_i}$. Notice that if f_i is non-trivial, each Θ^i_j is the roof of some previously added cylinder. For each such cylinder, let J^i_j be a set of r consecutive vertices of its roof. We add inside f_i a copy C_{f_i} of $C_{\rho_i \cdot r,q}$ such that its base is a subset of f_i and let $\{I_1, \ldots, I_{\rho_i}\}$ be a partition of its roof in ρ_i parts, each consisting of r consecutive base vertices. For each $x \in \{1, \ldots, \rho_i\}$ we add r edges each connecting a vertex of J^i_j with some vertex of I_x in a way that the resulting embedding remains plane (there is a unique way for this to be done). We apply this enhancement for each non-trivial face of $\widehat{\Gamma}_{r,q}$ and the resulting graph is the graph $\Gamma_{r,q}$. Notice that $\Gamma_{r,q}$ is not uniquely defined as its definition depends on the choice of the sets J_x. From now on, we always consider an arbitrary choice for $\Gamma_{r,q}$ and we call $\Gamma_{r,q}$ the (r, q)-*cylindrical enhancement of* Γ. Finally, given a plane graph Γ and $r, q \in \mathbb{N}_{\geq 3}$. Let $V^0_{\Gamma,r,q} = V(\Gamma)$ and $V^n_{\Gamma,r,q} = V(\Gamma_{r,q}) \setminus V(\Gamma)$ and notice that $\deg_{\Gamma_{r,q}}(v) \leq 4$, for every $v \in V^n_{\Gamma,r,q}$. (For an example, see Figure 2.) Given a positive integer k, we denote by $\tilde{\Gamma}_k$ the graph $\Gamma_{2\cdot k, 8\cdot k}$.

We are now ready to state one of the main results of this section.

▶ **Theorem 4.1** ($\star$). *Let* Γ *and* Δ *be plane graphs where* Δ *is connected and* $k = |E(\Delta)|^{2^{|E(\Delta)|}}$. *There exists a* k*-face completion* Γ^+ *of* Γ *such that* $\Delta \leq_{etm} \Gamma^+$ *if and only if* $\Delta \leq^S_{etm} \tilde{\Gamma}_k$ *where* $S = V(\tilde{\Gamma}_k) \setminus V(\Gamma) = V^n_{\Gamma,2\cdot k,8\cdot k}$.

Moreover, we have the following.

▶ **Theorem 4.2** ($\star$). *There exists an algorithm that given two plane graphs Γ and Δ and a set $V \subseteq V(\Gamma)$ with $\deg_\Gamma(z) \leq c$, for every $z \in V$ outputs a graph Γ', with $\Gamma' \subseteq_{sp} \Gamma$ and $\mathbf{tw}(\Gamma') = \mathcal{O}(f(|E(\Delta)|))$, for some computable function f such that $\Delta \leq^V_{etm} \Gamma$ if and only if $\Delta \leq^V_{etm} \Gamma'$. This algorithm runs in $\mathcal{O}_{|E(\Delta)|}(|E(\Gamma)|)$ steps.*

Let Γ be a connected plane graph and $Z \subseteq V(\Gamma)$, we define the following pair of vertex and edge structures:

$$\mathbf{G}_{\Gamma,Z} := (\mathbf{d}(\mathbf{p}(\Gamma, R(\Gamma)), Z), \mathbf{e}(\Gamma, R(\Gamma))).$$

Given two connected plane graphs Δ and Γ and $Z \subseteq V(\Gamma)$ we say that $\mathbf{G}_{\Delta,\emptyset}$ is a *restricted topological minor* of $\mathbf{G}_{\Gamma,Z}$, denoted by $\mathbf{G}_{\Delta,\emptyset} \leq_{rtm} \mathbf{G}_{\Gamma,Z}$, if and only if there exist two functions $f_1 : V(R^3(\Delta)) \to 2^{V(R^3(\Gamma))}$ and $f_2 : E(R^3(\Delta)) \to 2^{E(R^3(\Gamma))}$ satisfying the following:

1. for every $x \in V(\Delta)$, $f_1(x) \in V(\Gamma) \setminus Z$ and $|f_1(x)| = 1$,
2. for every $x \in \cup_{i\in[3]} V(R^i_s(\Delta))$, $f_1(x) \notin \cup_{i\in[3]}(V(R^i_r(\Gamma)))$ and $|f_1(x)| = 1$,
3. for every $x, y \in \cup_{i\in[3]} V(R^i_r(\Delta))$ is connected and $f_1(x) \cap f_1(y) = \emptyset$,
4. for every $xy \in E^3_s(\Delta)$, $G[f_2(xy)]$ is a path between $f_1(x)$ and $f_1(y)$ and $f_2(xy) \subseteq E^3_s(\Gamma)$, and
5. for every $xy \in E^3_r(\Delta)$, $G[f_2(xy)]$ is a path between some vertex of $f_1(x)$ and some vertex of $f_1(y)$.

▶ **Theorem 4.3.** *Let Γ, Δ be two connected plane graphs and $Z \subseteq V(\Gamma)$. Then $\Delta \leq^Z_{etm} \Gamma$ if and only if $\mathbf{G}_{\Delta,\emptyset} \leq_{rtm} \mathbf{G}_{\Gamma,Z}$.*

Our algorithm for PTMC . Let Γ and Δ be two plane graphs, where Δ is connected. From Therorem 4.1 we construct a cylindrical enhancement $\tilde{\Gamma}_k$ of Γ, where the vertices of the set $S = V^n_{\Gamma,2\cdot k,8\cdot k}$ have degree bounded by a constant such that Δ, Γ are a yes instance if and only if $\Delta \leq^S_{etm} \tilde{\Gamma}_k$. Then, the algorithm of Theorem 4.1 with inputs $\tilde{\Gamma}_k, \Delta, S$ outputs a graph Γ' with $\Gamma' \subseteq_{sp} \Gamma$ and $\mathbf{tw}(\Gamma') = \mathcal{O}(f(|E(\Delta)|))$. Moreover, Theorem 4.3 translates Γ', Δ, and S to two structures $\mathbf{G}_{\Delta,\emptyset}$ and $\mathbf{G}_{\Gamma,S}$, for which $\Delta \leq^S_{etm} \Gamma$ if and only if $\mathbf{G}_{\Delta,\emptyset} \leq_{rtm} \mathbf{G}_{\Gamma,S}$. Notice that the relation $\mathbf{G}_{\Delta,\emptyset} \leq_{rtm} \mathbf{G}_{\Gamma,S}$ can be expressed in Monadic Second Order Logic. Finally, by observing that $\mathbf{tw}(R^3(\Gamma)) = \mathcal{O}(f(|E(\Delta)|))$ we can employ Courcelle's Theorem [5] to obtain an $f(|E(\Delta)|) \cdot m^2$ time algorithm, for some computable function f.

5 Extensions

Our approach for the PSC problem can also solve the Plane Induced Subgraph Completion problem, with the same running time, where instead of an embedded subgraph we ask for an embedded *induced subgraph*. The only modification would be at the definition of a facial extension of Δ where we would additionally require that every connected graph Δ^+ contains Δ as an *induced* subgraph.

In the PTMC problem the connectivity of Δ is only required in the proof of Theorem 4.1 (that has been omitted). We would like to remark here that if we disregard the embedding of Δ then the Proposition holds for disconnected graphs as well. In this case by modifying the algorithm for PTMC we may obtain an FPT algorithm that given a plane graph Γ and a *planar* graph D decides whether there exists a face completion of Γ, say Γ^+, such that D is a rooted topological minor of Γ. That is, each vertex of D is mapped to a specified vertex of Γ. Notice that this approach also permits us to solve the Planar Disjoint Paths Completion problem where we allow edge additions inside all faces of Γ (in contrast to [2] where edge additions are allowed only inside a specified face of Γ).

Finally, with the same cylindrical enhancement that we apply for PTMC and the extra restriction that the sets of vertices of the enhanced graph that are contracted to a vertex of the pattern graph Δ contain only vertices of the initial graph we can solve the Plane Minor Completion problem. In these last two cases, however, only the existence of an FPT algorithm is verified (since both would be derived by Courcelle's Theorem).

References

1 Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos. Fast minor testing in planar graphs. *Algorithmica*, 64(1):69–84, 2012.

2 Isolde Adler, Stavros G. Kolliopoulos, and Dimitrios M. Thilikos. Planar disjoint-paths completion. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 80–93. Springer, 2011.

3 Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michal Pilipczuk. A subexponential parameterized algorithm for proper interval completion. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 173–184, 2014.

4 V. Chvatal. Hamiltonian cycles. In E.L. Lawler, J.K. Lenstra, A.H.G.Rinnooy Kan, and D.B. Shmoys (Eds.), editors, *The Traveling Salesman Problem*, pages 403–429. Wiley, New York, 1985.

5 Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *ITA*, 26:257–286, 1992.

6 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

7 Michael B. Dillencourt. Finding hamiltonian cycles in delaunay triangulations is np-complete. *Discrete Applied Mathematics*, 64(3):207 – 217, 1996.

8 Pål Grønås Drange, Fedor V. Fomin, Michal Pilipczuk, and Yngve Villanger. Exploring subexponential parameterized complexity of completion problems. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 288–299, 2014.

9 Haim Kaplan, Ron Shamir, and Robert Endre Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28(5):1906–1922, 1999.

10 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins series in the mathematical sciences. Johns Hopkins University Press, 2001.

11 Yngve Villanger, Pinar Heggernes, Christophe Paul, and Jan Arne Telle. Interval completion is fixed parameter tractable. *SIAM J. Comput.*, 38(5):2007–2020, 2009.

12 Hassler Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):pp. 150–168, 1932.

13 M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

Some Lower Bounds in Parameterized AC^0*

Yijia Chen[1] **and Jörg Flum**[2]

1 **School of Computer Science, Fudan University, Shanghai, China.**
`yijiachen@fudan.edu.cn`

2 **Mathematisches Institut, Universität Freiburg, Germany.**
`joerg.flum@math.uni-freiburg.de`

Abstract

We demonstrate some lower bounds for parameterized problems via parameterized classes corresponding to the classical AC^0. Among others, we derive such a lower bound for all fpt-approximations of the parameterized clique problem and for a parameterized halting problem, which recently turned out to link problems of computational complexity, descriptive complexity, and proof theory. To show the first lower bound, we prove a strong AC^0 version of the planted clique conjecture: AC^0-circuits asymptotically almost surely can not distinguish between a random graph and this graph with a randomly planted clique of any size $\leq n^{\xi}$ (where $0 \leq \xi < 1$).

1998 ACM Subject Classification F.1.1 Unbounded-action devices, F.1.3 Complexity hierarchies, F.4.1 Computational logic

Keywords and phrases parameterized AC^0, lower bound, clique, halting problem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.27

1 Introduction

For $k \in \mathbb{N}$ the k-clique problem asks, given a graph G, whether it contains a clique of size k. In [20], Rossman showed that the k-clique problem has no bounded-depth and unbounded-fan-in circuits of size $O(n^{k/4})$. Therefore, there doesn't exist a family $(\mathsf{C}_{n,k})_{n,k\in\mathbb{N}}$ of circuits such that for some functions $d, f : \mathbb{N} \to \mathbb{N}$,

- every $\mathsf{C}_{n,k}$ has depth at most $d(k)$ and size bounded by $f(k) \cdot n^{k/4}$,
- an n-vertex graph G has a k-clique if and only if $\mathsf{C}_{n,k}(G) = 1$.

If the constraint on the depth of the circuits could be removed, then we would immediately obtain that the *parameterized clique problem*

p-CLIQUE	
Instance:	A graph G and $k \in \mathbb{N}$.
Parameter:	k.
Problem:	Does G contain a clique of size k?

cannot be solved in time $f(k) \cdot n^{O(1)}$. Thus, p-CLIQUE would not be fixed-parameter tractable and hence, FPT $\neq$ W[1] since p-CLIQUE is in the parameterized class W[1]. Therefore, Rossman's result may be viewed as an AC^0 version of FPT $\neq$ W[1], an inequality conjectured by most experts in the field (recall that the complexity class AC^0 contains all problems that can be computed by bounded-depth and unbounded fan-in circuits of polynomial size).

* This research is partially supported by the Sino-German Center for Research Promotion (CDZ 996) and National Nature Science Foundation of China (Project 61373029).

In [11] Elberfeld et al. introduced the parameterized class para-AC^0 as the AC^0 analog of the class FPT: A problem is in para-AC^0 if it can be computed by *dlogtime-uniform* AC^0-circuits after an (arbitrarily complex) *precomputation* [12] on the parameter. Later in [3] it was shown that para-AC^0 contains the *parameterized vertex cover problem*, one of the archetypal fixed-parameter tractable problems. For various other problems the authors of [3] also proved their membership in para-AC^0. Concerning nonmembership, a result in [6] shows that the parameterized *st*-connectivity problem (p-stConn), i.e., the problem of deciding whether there is a path of length at most k between vertices s and t in a graph G, parameterized by k, is not in para-AC^0. It is worth noting that *st*-connectivity is solvable in polynomial time, and hence, p-stConn $\in$ FPT.

The class AC^0 is one of the best understood classical complexity classes. Already in [1, 13] it was shown that Parity, the problem of deciding whether a binary string contains an even number of 1's, is not in AC^0. Since Parity has a very low complexity, for many other problems, including Vertex-Cover and Clique, the AC^0-lower bound can be easily derived by reductions from Parity. Similarly, as p-Clique $\notin$ para-AC^0, it is not very hard to see, using some appropriate weak parameterized reductions, that many other parameterized problems, including the dominating set problem, are not in para-AC^0.

It is well known that the class AC^0 is intimately connected to first-order logic (FO). In fact, the problems decidable by a dlogtime-uniform AC^0-family of circuits are precisely those definable in FO$(<, +, \times)$, that is, in first-order logic for ordered structures with built-in predicates of addition and multiplication.

Now we can also study various parameterized classes based on fragments of FO$(<, +, \times)$. Let us emphasize that this is not merely an academic exercise. Logic and parameterized complexity are surprisingly intertwined with each other, which, among others, is witnessed by various algorithmic meta-theorems (see e.g. [15]). Moreover, the problem whether there is a logic for PTIME, a central problem of descriptive complexity, turned out (see [9] for a thorough discussion) to be related to the complexity of the parameterized halting problem

p-Halt	
Instance:	$n \in \mathbb{N}$ in *unary* and a nondeterministic Turing machine (NTM) $\mathbb{M}$.
Parameter:	$\lvert\mathbb{M}\rvert$, the size of the machine $\mathbb{M}$.
Problem:	Does $\mathbb{M}$ accept the empty input tape in at most n steps?

In fact, already in [19] it was shown that PTIME has a logic if p-Halt has an algorithm with running time $n^{f(|\mathbb{M}|)}$ for some function f. We get a family $(\mathsf{C}_{n,k})_{n,k\in\mathbb{N}}$ of circuits such that

- every $\mathsf{C}_{n,k}$ has depth 2 and size $g(k) \cdot n$ for some function $g : \mathbb{N} \to \mathbb{N}$,
- an NTM $\mathbb{M}$ accepts the empty input tape in at most n steps if and only if $\mathsf{C}_{n,|\mathbb{M}|}(n, \mathbb{M}) = 1$

by hard-wiring into $\mathsf{C}_{n,k}$ the NTMs of size k which halt on empty input in $\leq n$ steps. Therefore, p-Halt belongs to a *nonuniform* version of para-AC^0. The question arises whether p-Halt $\in$ para-AC^0. A positive answer will yield that p-Halt $\in$ FPT, which is considered to be highly unlikely [9]. Hence, the goal is to show *unconditionally* that p-Halt $\notin$ para-AC^0. To the best of our knowledge, all existing AC^0 lower bounds for natural problems apply to both uniform and nonuniform circuits. Perhaps, in order to settle the complexity of p-Halt with respect to para-AC^0, a better understanding of the uniformity conditions of circuits is really required.

1.1 Our work

In this paper, we investigate lower bounds in terms of para-AC^0. We show that a number of problems are not in this class or in some of its proper subclasses.

Following the framework proposed in [12], we first compare two possible definitions of para-AC^0 depending on different ways to obtain parameterized classes from classical ones. We already mentioned the first one, in which an arbitrary precomputation can be performed on the parameter before a standard computation according to the corresponding classical class. The second approach requires the parameterized problem to be in the classical class if we restrict to instances where the parameter is far smaller than the size of the input. We show that both views lead to the same para-AC^0.

Then we derive a first set of lower bound results: We show that many natural W[1]-hard problems are not in para-AC^0 by arguing that the corresponding reductions from p-Clique can be made in AC^0. Among others, they include the weighted satisfiability problems for classes of propositional formulas, which define the W-hierarchy.

We present a modeltheoretic tool, based on the color-coding method, which allows to show membership in AC^0 (similarly as done in [3] via circuits).

We generalize Rossman's result mentioned at the beginning of this introduction and show that any fpt-approximation of p-Clique is not in para-AC^0. To get this result we prove that AC^0-circuits asymptotically almost surely can not distinguish between a random graph and this graph with a randomly planted clique of any size $\leq n^{\xi}$ with $0 \leq \xi < 1$. Our first proof of the last two results used the sophisticated machinery in [20]. Here we outline a proof, suggested to us anonymously, which is directly built on Beame's *Clique Switching Lemma* [5]. The fpt-approximation lower bound of p-Clique again can be transferred to the weighted satisfiability problems, provided the propositional formulas are of odd depth.

Finally we turn to p-Halt. We are not able to show p-Halt $\notin$ para-AC^0, however, using the decidability of Presburger's arithmetic we prove that p-Halt is not in para-FO$(<,+)$, not even in XFO$(<,+)$. On the other hand, p-Halt $\in$ nonuniform-para-FO$(<,+)$.

Due to space limitations for some proofs we refer to the full version of the paper.

2 Preliminaries

By $\mathbb{N}$ we denote the set of nonnegative integers. For every $n \in \mathbb{N}$ we let $[n] := \{1, \ldots, n\}$. Let $\mathbb{R}$ be the set of real numbers, $\mathbb{R}_{+} := \{r \in \mathbb{R} \mid r > 0\}$, and $\mathbb{R}_{\geq 1} := \{r \in \mathbb{R} \mid r \geq 1\}$. For any set A and $k \in \mathbb{N}$ we define $\binom{A}{k}$ as the class of k-element subsets of A, i.e., $\{S \subseteq A \bigm| |S| = k\}$.

A (simple) graph $G = (V(G), E(G))$ (for short, $G = (V, E)$) is undirected and has no loops and multiple edges. Here, $V(G)$ is the vertex set and $E(G)$ the edge set, respectively. A subset $C \subseteq V(G)$ is a *clique* of G if for every $u, v \in C$ either $u = v$ or $\{u, v\} \in E(G)$. And $D \subseteq V(G)$ is a *dominating set* of G if for every $v \in V(G)$ either $v \in D$ or there exists $u \in D$ with $\{u, v\} \in E(G)$.

2.1 Relational structures and first-order logic

A *vocabulary* τ is a finite set of relation symbols. Each relation symbol has an *arity*. A *structure* $\mathcal{A}$ of vocabulary τ, or simply structure, consists of a finite set A called the *universe*, and an interpretation $R^{\mathcal{A}} \subseteq A^r$ of each r-ary relation symbol $R \in \tau$. For example, a graph G can be identified with a structure $\mathcal{A}(G)$ of vocabulary $\{E\}$ with binary relation symbol E such that $A(G) := V(G)$ and $E^{\mathcal{A}(G)} := \{(u, v) \mid \{u, v\} \in E(G)\}$.

Formulas of first-order logic of vocabulary τ are built up from atomic formulas $x = y$ and $Rx_1 \ldots x_r$, where $x, y, x_1, \ldots, x_r$ are variables and $R \in \tau$ is of arity r, using the boolean connectives and existential and universal quantification. For example, for every $k \geq 1$ let

$$\mathit{clique}_k := \exists x_1 \ldots \exists x_k \Big(\bigwedge_{1 \leq i < j \leq k} (\neg x_i = x_j \wedge E x_i x_j) \Big).$$

Then a graph G has a k-clique if and only if $\mathcal{A}(G) \models \mathit{clique}_k$.

2.2 Parameterized complexity

We fix an alphabet $\Sigma := \{0,1\}$. A *parameterized problem* (Q, κ) consists of a classical problem $Q \subseteq \Sigma^*$ and a function $\kappa : \Sigma^* \to \mathbb{N}$, the *parameterization*, computable in polynomial time. As an example, we have already seen p-CLIQUE in the Introduction. A similar problem is the *parameterized dominating set problem.*

p-DOMINATING-SET	
Instance:	A graph G and $k \in \mathbb{N}$.
Parameter:	k.
Problem:	Does G contain a dominating set of size k?

Both, p-CLIQUE and p-DOMINATING-SET, play an important role in parameterized complexity, mainly because they are complete for the classes W[1] and W[2], respectively. Recall that the classes of the W-hierarchy are defined by taking the closure under fpt-reductions of the following weighted satisfiability problem for suitable classes Γ of propositional formulas.

p-WSAT(Γ)	
Instance:	$\gamma \in \Gamma$ and $k \in \mathbb{N}$.
Parameter:	k.
Problem:	Does γ have a satisfying assignment of Hamming weight k?

▶ **Definition 1.** Let (Q, κ) and (Q', κ') be two parameterized problems. An *fpt-reduction* from (Q, κ) to (Q', κ') is a mapping $R : \Sigma^* \to \Sigma^*$ such that:

- For $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.
- For $x \in \Sigma^*$, $R(x)$ is computable in time $f(\kappa(x)) \cdot |x|^{O(1)}$ for some computable $f : \mathbb{N} \to \mathbb{N}$.
- There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

If there is an fpt-reduction from (Q, κ) to (Q', κ'), then we write $(Q, \kappa) \leq^{\text{fpt}} (Q', \kappa')$.

For $t \geq 0$ and $d \geq 1$ we inductively define the classes $\Gamma_{t,d}$ and $\Delta_{t,d}$ of propositional formulas: $\Gamma_{0,d}$ and $\Delta_{0,d}$ are the class of conjunctions of at most d literals and the class of disjunctions of at most d literals, respectively.

$$\Gamma_{t+1,d} := \Big\{ \bigwedge_{i \in I} \delta_i \mid I \text{ finite and } \delta_i \in \Delta_{t,d} \text{ for all } i \in I \Big\},$$
$$\Delta_{t+1,d} := \Big\{ \bigvee_{i \in I} \gamma_i \mid I \text{ finite and } \gamma_i \in \Gamma_{t,d} \text{ for all } i \in I \Big\}.$$

▶ **Definition 2.** Let $t \geq 1$. The class W[t] of the W-*hierarchy* is defined by

$$\mathrm{W}[t] := \bigcup_{d \geq 1} \big\{ (Q, \kappa) \mid (Q, \kappa) \leq^{\text{fpt}} p\text{-WSAT}(\Gamma_{t,d}) \big\}.$$

Circuit Complexity

A circuit C with n input gates is a directed acyclic graph in which every node (i.e., gate) is labelled by $\bigwedge$, $\bigvee$, $\neg$, or by one of the variables, or by $0, 1$. All $\bigwedge$ and $\bigvee$ gates may have arbitrarily many inputs, i.e., C is of *unbounded fan-in.* The *depth* of C is the length of a longest directed path in C. The *size* of C, denoted by $|\mathsf{C}|$, is the number of gates in C. We often tacitly identify C with the function $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ it computes. Here, n is the number of variables of C and m the number of its *output gates*.

AC^0 is the class of problems that can be computed by circuits of bounded-depth and polynomial size. More precisely:

▶ **Definition 3.** Let $Q \subseteq \Sigma^*$. We say that $Q \in \mathrm{AC}^0$ if there exists a family of boolean circuits $(\mathsf{C}_n)_{n\in\mathbb{N}}$ such that:

(A1) The depth of every C_n is bounded by a fixed constant.

(A2) $|\mathsf{C}_n| = n^{O(1)}$.

(A3) Let $x \in \Sigma^*$. Then ($x \in Q$ if and only if $C_{|x|}(x) = 1$). In particular, every C_n has n input gates.

(A4) $(\mathsf{C}_n)_{n\in\mathbb{N}}$ is *dlogtime-uniform*, that is: there is a *deterministic logtime Turing machine* $\mathbb{M}$ which on input 1^n outputs the circuit C_n. More precisely, $\mathbb{M}$ recognizes the language $\big\{(b, i, 1^n) \bigm| \text{the } i\text{th bit of the binary encoding of } C_n \text{ is } b\big\}$ (cf. Section 6 of [4]).

Often, $(\mathsf{C}_n)_{n\in\mathbb{N}}$ are called AC^0*-circuits*.

We remark that most lower bounds in our paper still hold without the requirement (A4). Therefore, (A4) is irrelevant for most of our results. However, with this uniformity condition, AC^0 characterizes precisely the class of problems that are definable in $\mathrm{FO}(<, +, \times)$ [4].

3 The class para-AC^0 and some natural examples

▶ **Definition 4** ([3])**.** Let (Q, κ) be a parameterized problem. Then (Q, κ) is in para-AC^0 if there exists a family $(\mathsf{C}_{n,k})_{n,k\in\mathbb{N}}$ of circuits such that:

(P1) The depth of every $\mathsf{C}_{n,k}$ is bounded by a fixed constant.

(P2) $|\mathsf{C}_{n,k}| \le f(k) \cdot n^{O(1)}$ for every $n, k \in \mathbb{N}$, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function.

(P3) Let $x \in \Sigma^*$. Then ($x \in Q$ if and only if $\mathsf{C}_{|x|,\kappa(x)}(x) = 1$).

(P4) There is a deterministic Turing machine that on input $(1^n, 1^k)$ computes the circuit $\mathsf{C}_{n,k}$ in time $g(k) + O(\log n)$, where $g : \mathbb{N} \to \mathbb{N}$ is a computable function.

For future reference, we restate Rossman's main result [20] as follows.

▶ **Theorem 5.** *Let $k \in \mathbb{N}$. Then there is no family $(\mathsf{C}_n)_{n\in\mathbb{N}}$ of circuits such that:*

- *The depth of every C_n is bounded by a fixed constant.*
- *The size of C_n is $n^{O(k/4)}$.*
- *Let G be a graph and $n := |V(G)|$. Then G has a k-clique if and only if $\mathsf{C}_n(G) = 1$.*

*In particular, p-*CLIQUE* $\notin$ para-AC^0.*

▶ Remark. Recall that Chen et al. [7] showed that p-CLIQUE has no algorithms of running time $f(k) \cdot |n|^{o(k)}$ unless the Exponential Time Hypothesis (ETH) fails. ETH is apparently stronger than $\mathrm{FPT} \neq \mathrm{W}[1]$. Theorem 5 establishes an AC^0 version of $\mathrm{FPT} \neq \mathrm{W}[1]$.

Next, we give two equivalent characterizations of para-AC^0 (for a proof see the full version). The first one (i.e., between (i) and (ii)) was already mentioned in [11]. Note that in [11] it is required that a problem in para-AC^0 has an AC^0 computable parameterization.

▶ **Proposition 6.** *Let (Q, κ) be a parameterized problem. Consider the following statements.*

(i) $(Q, \kappa) \in$ para-AC^0.

(ii) *There is a computable function $pre : \mathbb{N} \to \Sigma^*$ (i.e., a precomputation) and AC^0-circuits $(\mathsf{C}_n)_{n\in\mathbb{N}}$ such that for $x \in \Sigma^*$,*

$$x \in Q \iff \mathsf{C}_{|(x, pre(\kappa(x)))|}(x, pre(\kappa(x))) = 1.$$

(iii) *Q is decidable and there is a computable function, $h : \mathbb{N} \to \mathbb{N}$ and AC^0-circuits $\big(\mathsf{C}_n\big)_{n\in\mathbb{N}}$ such that for every $x \in \Sigma^*$ with $|x| \geq h(\kappa(x))$,*

$$x \in Q \iff C_{|x|}(x) = 1.$$

Then (iii) ⇒ (i) ⇔ (ii). If, in addition, the parameterization κ can be computed by AC^0-circuits, then (i) ⇒ (iii), i.e., they are all equivalent.

In order to use Theorem 5 to show para-AC^0 lower bounds for other problems, we introduce a more restricted form of fpt-reductions.

▶ **Definition 7.** Let (Q, κ) and (Q', κ') be two parameterized problems. A *para-*AC^0*-reduction* from (Q, κ) to (Q', κ') is a mapping $R : \Sigma^* \to \Sigma^*$ such that:
(R1) For all $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.
(R2) There is a family $(\mathsf{C}_{n,k})_{n,k\in\mathbb{N}}$ of circuits, whose depth is bounded by a fixed constant, such that
1. for all $x \in \Sigma^*$, $\mathsf{C}_{|x|,\kappa(x)}(x)$ outputs $R(x)$;
2. every $|\mathsf{C}_{n,k}| \leq f(k) \cdot |x|^{O(1)}$ for a computable function $f : \mathbb{N} \to \mathbb{N}$;
3. there is a deterministic Turing machine that on input $(1^n, 1^k)$ computes the circuit $\mathsf{C}_{n,k}$ in time $g(k) + O(\log\, n)$, where $g : \mathbb{N} \to \mathbb{N}$ is a computable function.

(R3) There is a computable function $h : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq h(\kappa(x))$ for all $x \in \Sigma^*$.
If there is a para-AC^0-reduction from (Q, κ) to (Q', κ'), then we write $(Q, \kappa) \leq^{\mathrm{pac}} (Q', \kappa')$.

However, in general para-AC^0 is *not* closed under para-AC^0-reductions:

▶ **Example 8.** Define $Q := \big\{(x, b) \;\big|\; x \in \{0, 1\}^* \text{ and } b = \sum_{i\in[|x|]} x_i \mod 2\big\}$. Clearly, Q is equivalent to the classical Parity problem of deciding whether there is an even number of 1's in x. Thus $Q \notin \mathrm{AC}^0$. We define polynomial time computable parameterizations of Q by $\kappa_1(x, b) := 0$ and $\kappa_2(x, b) := \sum_{i\in[|x|]} x_i \mod 2$. Then it is easy to see that $(Q, \kappa_1) \notin$ para-AC^0 and $(Q, \kappa_2) \in$ para-AC^0; yet $(Q, \kappa_1) \leq^{\mathrm{pac}} (Q, \kappa_2)$ by the identity mapping $R(x, b) = (x, b)$.

Note (Q, κ_2) also serves as a counterexample for the direction from (i) to (iii) in Proposition 6.

Therefore we need a further requirement on pac-reductions. The previous example suggests to require the AC^0-computability of the parameterization (as done in [11]). In fact, para-AC^0 is closed under those reductions. However, we choose another requirement, which is simpler to verify and is satisfied by almost all natural reductions.

▶ **Definition 9.** Let (Q, κ) and (Q', κ') be two parameterized problems. A *weak para-*AC^0*-reduction* from (Q, κ) to (Q', κ') is a *para-*AC^0*-reduction* which satisfies:
(R3') There is a computable function $h : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) = h(\kappa(x))$ for all $x \in \Sigma^*$.
$(Q, \kappa) \leq^{\mathrm{pwac}} (Q', \kappa')$ means that there is a weak para-AC^0-reduction from (Q, κ) to (Q', κ').

It is straightforward to verify that para-AC^0 is closed under weak para-AC^0-reductions.

▶ **Lemma 10.** *Let (Q, κ) and (Q', κ') be parameterized problems with $(Q, \kappa) \leq^{pwac} (Q', \kappa')$. If $(Q', \kappa') \in$ para-AC^0, then $(Q, \kappa) \in$ para-AC^0, too.*

It is well known that p-Clique is fpt-reducible to p-Dominating-Set. The reduction presented in the full version of this paper is a weak para-AC^0-reduction. Thus, by Theorem 5 and Lemma 10:

▶ **Proposition 11.** p-Dominating-Set $\notin$ para-AC^0.

▶ **Corollary 12.** *Let $t, d \geq 1$ with $t + d \geq 3$. Then p-WSAT$(\Gamma_{t,d}) \notin$ para-AC^0.*

Proof. For every graph $G = (V, E)$ we define a propositional formula

$$\delta_G := \bigwedge_{\substack{u,v \in V \text{ with} \\ u \neq v \text{ and } \{u,v\} \notin E}} \neg X_u \vee \neg X_v.$$

Clearly, for every $k \in \mathbb{N}$,

$$G \text{ has a } k\text{-clique} \iff \delta_G \text{ has a satisfying assignment of weight } k \tag{1}$$

This gives a weak para-AC^0-reduction from p-CLIQUE to p-WSAT$(\Gamma_{1,2})$, or p-WSAT$(\Gamma_{t,1})$ in case $t \geq 2$. ◀

Similarly, one can show that basic problems like p-SUBGRAPH-ISOMORPHISM, p-HOM, p-EMB, and p-MC(Σ_1^1) are not in para-AC^0 (we use the notations of [12]).

In view of Corollary 12 the reader might wonder about the status of p-WSAT$(\Gamma_{1,1})$. Using the color-coding technique as in [3], one can show that the problem is in fact solvable in para-AC^0. We present a more logic-oriented technique for such proofs. It is based on Proposition 13. It uses FO$(<, +, \times)$ instead of dlogtime-uniform AC^0. We defer the proofs of this proposition and of Proposition 14 to the full version of the paper.

For $n \in \mathbb{N}$ denote by $<^{[n]}$ the natural ordering on $[n]$. If $\mathcal{A}$ is any ordered structure, then $(A, <^{\mathcal{A}})$ is isomorphic to $([|A|], <^{[|A|]})$ and the isomorphism is unique. For ternary relation symbols $+$ and $\times$ we consider the ternary relations $+^{[n]}$ and $\times^{[n]}$ on $[n]$ that are the relations underlying the addition and the multiplication of $\mathbb{N}$ restricted to $[n]$. That is,

$$+^{[n]} := \{(a,b,c) \mid a,b,c \in [n],\ c = a + b\}, \quad \times^{[n]} := \{(a,b,c) \mid a,b,c \in [n],\ c = a \cdot b\}.$$

Let τ be a vocabulary which does not contain $<, +, \times$ and set $\tau_{<,+,\times} := \tau \cup \{<, +, \times\}$. We say that a $\tau_{<,+,\times}$-structure $\mathcal{A}$ *has built-in addition and built-in multiplication* if $(A, <^{\mathcal{A}}, +^{\mathcal{A}}, \times^{\mathcal{A}})$ is isomorphic to $([|A|], <^{[|A|]}, +^{[|A|]}, \times^{[|A|]})$. Sometimes we write $\varphi \in$ FO$(<, +, \times)$ to emphasize that φ is a first-order formula in a vocabulary containing the symbols $<, +, \times$.

▶ **Proposition 13.** *There is a computable function which associates every $k \in \mathbb{N}$ with a structure $\mathcal{C}(k)$ and every FO-formula $\varphi(x)$ with an FO$(<, +, \times)$-sentence χ_φ such that for every structure $\mathcal{A}$,*

$$[\mathcal{A} : \mathcal{C}(k)] \models \chi_\varphi \tag{2}$$
$$\iff \text{there are pairwise distinct } x_1, \dots, x_k \in A \text{ with } \mathcal{A} \models \varphi(x_i) \text{ for every } i \in [k].$$

Here, $[\mathcal{A} : \mathcal{C}(k)] := \mathcal{B} = \big(\mathcal{A} \,\dot\cup\, \mathcal{C}(k), U^{\mathcal{B}}, <^{\mathcal{B}}, +^{\mathcal{B}}, \times^{\mathcal{B}}\big)$ is defined as follows.

- *$\mathcal{A} \,\dot\cup\, \mathcal{C}(k)$ is the disjoint union of $\mathcal{A}$ and $\mathcal{C}(k)$ (see the full version of the paper for the definition of the disjoint union of structures).*
- *$U^{\mathcal{B}} := A$ and $<^{\mathcal{B}}$ is an ordering of B and every element of A precedes all elements of $C(k)$. Furthermore $<^{\mathcal{B}}$ extends the ordering $\prec^{\mathcal{C}(k)}$ given in $\mathcal{C}(k)$.*
- *$\mathcal{B}$ has built-in addition and multiplication.*

▶ **Proposition 14.** *p-WSAT$(\Gamma_{1,1}) \in$ para-AC^0.*

4 Inapproximability of p-Clique by para-AC^0

We recall the notion of fpt approximation introduced in [10]. We present the definition for p-Clique, the problem which interests us. It can easily be generalized to any maximization problem.

If not stated otherwise, $\rho : \mathbb{N} \to \mathbb{R}_{\geq 1}$ is always a computable function such that the mapping $k \mapsto k/\rho(k)$ is nondecreasing and unbounded.

▶ **Definition 15.** An algorithm $\mathbb{A}$ is a *parameterized approximation for* p-Clique *with approximation ratio* ρ if for every graph G and $k \in \mathbb{N}$ with $\omega(G) \geq k$ the algorithm $\mathbb{A}$ computes a clique C of G such that $|C| \geq k/\rho(k)$. Here the clique number $\omega(G)$ is the size of a maximum clique of G. If the running time of $\mathbb{A}$ is bounded by $f(k) \cdot |G|^{O(1)}$ where $f : \mathbb{N} \to \mathbb{N}$ is computable, then $\mathbb{A}$ is an *fpt approximation algorithm.*

We tend to believe that p-Clique has no fpt approximation algorithm for any ratio ρ. Since para-AC^0 is a class of decision problems, in order to prove a lower bound it is more convenient to deal with decision algorithms instead of algorithms computing a clique.

▶ **Definition 16** ([10]). A decision algorithm $\mathbb{A}$ is a *parameterized cost approximation for* p-Clique *with approximation ratio* ρ if for every graph G and $k \in \mathbb{N}$,
- if $k \leq \omega(G)/\rho(\omega(G))$, then $\mathbb{A}$ accepts (G, k);
- if $k > \omega(G)$, then $\mathbb{A}$ rejects (G, k).

In other words, $\mathbb{A}$ decides the *promise* problem:

p-Gap$_\rho$-Clique	
Instance:	A graph G and $k \in \mathbb{N}$ such that either $k \leq \omega(G)/\rho(\omega(G))$ or $k > \omega(G)$.
Parameter:	k.
Problem:	Is $k \leq \omega(G)/\rho(\omega(G))$?

The intuition behind this definition: If G contains a clique far bigger than k, detecting a k-clique might become easier. It is straightforward to verify that if p-Clique has no fpt cost approximation of ratio ρ, then it has no fpt approximation of ratio ρ either [10].

▶ **Theorem 17.** p-Gap$_\rho$-Clique $\notin$ para-AC^0.

Our original proof of this result was based on a generalization of the machinery developed in [20], a generalization we first used to prove that AC^0 circuits are not sensitive to planted cliques of a reasonable size, see Theorem 21. The much simpler proof of Theorem 21 we present here is based on Beame's Clique Switching Lemma [5] (see Section 4.1) and was suggested to us anonymously. In the full version of the paper we apply Theorem 21 to derive Theorem 17.

First we prove a consequence of Theorem 17. For $t \geq 0$, $d \geq 1$ we denote by $\Gamma^-_{t,d}$ the subset of subformulas of $\Gamma_{t,d}$ with only negative literals. Clearly, if $\gamma \in \Gamma^-_{t,d}$ has a satisfying assignment of Hamming weight k, then it has one of weight k' for every $k' < k$. Denote by $\omega(\gamma)$ the maximum Hamming weight of assignments satisfying γ. Then p-Gap$_\rho$-WSat$(\Gamma^-_{t,d})$ can be defined similarly as p-Gap$_\rho$-Clique.

▶ **Proposition 18.** *Let $t, d \geq 1$ with $t + d \geq 3$. Then* p-Gap$_\rho$-WSat$(\Gamma^-_{t,d}) \notin$ para-AC^0.

Proof. Consider the reduction from p-Clique to p-Gap$_\rho$-WSat$(\Gamma_{t,d})$ in the proof of Corollary 12. Clearly $\delta_G \in \Gamma^-_{t,d}$ and δ_G is independent of k. Thus, the equivalence (1) preserves the approximation ratio. The result then follows immediately. ◀

4.1 Beame's Clique Switching Lemma

Let $n \in \mathbb{N}$. We consider graphs with vertex set $[n]$. To represent functions on those graphs, every potential edge $e \in \binom{[n]}{2}$ is encoded by a boolean variable X_e. We set

$$\mathcal{X}_n := \left\{ X_e \;\middle|\; e \in \binom{[n]}{2} \right\}.$$

In particular, $X_e = 1$ means that e is present in the given graph, otherwise $X_e = 0$. Sometimes, it is convenient to understand e as a natural number with $e \in \left[\binom{n}{2}\right]$. Then, e is the eth potential edge in an n-vertex graph, and X_e is the eth variable in $\mathcal{X}_n$.

For every $\ell \in [n]$ and $q \in \mathbb{R}$ with $0 \le q \le 1$ let $\mu \in \mathcal{C}_n^{\ell,q}$ be a *random restriction*, $\mu : \mathcal{X}_n \to \{0, 1, \star\}$ generated as follows:

- Choose $U \in \binom{[n]}{\ell}$ uniformly at random and then set $\mu(X_e) := \star$ for every $e \in \binom{U}{2}$.
- For $e \notin \binom{U}{2}$ we set $\mu(X_e) := 1$ with probability q and $\mu(X_e) := 0$ with probability $1 - q$.

Let F be a boolean function defined on the set of assignments from $\mathcal{X}_n$ to $\{0, 1\}$ and $\mu \in \mathcal{C}_n^{\ell,q}$. The function $F{\restriction_\mu}$ is defined on the set of assignments from $\mu^{-1}(\star)$ to $\{0, 1\}$ by: For $S : \mu^{-1}(\star) \to \{0, 1\}$, we set $F{\restriction_\mu}(S) := F(S \cup \mu)$, where $S \cup \mu : \mathcal{X}_n \to \{0, 1\}$ is the assignment:

$$(S \cup \mu)(X_e) := S(X_e), \text{ if } X_e \in \mu^{-1}(\star) \quad \text{and} \quad (S \cup \mu)(X_e) := \mu(X_e), \text{ otherwise.}$$

Recall that a rooted binary tree is a *decision tree* on some variable set $\mathcal{X} \subseteq \mathcal{X}_n$ if every leaf is labeled either 0 or 1, every internal node is labelled by a variable of $\mathcal{X}$, and the edges between an internal node and its two children are labelled 0 and 1. The *vertex height* of a path P in T is the number of distinct vertices occurring in edges e such that the corresponding X_e appears in P. The *vertex height* $|T|_v$ of T is the maximum vertex height of a path in T.

For any boolean function F as above, we set

$$\text{DTdepth}_{\text{vertex}}(F) = \min\{|T|_v \mid T \text{ a decision tree computing } F\}.$$

The following lemma is the imbalanced version of [5, Lemma 3] mentioned in the first paragraph of page 12 of that paper. The *vertex length* of a clause is the number of distinct vertices in edges e with X_e appearing in this clause.

▶ **Lemma 19** ([5]). *Let $n, r \in \mathbb{N}$ and $0 \le q \le 1/2$. Moreover, let F be a* DNF*-formula of variable set $\mathcal{X}_n$ with conjunctive clauses of vertex length at most r. For $s, \ell \in \mathbb{N}$ with $\ell := pn$, where $s \ge 0$ and $\ell := pn$ with $p \le 1/(r(2/q)^{(r+s)/2})$, we have*

$$\Pr_{\mu \in \mathcal{C}_n^{\ell,q}} \left[\text{DTdepth}_{\text{vertex}}(F{\restriction_\mu}) > s \right] < \frac{8\left((2/q)^{(s+r-1)/2} pr\right)^s}{3}.$$

In the full version of the paper we apply Lemma 19 inductively on bounded-depth circuits and show

▶ **Lemma 20.** *Assume*

- *$k : \mathbb{N} \to \mathbb{R}_+$ with $k(n) \le \log_2 n$ for all sufficiently large n and $\lim_{n\to\infty} k(n) = \infty$,*
- *$S, d : \mathbb{N} \to \mathbb{N}$ with $S(n) \ge n$.*

Define $q : \mathbb{N} \to \mathbb{R}_+$ and $s : \mathbb{N} \to \mathbb{N}$ by

$$q(n) := n^{-1/k(n)} \quad \text{and} \quad s(n) := \left\lfloor \sqrt{k(n)(\log_n S(n) d(n))} \right\rfloor, \tag{3}$$

and $\ell_i : \mathbb{N} \to \mathbb{N}$ *inductively by*

$$\ell_0(n) := n \quad \textit{and} \quad \ell_{i+1}(n) := \left\lfloor \frac{\ell_i(n)}{n^{5s(n)/k(n)}} \right\rfloor . \tag{4}$$

Then, $\ell_{d(n)}(n) = n^{1-\Theta\left(5d(n)\sqrt{(\log_n S(n)d(n))/k(n)}\right)}$ *and for every circuit* C *with variable set* $\mathcal{X}_n$*, size bounded by* $S(n)$*, and depth bounded by* $d(n)$*,*

$$\Pr_{\mu \in \mathfrak{C}_n^{\ell_{d(n)}(n),q(n)}} \left[\mathsf{C}{\restriction_\mu} \textit{ is constant} \right] = 1 - o(1).$$

4.2 A strong AC^0 version of the planted clique conjecture

In the standard planted clique problem, we are given a graph G whose edges are generated by starting with a random graph with universe $[n]$ and edge probability $1/2$, then "planting" (adding edges to make) a random clique on k vertices; the problem asks for efficient algorithms finding such a clique of size k. The problem was addressed in [17, 18, 2], among many others. It is conjectured that no such algorithm exists. Here, as a consequence of Lemma 20, we prove a statement considerably stronger than the AC^0 version of this conjecture.

Let us be more precise. The Erdős-Rényi probability space $\mathrm{ER}(n,p)$, where $n \in \mathbb{N}$ and $0 \le p \le 1$, is obtained as follows. We start with the set $[n]$ of vertices. Then we choose every $e \in \binom{[n]}{2}$ as an edge of G with probability p, independently of the choices of other edges.

For $G \in \mathrm{ER}(n,1/2)$ the expected size of a maximum clique is approximately $2 \log\, n$. Therefore G almost surely has no clique of size, say, $4 \log\, n$. For any graph G with vertex set $[n]$ and any $A \subseteq [n]$ we denote by $G + C(A)$ the graph obtained from G by adding edges such that the subgraph induced on A is a clique. For $n, c \in \mathbb{N}$ with $c \in [n]$ and $p \in \mathbb{R}$ with $0 \le p \le 1$ we consider a second distribution $\mathrm{ER}(n,p,c)$: Pick a random graph $G \in \mathrm{ER}(n,p)$ and a uniformly random subset A of $[n]$ of size c and plant in G a clique on A, thus getting the graph $G + C(A)$. The notation $(G,A) \in \mathrm{ER}(n,p,c)$ should give the information that the random graph was G and that the random subset of $[n]$ of size c was A.

▶ **Theorem 21.** *Let* $k : \mathbb{N} \to \mathbb{R}^+$ *with* $\lim_{n\to\infty} k(n) = \infty$*, and* $c : \mathbb{N} \to \mathbb{N}$ *with* $c(n) \le n^\xi$ *for some* $0 \le \xi < 1$*. Then for all* AC^0 *circuits* $(\mathsf{C}_n)_{n\in\mathbb{N}}$*,*

$$\lim_{n\to\infty} \Pr_{(G,A)\in \mathrm{ER}(n,n^{-1/k(n)},\ c(n))} \left[\mathsf{C}_n(G) = \mathsf{C}_n(G + C(A)) \right] = 1.$$

Proof. We assume that $k(n) \le \log_2\, n$ for all sufficiently large n. The general case can be reduced to it by standard techniques from probability theory.

Let $(\mathsf{C}_n)_{n\in\mathbb{N}}$ be a family of circuits such that for some $\bar{d}, t \in \mathbb{N}$ every C_n has depth at most $\bar{d}$ and size bounded by n^t. In order to apply Lemma 20, we set for $n \in \mathbb{N}$,

$$S(n) = n^t \text{ and } d(n) = \bar{d}. \tag{5}$$

By Lemma 20, it follows that (recall that $q(n) = n^{-1/k(n)}$)

$$\Pr_{\mu \in \mathfrak{C}_n^{\ell_{\bar{d}}(n),q(n)}} \left[\mathsf{C}_n{\restriction_\mu} \text{ is constant} \right] = 1 - o(1). \tag{6}$$

Furthermore, $\ell_{\bar{d}}(n) = n^{1-\Theta\left(5d(n)\sqrt{(\log_n S(n)d(n))/k(n)}\right)} = n^{1-o(1)}$; the first equality holds by Lemma 20 and the second by (5). The key step consists of the following random process, which generates $(G,A) \in \mathrm{ER}(n, n^{-1/k(n)},\ c(n))$ from $\mu \in \mathfrak{C}_n^{\ell_{\bar{d}}(n),q(n)}$.

(a) Let $V(G) := [n]$.
(b) Add edges $e \in \binom{[n]}{2}$ with $\mu(e) = 1$ to $E(G)$.
(c) Recall that $\mu^{-1}(\star) = \binom{U}{2}$, where $U \in \binom{[n]}{\ell_{\bar{d}}(n)}$ was chosen uniformly at random. For every $e \in \binom{U}{2}$, add e to $E(G)$ with probability $q(n)$.
(d) Choose $A \in \binom{U}{c(n)}$ uniformly at random. Note that this is possible as $|U| = \ell_{\bar{d}}(n) = n^{1-o(1)} > n^{\xi} \geq c(n)$ for sufficiently large n.

By (b)–(d), G and $G + C(A)$ contain the same edges from $\binom{[n]}{2} \setminus \mu^{-1}(\star)$. Thus, by (6), $\mathsf{C}_n(G) = \mathsf{C}_n(G + C(A))$ with high probability. By (c) and (d), A can be viewed as being chosen in $\binom{[n]}{c(n)}$ uniformly at random. ◀

5 The complexity of p-Halt

We already mentioned in the abstract of this article that the complexity of the parameterized halting problem p-HALT is linked to open problems in computational complexity, descriptive complexity, and proof theory [9]. For example, p-HALT $\in$ XP is equivalent to the existence of an almost optimal algorithm for the set of tautologies of propositional logic, or to the fact that a certain logic, presented in [16], is a logic for PTIME. Both statements are conjectured to be false. The origin of our interest in para-AC^0 was our hope to get a lower bound on the complexity of p-HALT in terms of para-AC^0, that is, to show p-HALT $\notin$ para-AC^0. But also this problem remains open. We know that AC^0 corresponds to FO$(<, +, \times)$, first-order logic with an ordering relation and built-in addition and multiplication. In this section we prove that p-HALT $\notin$ para-FO$(<, +)$, even p-HALT $\notin$ XFO$(<, +)$, hold unconditionally, to our knowledge the best known lower bound for the complexity of p-HALT.

Recall that in the paragraph preceding Proposition 13 we defined the natural ordering $<^{[n]}$ on $[n]$ and the ternary relations $+^{[n]}$ and $\times^{[n]}$ of addition and multiplication, respectively, on $[n]$. Now we address the definition of XFO$(<, +, \times)$. For this purpose we view inputs to parameterized problems as structures.

Any string $x \in \Sigma^*$ with $|x| = n$ can be identified with the $\{<, +, \times, One\}$-structure $\langle x \rangle^{<,+,\times} := ([n], <^{[n]}, +^{[n]}, \times^{[n]}, One^{[n]})$. Here $i \in [n]$ is in $One^{[n]}$, the interpretation of the unary relation symbol One, if and only if the ith bit of x is a '1'. The structures $\langle x \rangle^{<,+}$ and $\langle x \rangle^{<}$ are reducts of $\langle x \rangle^{<,+,\times}$ over the vocabularies $\{<, +, One\}$ and $\{<, One\}$, respectively.

▶ **Definition 22.** Let (Q, κ) be a parameterized problem. Then $(Q, \kappa) \in$ XFO$(<, +, \times)$ if there is a computable function that assigns to every $k \in \mathbb{N}$ a first-order sentence φ_k such that for every instance x of (Q, κ) we have $\big(x \in Q \iff \langle x \rangle^{<,+,\times} \models \varphi_{\kappa(x)}\big)$. Analogously, the class XFO$(<, +)$ is defined.

▶ **Theorem 23.** p-HALT $\notin$ XFO$(<, +)$.

Proof. For a contradiction we assume that p-HALT $\in$ XFO$(<, +)$ and show that then the halting problem for Turing machines would be decidable.

Assume that there is a computable function that assigns to every $k \in \mathbb{N}$ a first-order sentence φ_k such that $\big((1^n, \mathbb{M}) \in p\text{-HALT} \iff \langle(1^n, \mathbb{M})\rangle^{<,+} \models \varphi_{|\mathbb{M}|}\big)$ for every instance $(1^n, \mathbb{M})$. Fix $\mathbb{M}$. There is a first-order interpretation I that for every $n \in \mathbb{N}$ defines an isomorphic copy of $\langle(1^n, \mathbb{M})\rangle^{<,+}$ in $([n], <^{[n]}, +^{[n]})$: Let $c(n) := |(1^n, \mathbb{M})|$ be the length of the string $(1^n, \mathbb{M})$. We define the interpretation stepwise. As $\mathbb{M}$ is fixed, it is easy to see that we can define in $([n], <^{[n]}, +^{[n]})$ a subset S of $c(n)$ elements of $[n]^s$ for suitable s, the universe of the structure defined by the intended interpretation. We order S by the lexicographical order on s-tuples with respect to $<^{[n]}$. Now it is easy to define, using $+^{[n]}$, the corresponding built-in addition.

Then, from $\mathbb{M}$ we can compute $\varphi_{|\mathbb{M}|}$ and $\varphi^I_{|\mathbb{M}|}$ such that $(1^n, \mathbb{M})^{<,+} \models \varphi_{|\mathbb{M}|}$ if and only if $([n], <^{[n]}, +^{[n]}) \models \varphi^I_{|\mathbb{M}|}$, and thus,

$$(1^n, \mathbb{M}) \in p\text{-}\textsc{Halt} \iff \big([n], <^{[n]}, +^{[n]}\big) \models \varphi^I_{|\mathbb{M}|}. \tag{7}$$

By the Ginsburg-Spanier [14] improvement of Presburger's Theorem we know that for $\varphi^I_{|\mathbb{M}|}$ we may compute $n_0, p_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have $\big([n], <^{[n]}, +^{[n]}\big) \models \varphi^I_{\mathbb{M}}$ if and only if $\big([n+p_0], <^{[n+p_0]}, +^{[n+p_0]}\big) \models \varphi^I_{\mathbb{M}}$. By this equivalence and (7) we see that

$$\mathbb{M} \text{ does not hold on empty input tape} \iff \big([n_0], <^{[n_0]}, +^{[n_0]}\big) \models \neg\varphi^I_{\mathbb{M}}.$$

We can decide the halting problem by checking whether $\big([n_0], <^{[n_0]}, +^{[n_0]}\big) \models \neg\varphi^I_{\mathbb{M}}$. ◀

For the proof it was essential that the function assigning to every $k \in \mathbb{N}$ the FO$(<,+)$-sentence φ_k is *computable.* The class obtained if we drop the requirement of computability is called nonuniform-XFO$(<,+)$. We will see that p-Halt $\in$ nonuniform-XFO$(<,+)$ by the even stronger statement of part 1 of Proposition 24, a proposition we prove in the full version of the paper.

We note in passing that by standard modeltheoretic techniques one can show that the parameterized vertex cover problem, a fixed-parameter tractable problem, is not in the subclass nonuniform-XFO$(<)$ of nonuniform-XFO$(<,+)$. Thus we get a lower bound for the parameterized complexity of this problem.

We come back to our claim p-Halt $\in$ nonuniform-XFO$(<,+)$. We even show p-Halt $\in$ nonuniform-para-FO$(<,+)$. By definition, a parameterized probelm (Q,κ) belongs to nonuniform-para-FO$(<,+)$ (to para-FO$(<,+)$) if there are a sentence $\varphi \in$ FO$(<,+)$ and a (computable) function $pre : \mathbb{N} \to \Sigma^*$ such that for all x,

$$x \in Q \iff \langle (x, pre(\kappa(x))) \rangle^{<,+} \models \varphi.$$

So, in the nonuniform version we allow noncomputable precomputations. Note that para-FO$(<,+) \subseteq$ XFO$(<,+)$ as the role of the precomputation (in the definition of para-FO$(<,+)$) can be taken over by the sentences φ_k (in the definition of XFO$(<,+)$).

▶ **Proposition 24.** ***1.*** p-Halt $\in$ nonuniform-para-FO$(<,+)$.
2. p-Halt $\notin$ nonuniform-para-FO$(<)$.

Let τ be a vocabulary which does not contain the reation symbols $<, +, \times$ and set $\tau_{<,+,\times} := \tau \cup \{<,+,\times\}$. Recall that a $\tau_{<,+,\times}$-structure $\mathcal{A}$ *has built-in addition and built-in multiplication* if $(A, <^{\mathcal{A}}, +^{\mathcal{A}}, \times^{\mathcal{A}})$ is isomorphic to $([|A|], <^{[|A|]}, +^{[|A|]}, \times^{[|A|]})$.

A first-order sentence φ of vocabulary $\tau_{<,+,\times}$, shortly $\varphi \in$ FO$(<,+,\times)$, is *invariant* (more precisely, $<$-*invariant*) if for every τ-structure $\mathcal{A}$ and any expansions $(\mathcal{A}, <_1, +_1, \times_1)$ and $(\mathcal{A}, <_2, +_2, \times_2)$ of $\mathcal{A}$ to structures with built-in addition and multiplication, we have:

$$(\mathcal{A}, <_1, +_1, \times_1) \models \varphi \quad\iff\quad (\mathcal{A}, <_2, +_2, \times_2) \models \varphi.$$

It should be clear what we mean if we say that a $\varphi \in$ FO$(<,+)$ or a $\varphi \in$ FO$(<)$ is *invariant.*

Along the lines of [8, Theorem 10] one can show:

▶ **Proposition 25.** *Assume that p-*Halt *$\in$ XFO$(<,+,\times)$. Let τ be any vocabulary not containing the symbols $<$, $+$, and $\times$. Then there is a computable function F defined on the class of* FO$(<,+,\times)$*-sentences of vocabulary $\tau \cup \{<,+,\times\}$ such that*

- *for every $\varphi \in \mathrm{FO}(<,+,\times)$ the sentence $F(\varphi)$ is invariant;*
- *if φ is an invariant $\mathrm{FO}(<,+,\times)$-sentence, then φ and $F(\varphi)$ are equivalent.*

Thus, $\{F(\varphi) \mid \varphi$ an invariant $\mathrm{FO}(<,+,\times)\}$ is the class of sentences of vocabulary τ of a logic for the invariant fragment of $\mathrm{FO}(<,+,\times)$.

In view of Theorem 23, we tried, without success, to show that for $\mathrm{FO}(<,+)$ there is no computable function F with the properties mentioned in the preceding result for $\mathrm{FO}(<,+,\times)$, or even to show that there is no effective enumeration of the invariant sentences of $\mathrm{FO}(<,+,\times)$.

References

1 M. Ajtai. Σ^1_1 formulae on finite structures. *Annals of Pure and Applied Logic*, 24(3):1–48, 1983.

2 N. Alon, M. Krivelevich, and B. Sudakov. Finding a large hidden clique in a random graph. *Random Struct. Algorithms*, 13(3-4):457–466, 1998.

3 M. Bannach, C. Stockhusen, and T. Tantau. Fast parallel fixed-parameter algorithms via color coding. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, pages 224–235, 2015.

4 D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.

5 P. Beame. *A Switching Lemma Primer.* Technical Report, University of Washington, 1984.

6 P. Beame, R. Impagliazzo, and T. Pitassi. Improved depth lower bounds for small distance connectivity. *Computational Complexity*, 7(4):325–345, 1998.

7 J. Chen, X. Huang, I. A. Kanj, and G. Xia. Linear FPT reductions and computational lower bounds. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, STOC 2004, IL, USA, June 13-16, 2004*, pages 212–221, 2004.

8 Y. Chen and J. Flum. A logic for PTIME and a parameterized halting problem. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 397–406, 2009.

9 Y. Chen and J. Flum. From almost optimal algorithms to logics for complexity classes via listings and a halting problem. *Journal of the ACM*, 59(4):17, 2012.

10 Y. Chen, M. Grohe, and M. Grüber. On parameterized approximability. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(106), 2007.

11 M. Elberfeld, C. Stockhusen, and T. Tantau. On the space and circuit complexity of parameterized problems: Classes and completeness. *Algorithmica*, 71(3):661–701, 2015.

12 J. Flum and M. Grohe. Describing parameterized complexity classes. *Information and Computation*, 187(2):291–319, 2003.

13 M. L. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

14 S. Ginsburg and E.H. Spanier. Semigroups, Presburger fomulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

15 M. Grohe, S. Kreutzer, and S. Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th Annual ACM Symposium on the Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 89–98, 2014.

16 Y. Gurevich. Logic and the challenge of computer science. In *Current trends in Theoetical computer Science*, Computer Science Press, pages 1–57, 1988.

17 M. Jerrum. Large cliques elude the metropolis process. *Random Structures and Algorithms*, 3(4):347–360, 1992.

18 L. Kučera. Expected complexity of graph partitioning problems. *Discrete Applied Mathematics*, 57(2-3):193–212, 1995.

19 A. Nash, J. B. Remmel, and V. Vianu. PTIME queries revisited. In *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, volume 3363 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2005.

20 B. Rossman. On the constant-depth complexity of k-clique. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, STOC 2008, Victoria, British Columbia, Canada*, pages 721–730, 2008.

Space-Efficient Approximation Scheme for Maximum Matching in Sparse Graphs*

Samir Datta[1], Raghav Kulkarni[2], and Anish Mukherjee[3]

1 **Chennai Mathematical Institute, India**
`sdatta@cmi.ac.in`
2 **Chennai Mathematical Institute, India**
`kulraghav@gmail.com`
3 **Chennai Mathematical Institute, India**
`anish@cmi.ac.in`

Abstract

We present a Logspace Approximation Scheme (LSAS), i.e. an approximation algorithm for maximum matching in planar graphs (not necessarily bipartite) that achieves an approximation ratio arbitrarily close to one, using only logarithmic space. This deviates from the well known Baker's approach for approximation in planar graphs by avoiding the use of distance computation - which is not known to be in Logspace. Our algorithm actually works for any "recursively sparse" graph class which contains a linear size matching and also for certain other classes like bounded genus graphs.

The scheme is based on an LSAS in bounded degree graphs which are not known to be amenable to Baker's method. We solve the bounded degree case by parallel augmentation of short augmenting paths. Finding a large number of such disjoint paths can, in turn, be reduced to finding a large independent set in a bounded degree graph. The bounded degree assumption allows us to obtain a Logspace algorithm.

1 Introduction

Historically, matching problems have played a central role in Algorithms and Complexity Theory. Edmond's blossom algorithm [14] for maximum matching was one of the first examples of a non-trivial polynomial time algorithm. It had a considerable share in initiating the study of efficient computation, including the class P itself; Valiant's #P-hardness [32] for counting perfect matchings in bipartite graphs provided surprising insights into the counting complexity classes. The rich combinatorial structure of matching problems combined with their potential to serve as central problems in the field invites their study from several perspectives.

The study of whether matching is parallelizable has yielded powerful tools, such as the isolating lemma [27], that have found numerous other applications. The RNC bound remains the best known parallel complexity for maximum matching till date. The best known upper bound for Perfect-Matching is non-uniform SPL[1] whereas the best hardness known is NL-hardness [8].

Matching in Planar Graphs

A well known example where planarity is a boon is that of counting perfect matchings. The problem in planar graphs is in P [21] and can in fact be done in NC[33]; thus Perfect-Matching (Decision) in planar graphs is in NC. "Is the construction version of Perfect-Matching in planar graphs in NC ?" remains an outstanding open question, whereas the bipartite planar case is known to be in NC [26, 25, 23, 11].

* The third author was partially supported by a TCS PhD fellowship. The first and the third authors were partially funded by a grant from Infosys foundation.

The space complexity of matching problems in planar graphs was first studied by Datta, Kulkarni, and Roy [11] where it is shown that minimum weight Perfect-Matching (Min-Wt-PM) in bipartite planar graphs is in SPL. Computing a maximum matching for bipartite planar graphs is shown to be in NC by Hoang [16]. Kulkarni [22] shows that Min-Wt-PM in planar graphs (not necessarily bipartite) is NL-hard. The only known hardness for Perfect-Matching in planar graphs is L-hardness (cf. [10]).

1.1 Motivation

Time efficient approximation algorithms are well studied and have a lot of applications. Space is arguably the second most important resource other than time. Although there is an abundance of work on time efficient approximation, work on space efficient approximation seems limited. To the best of our knowledge even some basic problems such as maximum matching have not been considered. Notice that for (the construction version of) this well studied problem we know of no better complexity bound than $\mathsf{P} \cap \mathsf{RNC}$ [14, 27, 20] even in the planar case. In particular we do not know if it is in SC or NC.

Bounded space approximation algorithms in the presence of non-determinism can be obtained by using Baker's approach [4] for some problems in certain sparse graphs, the most prominent being planar graphs. Dispensing with non-determinism in algorithms even for reachability (not to say matching) leads to either a quasipolynomial blow-up in the time requirement via Savitch's theorem [30] or a large space footprint ($O(\sqrt{n})$) if we want to simultaneously keep the algorithms in polynomial time (see e.g. [19, 3] for reachability in planar graphs). For general graphs the tradeoff at the low space side is even worse, with $O(\frac{n}{2^{\sqrt{\log n}}})$ space and polynomial time [5].

In the context of simultaneous polylogarithmic space and polynomial time (i.e. the class SC), Logspace is the gold-standard and therefore a Logspace Approximation Scheme is the desired result we are able to achieve for planar graphs. An LSAS for bounded degree graphs and a plethora of related graph classes is a serendipitous side effect.

1.2 Previous Work

The problem of approximating maximum matching has been considered both in time and parallel complexity model. [13] gives a linear-time approximation scheme for maximum matching which has the best known time complexity. An NC approximation scheme for maximum matching is given in [18].

Two papers [31, 36] have strived to rephrase Logspace approximation algorithms in the general approximation framework. Their well directed efforts need to be augmented with more concrete problems.

In this direction [9] studied planar MaxCut and related problems in the context of approximation but had to be satisfied with a $\mathsf{UL} \cap \mathsf{co}\text{-}\mathsf{UL}$ approximation scheme which closely follows Baker's approach and is unsatisfactory since it uses non-determinism.

The folklore randomized algorithm for a 1/2-approximation to MaxCut and which can be derandomized in L, with the help of pair-wise independence, is another example in the same spirit.

1.3 Our Results

In this work we first show that there is a Logspace Approximation Scheme for maximum matching in bounded degree graphs.

▶ **Theorem 1.** *Let G be a graph with degrees bounded by a constant d then for any fixed $\epsilon > 0$, we can find a $(1 - \epsilon)$ factor approximation to the maximum matching in Logspace.*

The main fact we use here is that any bounded degree graphs (assuming it's connected) always contains a linear size matching. Many planar graph classes are known to have the property of containing a large matching. Such classes include 3−connected planar graphs [7]. In fact our algorithm works for any *recursively sparse* graph containing a large matching.

Next we show that we can actually give Logspace Approximation Scheme for maximum matching in any planar graph by reducing it to the bounded degree graphs by suitable modifications.

▶ **Theorem 2.** *Let G be a planar graph then for any fixed $\epsilon > 0$, we can find a $(1-\epsilon)$ factor approximation to the maximum matching in Logspace.*

This result extends to many other graph classes, namely for classes which are "biparted" i.e. sparse graphs with bipartite graphs in the class being even *significantly sparser* such as: in bounded genus graphs, k-Apex graphs, (g, k)-Apex graphs, 1-planar graphs, k-page graphs.

Notice that while some of our ideas are similar to the classical sequential algorithm of Hopcroft and Karp [17] for maximum matching in bipartite graphs, we consider graphs which are not necessarily bipartite. Our algorithm trades off Logspace and non-bipartiteness for approximation and sparsity.

1.4 Our Techniques

The primary algorithmic tool is augmentation along short augmenting paths. We prove that in a bounded degree graph, if there are many unmatched but matchable vertices remaining there exist precisely linearly many short augmenting paths. We need to pick a large subset of independent ones from these.

This prompts us to find a large independent set in a bounded degree graph *that works in Logspace*. Notice that the simple greedy strategy that removes a least degree vertex and its neighbourhood will find a linear sized independent set but the algorithm is not implementable in Logspace.

The above method needs the graph to be bounded degree. To convert a planar graph to a bounded degree graph we simply delete high degree vertices and show that this does not affect the size of the matching considerably since the number of high degree vertices is small though possibly still linear in the graph size. This will work if we are sure that the size of the maximum matching is at least linear.

Next we work to whittle the graph down to one containing a linear sized matching without reducing the matching size. We show that removing some small number of vertices ensures this. The proof of this part is based on a lengthy case analysis.

1.5 Organization

After some preliminaries in Section 2, we describe in Section 3 the approximation algorithm for bounded degree graphs where they contain a large (linear in the number of vertices) matching. In Section 4 we then show that our algorithm can be extended for planar graphs also. We conclude in Section 5 with some open ends.

2 Preliminaries

A graph $G = (V, E)$ consists of a finite set of vertices $V(G) = V$ and edges $E(G) = E \subseteq V \times V$.

The class L is the class of languages accepted by deterministic logspace Turing machines. We know that undirected graph connectivity is in L[29]. For the definition of other complexity classes we refer the reader to any standard text book, for example [34, 2]. The concept of Logspace transducer is implicit in Definition 4.16 of [2] and is made explicit in Exercise 4.8 from the same text.

A matching in G is a set $M \subseteq E$, such that no two edges in M have a vertex in common. A matching M is called *perfect* if M covers all vertices of G, M of maximum size is called *maximum* matching. Vertices not incident to an M edge are *free*. An alternating path is one whose edges alternate between M and $E \setminus M$. An alternating path P is augmenting if P begins and ends at free vertices, that is, $M \oplus P = (M \setminus P) \cup (P \setminus M)$ is a matching with cardinality $|M \oplus P| = |M| + 1$. For a complete treatment on matching see [24].

An *independent set* is a set of vertices in a graph, no two of which are adjacent. A *maximum* independent set is an independent set of largest possible size in a given graph. A (vertex) colouring of a graph is an assignment of labels (called "colours") to the vertices of a graph such that no two adjacent vertices share the same color.

An *induced* subgraph of a graph is another graph, formed from a subset of the vertices of the graph and all of the edges connecting pairs of vertices in that subset. An *induced path* is a path that is an induced subgraph. A graph is called *recursively sparse* if every subgraph of it is a sparse graph.

▶ **Definition 3** (Approximation Ratio)**.** We call an algorithm A a β-approximation algorithm if, on every instance I, the algorithm outputs a set I_A such that $1/\beta \cdot I_A \leq I_{Opt} \leq \beta \cdot I_A$ where I_{Opt} is the optimal result on the instance I. The β is called the approximation ratio (or approximation factor) of the algorithm.

▶ **Definition 4** (Approximation Scheme)**.** Let X be a minimization (respectively, maximization) problem.

- An approximation scheme is a family of $(1+\epsilon)$-approximation algorithms A_ϵ (respectively, $(1-\epsilon)$-approximation algorithms A_ϵ) for problem X for any $0 < \epsilon < 1$.
- A Logspace approximation scheme (LSAS) for problem X is an approximation scheme which runs in Logspace.

For a more general treatment of LSAS, consult [36, 31].

A planar graph is a graph that can be embedded in the plane, i.e., the edges can be drawn on the plane in such a way that no edges cross each other (i.e.the edges intersect only at their endpoints). A graph G is said to have *genus* g if G has a minimal embedding (an embedding where every face of G is homeomorphic to a disc) on a genus g surface. Euler's formula for a genus g graph states that $\chi(g) = |V| - |E| + |F|$ where $\chi(g) = 2 - 2g$ and $|F|$ is the number of faces of G. For planar graphs, this implies $|E| \leq 3n - 6$ and so the *average degree* of a planar graph is at most 6. See standard texts on Graph theory (e.g. [12, 35]) for further information. Consult [28] for definitions and properties of various other sparse graph classes.

3 Approximating maximum matching in bounded degree graphs

In this section we show that given any bounded degree graph, we can give a Logspace approximation scheme for the maximum matching.

Our strategy is to design a Logspace transducer that takes in a bounded degree graph and a matching therein as input and while the matching has size significantly smaller than the size of the maximum matching finds a number of disjoint augmenting paths that can then be augmented in parallel in Logspace. The output of the transducer is thus a somewhat larger matching - in fact a matching which is larger than the previous matching by a constant fraction of the maximum matching. We are of course assuming that we are not already very close to the optimal matching. Since we can compose constantly many Logspace transducers to yield another Logspace transducer we are done.

All the augmenting paths we deal with are short i.e. of length at most $2k+1$ for some constant k. This is because such paths can be found in Logspace by say exhaustively listing all $(2k+1)$-tuples of vertices and checking if they form valid augmenting paths.

These short augmenting paths are at most linearly many in n, at most $n(2k+1)^2d^{2k+1}$ to be more precise where d is an upper bound on the maximum degree. Now suppose that the current matching cardinality differs significantly from the maximum matching size $|M_{opt}|$ (by a factor $\Omega(1/k)$ of the maximum matching) then we show that there are at least $\Omega(|M_{opt}|/k)$ many augmenting paths of length $2k+1$ (which happen to be disjoint - though this fact is not used subsequently).

Having demonstrated that there exist lots of paths, we have to find a large fraction in Logspace, which are mutually disjoint. If we form an intersection graph of these short augmenting paths by making two paths adjacent if they have a vertex in common, then we are looking for a large independence set in this intersection graph. We would be done if we can colour the paths with $O(1)$ colours (so that no two intersecting paths get the same colour) because then the largest colour class serves as the desired constant fraction independence set. Since the original graph is bounded degree so is the intersection graph - so it is, at least existentially, $O(1)$-colourable. We in fact show how to constant colour this graph in Logspace.

3.1 Lower bounding the number of short paths

Let $G = (V, E)$ (where $n = |V|$) be the given bounded degree graph with an upper bound of d on the degrees. Let M_{opt} be an optimal maximum matching contained in G. Let M be any other matching which is not necessarily maximum. We assume that the gap $|M_{opt}| - |M|$ is sufficiently large so that lot of augmenting paths exist since the number of unmatched but matchable vertices is large. Yet conceivably very few or none of these paths may be short. Because we can only hope to explore augmenting paths of a constant length in L such a possibility would be very injurious to the approach. Fortunately, we can show that as long as we are not very close to the maximum matching there are *many* short augmenting paths that survive. The following lemma is an adaptation of Corollary 2 of [17] tailored for augmenting paths of constant length where the number of such paths is also important to us.

▶ **Lemma 5.** *If $|M| < (1-\frac{3}{k})|M_{opt}|$ for some positive integer k then there are at least $3|M_{opt}|/2k$ augmenting paths consisting of at most $2k+1$ edges.*

Proof. The maximum number of vertices that can be matched in any matching is precisely, $2|M_{opt}|$. The symmetric difference $M \oplus M_{opt}$ consists of $|M_{opt}| - |M|$ augmenting paths and a number of alternating cycles, which are all mutually disjoint. Suppose the length of the i-th augmenting path is ℓ_i. Then $\sum_{i=1}^{|M_{opt}|-|M|}(\ell_i - 1) \leq 2|M_{opt}|$. This is because an augmenting path of length ℓ contains $\ell - 1$ matched vertices which are distinct across other paths. Thus, $(|M_{opt}| - |M|)\ell_{avg} \leq 3|M_{opt}| - |M| \leq 3|M_{opt}|$ where ℓ_{avg} is the average path length. Thus, $\ell_{avg} \leq k$.

Since at least half fraction of the paths have length at most double the average, we get that at least $3|M_{opt}|/2k$ paths have length at most $2k$. ◀

3.2 Approximating Maximum Independent Set

As graph G still contains a large set of (linearly many) disjoint augmenting paths, we find a constant factor approximation to the maximum independent set in intersection graph of bounded length augmenting paths of G.

Let H be the intersection graph of augmenting paths of length at most $2k+1$ in G.

▶ **Lemma 6.** *A β-factor approximation to the maximum independent set in the graph H can be computed in L where $\beta = 2^{-(2k+1)^2d^{2k+1}}$*

Proof. The graph H has maximum degree upper bounded by $D = (2k+1)^2 d^{2k+1}$ since there are at most $d^i d^{2k+1-i} = d^{2k+1}$ paths in which a fixed vertex appears as the i-th vertex[1]. Since d, k are constants D is also a constant. If we can colour the intersection graph by at most $f(D)$ colours then we would be done because the largest colour class will be a constant (say β) factor approximation to the maximum independent set. Now we give a simple procedure to do this.

For a graph with maximum degree bounded by D, we can find at most D disjoint forests that partition the edge set. This can be done by running Reingold's algorithm for undirected connectivity [29] at most D times on the graph. Now we colour each forest with 2 colours and it gives D bit colours to every node (1 bit for every colouring). This yields an $f(D) = 2^D$ colouring of the graph because two vertices that are adjacent must belong to at least one common forest. ◀

▶ **Theorem 7.** *Let G be a graph with degrees bounded by a constant d then for any fixed $\epsilon > 0$, we can find a $(1-\epsilon)$ factor approximation to the maximum matching in Logspace.*

Proof. Fix integer $k = \lceil \frac{3}{\epsilon} \rceil$. If the current matching is of size at most $(1-3/k)$ fraction of the maximum matching there are a lot (at least $|M_{opt}|/2k$ from Lemma 5) of augmenting paths of length $2k+1$ remaining. Thus the number of vertices in H is at least linear in $|M_{opt}|$.

By Lemma 6 we can find an independent set of size at least $\beta|V(H)| = \beta|M_{opt}|/2k$. This yields a linear number in the size of the maximum matching, of short (length $\leq 2k+1$) augmenting paths which are vertex disjoint and thus are augmentable in parallel. In fact a L-transducer can do the augmentation and output the new matching (it just has to interchange the matched and the unmatched edges in every picked augmenting path).

At every step we increase the matching size by an additive term of $|M_{opt}|/(2k/\beta)$ (unless we get closer than a factor of $(1-3/k)$ to the maximum matching). We chain $(1-3/k)2k/\beta$ such transducers. Note that since we start with an empty matching, after K rounds the approximation ratio would be at least $(1-3/k)$. Thus we get an approximation ratio of at least $(1-3/k) \leq 1-\epsilon$. ◀

4 Approximating Planar Maximum Matching

In this section we show that we can give Logspace Approximation Scheme for finding maximum matching in planar graphs using the LSAS for bounded degree graphs. We first show that a *tame* graph and so a minimum degree 3 planar graph contains a linear size matching in Subsection 4.1. In Subsection 4.2 we describe the Logspace Approximation Scheme.

4.1 Existence of a linear matching subgraph

We say that a maximal induced path is k-isolated if its length is $k > 1$ edges and each of its $(k-1)$-internal vertices have degree precisely two in G. A k-isolated path is long if $k > 2$. An endpoint of an isolated path is called a branch vertex if its degree is G is at least 3 and a pendant vertex if its degree is 1.

Consider the set P of all isolated paths in a graph G. Let P_0 represent the paths in P which contain an even number of edges. Let B_0 represent the set of pairs of endpoints of all the paths in P_0 which support at least two paths from P_0. For each pair in B_0 pick exactly two paths from P_0 supported by vertices of B_0 to yield set P'_0. Let E_0 be the set of extreme[2] edges of all paths in $P_0 \setminus P'_0$.

▶ **Definition 8.** A graph is *tame* if all pairs in B_0 support exactly two paths from P_0.

[1] This is a very crude upper bound which does not take into account that the $2k+1$ length length path is augmenting so the bound of $k^2 d^k$ is closer to truth. Our bound however suffices for the purpose at hand

[2] i.e. the first and the last

We can use the following Lemma to compress the graph preserving maximum matching size:

▶ **Lemma 9.** *The size of the maximum matching in $G - E_0$ is the same as in G.*

Proof. Every matching in $G - E_0$ is a matching in G. Thus we just need to prove that for any maximum matching in G there is a matching in in $G - E_0$ of the same cardinality. To see this notice that for any pair $\{u, v\} \in B_0$ in any matching M it is the case that $0, 1$ or 2 edges from E_0 are used. If 1 or 2 edges of E_0 are used in the matching then 1 or 2 paths, respectively, in P'_0 which are incident on u, v have at least one unmatched vertex (because they contain odd number of vertices apart from their externally matched endpoints). Switching the $1, 2$ matched edges incident on u, v to these $1, 2$ paths in P'_0 so that the unmatched vertices on these paths are matched we reach a matching with the same cardinality as M. ◀

Notice that for a tame graph there may be zero, one or two isolated even length paths between any pair of vertices. Removing the edges in E_0 ensures that we are left with a tame graph. The following is the property of tame graphs that we plan to exploit:

▶ **Lemma 10.** *A tame planar graph has a linear sized maximum matching.*

Proof. Let N_0 be a yet to be fixed threshold[3]. We use a case analysis:

1. The total length of long isolated paths $N \geq N_0$. We have a matching of size at least $N_0/4$ in this case by Lemma 11.
2. The total length of long isolated paths $N < N_0$: In this case for every pair of endpoints of long paths.
   a. We replace each such long path by a path of length 2 or 3 depending on whether the path was even and odd. This reduces the max matching size by at most $N/2$ without increasing the number of vertices.
   b. If there are more than 2 paths of length 3 between u, v then delete all but 2. This further reduces the max matching size by at most 2ν without increasing the number of vertices. Here ν is the number of odd paths in the initial graph. Thus the loss in matching in this step is at most $2N/3$.
   c. Attach the Lollipop graph (i.e. a K_4 with a pendant edge attached to one of the vertices) to each of the 2 internal vertices of the 3-isolated paths. This does not decrease the matching size. The number of vertices goes up by at most $4N$. In the resulting graph only 2-isolated paths have degree 2 vertices.
      i. If there are at least $N' \geq N'_0$ isolated 2-paths in the graph.
         A. Consider the subgraph of this graph where all edges not incident on vertices of degree 2 have been deleted and all isolated vertices formed as a result have been deleted. The resulting subgraph has at least $2N'$ edges and N' (degree 2) vertices.
         B. Find a spanning forest of this graph and root every tree in the forest at a vertex which wasn't a degree 2 vertex in G. It is easy to see that all the vertices of degree 2 in G are matchable in the forest - just match them to their unique child in the rooted forest. Thus a matching size of $N' \geq N'_0$ is guaranteed in G.
      ii. If there are at most $N' < N'_0$ isolated 2-paths in the graph.
         A. Attach the Lollipop graph to each degree 2 vertex of the graph. This does not decrease the matching size and increases the number of vertices by at most $4N'$. We obtain a min degree 3 graph.

[3] which will turn out to be $n/35$

Thus we have a matching of size at least $\min(N_0/4, N_0', m - (N_0/2 + 2N_0/3))$ and the number of vertices is at most $n + 4N_0 + 4N_0'$ in the last case of the minimum. Since the ratio of matching edges and vertices cannot be better than 1/140 from Lemma 12, we just need to assume that: $N_0/4 \geq n/140, N_0' \geq n/140, (m - 5N_0/6) \geq (n + 4N_0 + 4N_0')/140$. Taking $N_0 = 4n/140 = n/35$ and $N_0' = n/140$, we get:

$$m - n/42 \geq (n + n/7)/140$$

or

$$m \geq n/42 + n/140 + n/980 > n/140.$$

Thus, overall, $m \geq n/140$. ◀

▶ **Lemma 11.** *A graph in which the total length of long isolated paths is N has a matching of size at least $N/4$.*

Proof. Let the sum of lengths, number of odd, even isolated paths be denoted by respectively N_{odd}, N_{even} and ν_{odd}, ν_{even} An isolated path of odd length N_i has a matching of size at least $(N_i - 1)/2$ (leaving out the two extreme edges). Similarly, an even length isolated path has a matching of size at least $N_i/2 - 1$. Thus the size of a matching from long odd isolated paths is at least $N_{odd}/2 - \nu_{odd}/2$ and from even isolated paths is at least $N_{even}/2 - \nu_{even}$. Now each long even isolated path has length at least 4 so $4\nu_{even} \leq N_{even}$ and each long odd isolated path has length at least 3 so that $3\nu_{odd} \leq N_{odd}$. Thus the total size of matchings is at least

$$\sum_i N_i/2 - \nu_{odd}/2 - \nu_{even} \geq N/2 - N_{odd}/6 - N_{even}/4 \geq N/2 - N_{odd}/4 - N_{even}/4 = N/4$$

◀

▶ **Lemma 12.** *A minimum degree 3 planar graph has a matching of size at least $n/140$.*

Proof. Consider the set S of all vertices of degree at least d in G. Let S_0 be the isolated vertices in $G - S$ i.e. those vertices in $V(G) - S$ all whose neighbours are in S. Consider the bipartite graph G' with bipartitions S_0, S where we connect a vertex $u \in S_0$ to all $v \in S$ such that $(u, v) \in E(G)$. Now the number of edges incident on S_0 is at least $3|S_0|$ (because every edge incident on $u \in S_0$ is still present in G'). On the other hand, the number of edges from average degree is at most $2(|S| + |S_0|)$. Thus $|S_0| \leq 2|S|$. But $|S| \leq 6n/d$. thus together the number of vertices deleted is at most $3|S| = 18n/d$. Hence the number of remaining vertices is at least $(1 - 18/d)n$.

Now suppose the graph has c components. Find a spanning forest of this graph. Vertices in each spanning tree have degree at most $d - 1$. Then,

▶ Claim 1. Any tree on n vertices and maximum degree d supports a matching of size at least $(n - 1)/d$.

Proof. To see this fix a root to the tree and consider a deepest leaf v in the tree. Remove the other endpoint w of the pendant edge (v, w) leads to a tree containing d lesser vertices. Since at the end we might be left with just the root as an isolated vertex, the claimed bound follows. ◀

Thus the tree supports a matching of size at least $(n'' - 1)/(d - 1)$ where n'' is the number of vertices in the component. Therefore the total size of the matching is at least $(n' - c)/(d - 1)$ where $n' \geq (1 - 18/d)n$ is the number of vertices spanned by the forest. Since none of the components is a singleton it must be that $c \leq n'/2$. So the size of the maximum matching is at least $(1 - 18/d)(1/(2d - 2))n$. Putting $d = 36$, we get that the size of the maximum matching is at least $n/140$. ◀

4.2 Finding a large planar matching

▶ **Theorem 13.** *There is a Logspace Approximation Scheme for maximum matching in planar graphs.*

Proof. We first convert the original graph G into a tame graph G' by using Lemma 9. This preserves the maximum matching size. Suppose there are least αn matching edges in G' for some $\alpha < 1/2$. Fix a positive $\epsilon < \alpha$.

We will delete all vertices of degrees greater than d from G' to yield graph G'' which is of degree bounded by d. Since the number of vertices of degree at least d in G' is at most $6n/d$, the number of matching edges removed by deleting the high degree vertices is at most $6n/d$. So we will have a large $= (\alpha - 6/d)n$ sized matching remaining after this if $\alpha - 6/d = \epsilon/2$ i.e. $d = \frac{12}{2\alpha-\epsilon}$. Thus it suffices to find a $(1 - \epsilon/2)$ factor approximation to the maximum matching using Theorem 7 in Logspace. ◀

Notice that, here we had to tame the graph only to ensure the existence of a linear size matching. But given promise that the graph contains a linear size matching, we can get a approximation scheme, for any recursively sparse graph, without taming it.

▶ **Corollary 14.** *There is a Logspace Approximation Scheme for maximum matching in recursively graphs which contains a linear size matching.*

▶ Note 1. We require only the following properties of planar graphs in proving Lemma 12:

- Sparsity: The average degree is upper bounded by 6.
- Bipartite sparsity: The average degree of every bipartite subgraph is even lower i.e. 4.
- Min-degree: The minimum degree is at least 3 i.e. at least half the average degree.

Thus the proof of Lemma 12 goes through for any family of graphs satisfying these properties. Also notice that Lemma 9 works for arbitrary graphs and Lemma 10 works for any family of graphs satisfying the first two properties above. Hence we also get Logspace Approximation Schemes for the following families of graphs [15]:

1. Genus g graphs: graphs that are embeddable on a surface of genus $g = O(1)$.
2. k-Apex graphs: graphs such that deleting k vertices leads to planar graphs.
3. (g, k)-Apex graphs: graphs such that deleting k vertices leads to genus g graphs.
4. 1-planar graphs: graphs that can be drawn with at most one crossing per edge.
5. k-page graphs: graphs such that all edges can be accommodated on a k-page book with vertices on the spine.

4.3 The Algorithm

Here we present the full algorithm for finding the approximate maximum matching. First we present the algorithms for finding the approximate maximum matching in bounded degree graphs in Algorithm 1 and then we present our main algorithm, describing the *taming* procedure and using the previous algorithm as subroutine, in Algorithm 2.

5 Conclusion and Open-Ends

The main open question which remains is to show that whether we can devise an LSAS for maximum matching in *general* graphs or at least in arbitrary sparse graphs. In this work, we have been able to resolve this for bounded degree graphs, planar graphs and some related classes of sparse graphs.

Algorithm 1 (Matching in Bounded Degree Graphs)

Input : (G, ϵ, M) where $G = (V, E)$ is bounded degree graph with $deg(v) \leq d$ for all $v \in V, \epsilon > 0$ and M is a set of matched edges.
Output : A set $M' \subseteq E$ of matched edges.

```
1: Fix integer $k = \lceil \frac{3}{\epsilon} \rceil$.
2: Construct the intersection graph of augmenting paths of length at most $2k + 1$ in $G$.
3: Let the graph be $H$ with maximum degree $\leq D = (2k + 1)^2 d^{2k+1}$
4: Find at most $D$ disjoint forests that partition the edge set.
5: Colour each forest with 2 colours, giving $D$ bit colours to every node
6: Augment the vertex disjoint augmenting paths in parallel using L-transducer
7: Add the new matching to $M$
8: return $M$
```

Biedl [6] showed that there exists a linear-time (also in Logspace) reduction from maximum matching in arbitrary graphs to maximum matching in 3-regular graphs, though it is not immediate that it is approximation preserving. It will interesting to show such a reduction which is also approximation preserving.

Proving lower bounds for maximum matching in the context of approximation is another important goal. Currently we do not know of any non-trivial hardness results including NC^1-hardness or even TC^0-hardness let alone a L-hardness for approximation to any factor.

Acknowledgements. The first author would like to thank Abhishek Kadian for work on a previous avatar of this paper.

References

1 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

3 Tetsuo Asano, David G. Kirkpatrick, Kotaro Nakagawa, and Osamu Watanabe. $\tilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 45–56, 2014. doi:10.1007/978-3-662-44465-8_5.

4 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.

5 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed *s-t* connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998. doi:10.1137/S0097539793283151.

6 Therese C. Biedl. Linear reductions of maximum matching. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 825–826, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365789.

7 Therese C. Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Mathematics*, 285(1-3):7–15, 2004. doi:10.1016/j.disc.2004.05.003.

8 Ashok K. Chandra, Larry Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984. doi:10.1137/0213028.

Algorithm 2 (Main Algorithm)

```
Input : A graph $G = (V, E)$ and an $\epsilon > 0$.
Output : A set $M \subseteq E$ of matched edges in $G$ such that $|M| \geq (1 - \epsilon)|M_{Opt}|$

1: Let $M = \emptyset$ and $d = 36$
2: Let $P_0$ be the set of all isolated paths containing an even number of edges.
3: Let $B_0$ be the set of pairs of endpoints of all the paths in $P_0$ supporting at least two
   paths from $P_0$.
4: $P'_0 = \emptyset$
5: for each pair $(a, b) \in B_0$ pick exactly two paths from $P_0$ supported by $a, b$ do
6:     Add the two paths to $P'_0$
7: end for
8: Let $E_0$ be the set of extreme edges of all paths in $P_0 \setminus P'_0$.
9: $G = G \setminus E_0$
10: Remove vertices of degree at least $d$
11: Remove all the isolated vertices
12: Let the modified graph be $G'$
13: for $i = 1$ to $(2k - 6)/\beta$ do
14:     Call Algorithm 1 on $G', \epsilon/2$
15:     Remove the matched edges along with endpoints from $G'$
16: end for
17: return $M$
```

9 Samir Datta and Raghav Kulkarni. Space complexity of optimization problems in planar graphs. In *Theory and Applications of Models of Computation - 11th Annual Conference, TAMC 2014, Chennai, India, April 11-13, 2014. Proceedings*, pages 300–311, 2014. doi:10.1007/978-3-319-06089-7_21.

10 Samir Datta, Raghav Kulkarni, Nutan Limaye, and Meena Mahajan. Planarity, determinants, permanents, and (unique) matchings. *ACM Trans. Comput. Theory*, 1(3):1–20, 2010. doi:10.1145/1714450.1714453.

11 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010. doi:10.1007/s00224-009-9204-8.

12 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

13 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014. doi:10.1145/2529989.

14 J. Edmonds. Paths, trees and flowers. *Canad. J. Math.*, 17:449–467, 1965.

15 David Eppstein. Sparser bipartite graphs? Theoretical Computer Science Stack Exchange. URL: http://cstheory.stackexchange.com/q/31567.

16 Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150, 2010. doi:10.1109/CCC.2010.21.

17 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

18 Stefan Hougardy and Doratha E. Drake Vinkemeier. Approximating weighted matchings in parallel. *Inf. Process. Lett.*, 99(3):119–123, 2006. doi:10.1016/j.ipl.2006.03.005.

19 Tatsuya Imai, Kotaro Nakagawa, Aduri Pavan, N. V. Vinodchandran, and Osamu Watanabe. An o(n½+$\sum$)-space and polynomial-time algorithm for directed planar reachability. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 277–286, 2013. doi:10.1109/CCC.2013.35.

20 Richard M. Karp, Eli Upfal, and Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986. doi:10.1007/BF02579407.

21 P. W. Kasteleyn. Graph theory and crystal physics. *Graph Theory and Theoretical Physics*, 1:43–110, 1967.

22 Raghav Kulkarni. On the power of isolation in planar graphs. Technical Report TR09-024, Electronic Colloquium on Computational Complexity, 2009.

23 Raghav Kulkarni, Meena Mahajan, and Kasturi R. Varadarajan. Some perfect matchings and perfect half-integral matchings in NC. *Chicago Journal of Theoretical Computer Science*, 2008(4), September 2008.

24 L. Lovász and M.D. Plummer. *Matching Theory*, volume 29. North-Holland Publishing Co, 1986.

25 Meena Mahajan and Kasturi R. Varadarajan. A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In *STOC*, pages 351–357, 2000.

26 Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995.

27 Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

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

29 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008. doi:10.1145/1391289.1391291.

30 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

31 Till Tantau. Logspace optimization problems and their approximability properties. *Theory Comput. Syst.*, 41(2):327–350, 2007. doi:10.1007/s00224-007-2011-1.

32 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

33 Vijay Vazirani. NC algorithms for computing the number of perfect matchings in $k_{3,3}$–free graphs and related problems. In *Proceedings of SWAT '88*, pages 233–242, 1988.

34 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

35 Douglas B. West. *Introduction to Graph Theory.* Prentice Hall, 2 edition, September 2000.

36 Tomoyuki Yamakami. *Combinatorial Optimization and Applications: 7th International Conference, COCOA 2013, Chengdu, China, December 12-14, 2013, Proceedings*, chapter Uniform-Circuit and Logarithmic-Space Approximations of Refined Combinatorial Optimization Problems, pages 318–329. Springer International Publishing, Cham, 2013. doi:10.1007/978-3-319-03780-6_28.

Logical Characterization of Bisimulation for Transition Relations over Probability Distributions with Internal Actions[*]

Matias David Lee[1] **and Erik P. de Vink**[2]

1 **Univ Lyon, ENS de Lyon, CNRS, UCB Lyon 1, LIP, France and FaMAF, UNC–CONICET, Córdoba, Argentina**
`matias-david.lee@ens-lyon.fr`

2 **Eindhoven University of Technology, The Netherlands and CWI, Amsterdam, The Netherlands**
`evink@win.tue.nl.`

Abstract

In recent years the study of probabilistic transition systems has shifted to transition relations over distributions to allow for a smooth adaptation of the standard non-probabilistic apparatus. In this paper we study transition relations over probability distributions in a setting with internal actions. We provide new logics that characterize probabilistic strong, weak and branching bisimulation. Because these semantics may be considered too strong in the probabilistic context, Eisentraut et al. recently proposed weak distribution bisimulation. To show the flexibility of our approach based on the framework of van Glabbeek for the non-deterministic setting, we provide a novel logical characterization for the latter probabilistic equivalence as well.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.4.1 Mathematical Logic

Keywords and phrases Probabilistic Transition Systems, Weak Bisimulations, Logical Characterization, Transition Relation over Distributions, Modal Logics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.29

1 Introduction

Labeled transition systems (LTS) are a standard way of modeling processes. To verify processes, process theory has embraced two related lines of research, viz. *behavioral equivalences* and *modal logics*. Behavioral equivalences state when two processes present the same behavior. Corresponding minimization algorithms facilitate, e.g., state space reduction. On the other hand, modal logics allow to express extensional properties of processes to be used, for example, in formal specification and verification of systems. In classical cases, a logic characterizes a particular equivalence; two processes are equivalent precisely when they satisfy the same logical formula. In such a situation, when two processes are not equivalent, the logic has a formula that is only satisfied by one of the processes. In a way, the particular formula provides an explanation why the two processes do not have the same behavior.

The introduction of probabilistic transition systems called for an extension of the results known for the non-probabilistic context to the probabilistic one. In [10], Hennessy takes *"a*

[*] M.D. Lee has been supported by EU7FP grant agreement 295261 (MEALS), SECyT–UNC and the project ANR 12IS02001 PACE

fresh look at strong probabilistic bisimulation for processes which exhibit both non-determinism and probabilistic behavior". In his view a process is not in a single state of a probabilistic LTS, but control is spread over a distribution of states. Operationally, this makes sense, because after the execution of an action, a process reaches a set of states with a particular distribution. Thus, transitions of the shape $s \xrightarrow{a} \mu$ are replaced by transitions of the shape $\mu \xrightarrow{a} \mu'$, where s is a state and μ and μ' are distributions over states. This underlies the smooth adaptation of many results for strong bisimulation to probabilistic strong bisimulation in [10].

In this paper we study transitions over distributions in the context of *weak semantics*, i.e. semantics that allow to abstract from internal actions, and the logical characterization of these semantics. We present two concrete enhancements w.r.t. earlier works. First, we do not require processes to be *divergence-free*, i.e. processes may execute infinite τ-actions. Second, we do not focus our attention on weak simulation and bisimulation only; our goal is provide a framework to deal with more general weak semantics. Key for this development is to define what constitutes a transition between two distributions. Following [14, 9], we opt for *hyper-transitions*. Different alternatives and variations of hyper-transitions appear in the literature, e.g. [6, 20, 5, 2]. In [2] also provides a comparison. Given a notion of transitions over distributions, there is a natural way to adapt the relational definition of a semantics from the state-based context to the distributions-based context: just take the standard definition and replace states by distributions.

We discuss *probabilistic strong bisimulation*, *probabilistic weak bisimulation* and *probabilistic branching bisimulation*. For each semantics we propose a logical characterization. These characterizations follow the set-up for the non-deterministic context of van Glabbeek presented in [21]. However, we add in particular the modal operator $[\cdot]_{\geq q}$ to measure probabilities. Because probabilistic weak bisimulation, as one can argue, may be considered too strong, [9] introduces a variant of weak bisimulation, so-called weak distribution bisimulation. We also introduce a logical characterization for this semantics. The peculiarity of this logic is the way probabilities are measured, for which the modal operator $\oplus_q$ is used as introduced in [10]. However, the semantics of $[\cdot]_{\geq q}$ and $\oplus_q$ are rather different: $[\cdot]_{\geq q}$ governs the support of a distribution, while $\oplus_q$ concerns decomposition of a distribution.

Related work on logics for distributions includes [17, 7, 18, 10, 2]. In [17, 2], the idea of transitions over sets of states or distributions does not appear. There, the semantics of the prefix operator depends on the actions that can be executed by the states in the support of the distribution rather than the distribution itself. In addition, in [17] the relational characterization is given for a probabilistic LTS and not for transition relation over distributions. Inspired by the different logics, [2] introduces relations over distributions. On the other hand, [10] takes into account the idea of transitions over set of states but it only focuses on probabilistic strong bisimilarity. Also [7, 18] deal with logics over distributions. The logics introduced in [7] are variants of the probabilistic μ-calculus. Two of these variants, that do not use fix-point operators, characterize weak and strong bisimulation. In [18] only extensions of HML logics are considered. The main difference between [7, 18] and our work lies in the definition of hyper-transitions; in [7, 18], only divergence-free processes are considered. Moreover, they only consider logics to characterize weak (bi)similarity and the approach cannot be uniformly extended other weak semantics as in our case. Specific details on the relation relating to the work mentioned are discussed throughout the paper.

Since the seminal work of [13] on the logical characterization of probabilistic strong bisimulation many extensions have been presented. Work on the logical characterization of weak bisimulation in the probabilistic setting includes [8, 19]. In [8], Desharnais et al. prove that, for the alternating model, PCTL* is sound and complete with respect to weak bisimulation. Song et al. do a similar study taking into account probabilistic automata, see [19]. First, they show that the logic is sound but not complete for strong bisimulation. For this reason, they introduce a variant of the semantics such that PCTL* is complete too. This new semantics relaxes the transfer property: existence of a matching (combined) transition is replaced by existence of a transition of at least the same weight on downward closed sets. Along the same lines a variant for weak bisimulation is obtained. A counterexample in [19] shows that the results for the alternating model in [8] do not hold for probabilistic automata.

The remainder of the paper is organized as follows. In Section 2, following [9], we review *probabilistic automata*, the model used to formalize probabilistic LTS, and transitions between two distributions, so-called *hyper-transitions*, taking into account both visible and internal actions. In Section 3, we introduce the relational characterizations of the various semantics and their logical characterization. Section 4 collects concluding remarks.

2 Preliminaries

For a set X, we denote by $\mathit{SubDisc}(X)$ the set of discrete sub-probability distributions over X. Given $\varrho \in \mathit{SubDisc}(X)$, we denote by $\mathit{spt}(\varrho)$ the support of ϱ, i.e. the set $\{ x \in X \mid \varrho(x) > 0 \}$, by $\varrho(\bot)$ the value $1 - \varrho(X)$, for a distinguished symbol $\bot \notin X$. For $x \in X$, we use $\delta(x)$ to denote the *Dirac* distribution of x given by $\delta(x)(y) = 1$ for $y = x$, 0 otherwise; $\delta_\bot$ represents the empty distribution with $\delta_\bot(X) = 0$. We call a distribution a *probability* distribution if $\varrho(X) = 1$. The set of all discrete probability distributions over X is denoted by $\mathit{Disc}(X)$. Given $\{x_1, \ldots, x_n\} \subseteq X$ and $p_1, \ldots, p_n > 0$ such that $p_1 + \cdots + p_n = 1$, we write $\sum_{i=1}^n p_i x_i$ to denote the distribution that assigns probability p_i to x_i, for $i = 1, \ldots, n$. In addition, given distributions $\mu_1, \ldots, \mu_n \in \mathit{Disc}(X)$, the distribution $\sum_{i=1}^n p_i \mu_i$ is called a convex combination of μ_1 to μ_n. If $n = 2$, we may write $\mu_1 \oplus_p \mu_2$ where $p = p_1$ instead of $p_1\mu_1 + p_2\mu_2$.

We reserve the symbol τ to denote the silent action. For a set X with $\tau \notin X$ we write X_τ for $X \cup \{\tau\}$.

▶ **Definition 1.** A *probabilistic automaton* or PA $\mathcal{A}$ is a tuple (S, Σ_τ, D), where S is the finite set of states, Σ_τ is the set of actions, and $D \subseteq S \times \Sigma_\tau \times \mathit{Disc}(S)$ is the transition relation.

For the rest of the paper we assume that a PA $\mathcal{A} = (S, \Sigma_\tau, D)$ is given. Moreover, $\mathcal{A}$ is *image-finite*, i.e. for all $a \in \Sigma_\tau$ and $s \in S$, the set $\{ \mu \mid (s, a, \mu) \in D \}$ is finite. We write $s \xrightarrow{a} \mu$ for $(s, a, \mu) \in D$. We write $D(a)$ for the set of transitions with label a and $D(s)$ for the set of transitions with source s.

An *execution fragment* $\alpha = s_0 a_1 s_1 a_2 s_2 \ldots$ of $\mathcal{A}$ is a finite or infinite alternating sequence of states and actions such that for each $i > 0$ there exists a transition $(s_{i-1}, a_i, \mu_i) \in D$ with $\mu_i(s_i) > 0$. We say, α is starting from $\mathit{fst}(\alpha) = s_0$, and in case the sequence is finite, ending in $\ell\mathit{st}(\alpha)$. We use $\mathit{frags}(\mathcal{A})$ to denote the set of execution fragments of $\mathcal{A}$, and by $\mathit{ffrags}(\mathcal{A})$ the set of finite execution fragments of $\mathcal{A}$. An execution fragment α is a prefix of an execution fragment α', notation $\alpha \preccurlyeq \alpha'$, if the sequence α is a prefix of the sequence α'. The trace $\mathit{trace}(\alpha)$ of α is the subsequence of non-silent actions of α. We use ε to denote the empty trace. Thus, $\mathit{trace}(a) = a$ for $a \in \Sigma$ and $\mathit{trace}(\tau) = \varepsilon$.

A *scheduler* for $\mathcal{A}$ is a map $\sigma\colon \mathit{ffrags}(\mathcal{A}) \to \mathit{SubDisc}(D)$ with $\sigma(\alpha) \in \mathit{SubDisc}(D(\ell\mathit{st}(\alpha)))$ for every finite execution fragment α. The scheduler is *deterministic* if for every α, $\sigma(\alpha)$ is a Dirac distribution or $\delta_\bot$. Note that by using sub-probability distributions, it is possible that with non-zero probability no transition is chosen after α, that is, the computation stops after α with probability $\sigma(\alpha)(\bot)$. Given a scheduler σ and a finite execution fragment α, the

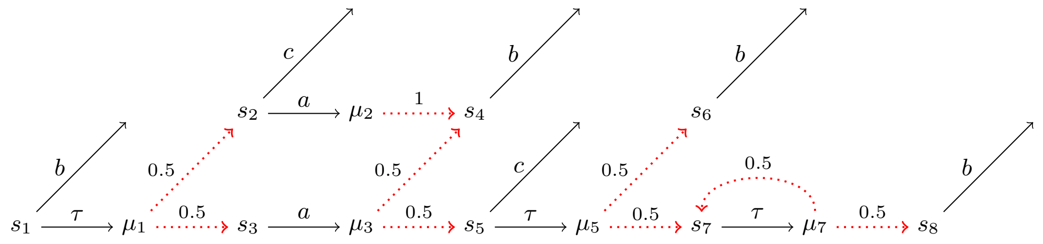


Figure 1 Example probabilistic automaton $\mathcal{A}$.

distribution $\sigma(\alpha)$ describes how transitions are chosen to move on from $\ell st(\alpha)$. A scheduler σ and a state s induce a probability distribution $\mu_{\sigma,s}$ over execution fragments as follows.

The cone C_α of a finite fragment α is the set $\{\, \alpha' \in \mathit{frags}(\mathcal{A}) \mid \alpha \preccurlyeq \alpha' \,\}$. Given a scheduler σ and states s and t, the distribution $\mu_{\sigma,s}$ on cones C_α is recursively defined by

$$\mu_{\sigma,s}(C_t) = \delta(s)(t) \qquad \mu_{\sigma,s}(C_{\alpha a t}) = \mu_{\sigma,s}(C_\alpha) \cdot \textstyle\sum_{\ell st(\alpha) \xrightarrow{a} \mu} \sigma(\alpha)\big(\ell st(\alpha) \xrightarrow{a} \mu\big) \cdot \mu(t)$$

For a finite execution fragment α, the *probability* $\mu_{\sigma,s}(\alpha)$ *of executing* α *(and stop)* based on σ and s is defined as $\mu_{\sigma,s}(\alpha) = \mu_{\sigma,s}(C_\alpha) \cdot \sigma(\alpha)(\perp)$.

A state s can execute a combined weak transition for an action $a \in \Sigma$ if there is a scheduler σ such that with probability 1 the action a is executed once while no other visible action is executed. After a is executed, a state t will be reached with probability $\mu_{\sigma,s}(\{\, \alpha \in \mathit{ffrags}(\mathcal{A}) \mid \ell st(\alpha) = t \,\})$. If $a = \tau$, we have a similar definition but with probability 1 no visible action is executed. Definition 2 takes both cases into account. As usual, $\hat{a} = a$ if $a \in \Sigma$ and $\hat{a} = \varepsilon$ if $a = \tau$

▶ **Definition 2.** Let $s \in S$ and $a \in \Sigma_\tau$. A transition $s \overset{\hat{a}}{\Longrightarrow}_c \mu$ is called a *weak combined transition* if there exists a scheduler σ such that $\mu_{\sigma,s}$ satisfies the following:
1. $\mu_{\sigma,s}(\mathit{ffrags}(\mathcal{A})) = 1$,
2. for each $\alpha \in \mathit{ffrags}(\mathcal{A})$, if $\mu_{\sigma,s}(\alpha) > 0$, then $\mathit{trace}(\alpha) = \mathit{trace}(a)$,
3. for each state t, $\mu_{\sigma,s}(\{\, \alpha \in \mathit{ffrags}(\mathcal{A}) \mid \ell st(\alpha) = t \,\}) = \mu(t)$.

Occasionally we want to make reference to the scheduler σ underlying a weak combined transition $s \overset{\hat{a}}{\Longrightarrow}_c \mu$. We do so by writing $s \overset{\hat{a}}{\Longrightarrow}_\sigma \mu$. For execution fragment α, let $\ell gt(\alpha) = n$ in case $\alpha = s_0 a_1 s_1, \ldots a_n s_n$ is finite, and $\ell gt(\alpha) = \infty$ if α is infinite. We define the length of a scheduler σ with respect to a state s by $\ell gt_s(\sigma) = \sup\{\, \ell gt(\alpha) \mid \mathit{fst}(\alpha) = s, \sigma(\alpha) = \delta_\perp \,\}$.

▶ **Example 3.** Let $\mathcal{A}$ be the PA in Fig. 1. The state s_1 can execute the following weak combined transitions:
(i) $s_1 \overset{\varepsilon}{\Longrightarrow}_c \nu_0$ with $\nu_0(s_1) = 0.5$, $\nu_0(s_2) = 0.25$ and $\nu_0(s_3) = 0.25$;
(ii) $s_1 \overset{a}{\Longrightarrow}_c \nu_1$ with $\nu_1(s_4) = 0.75$ and $\nu_1(s_5) = 0.25$;
(iii) $s_1 \overset{a}{\Longrightarrow}_\sigma \nu_2$ with $\nu_2(s_4) = 0.75$, $\nu_2(s_5) = 0.05$ and $\nu_2(s_6) = \nu_2(s_7) = 0.1$, where σ stops at state s_5 with probability 0.2 and selects the transition $s_5 \xrightarrow{\tau} \mu_5$ with probability 0.8;
(iv) $s_7 \overset{\varepsilon}{\Longrightarrow}_\sigma \delta(s_8)$ where σ is such that $\ell gt(\sigma) = \infty$.

Notice there is no combined transition from s_1 that executes an action c, since from s_1, there is no scheduler that allows to execute this action with probability 1.

A *weak hyper-transition* is a linear combination of weak combined transitions with the same label, see [9]. The weight of each weak combined transition is defined by a distribution μ. The notion of a weak hyper-transition allows to work with transitions over distributions.

▶ **Definition 4.** Given $\mu, \mu' \in \mathit{Disc}(S)$ and $a \in \Sigma_\tau$, there is a *weak hyper-transition* $\mu \overset{\hat{a}}{\Longrightarrow}_c \mu'$ if there exists a family of weak combined transitions $\{s \overset{\hat{a}}{\Longrightarrow}_c \mu_s\}_{s \in spt(\mu)}$ such that $\mu' = \sum_{s \in spt(\mu)} \mu(s) \cdot \mu_s$.

Given a scheduler σ, the scheduler σ_n is such that $\sigma_n(\alpha) = \sigma(\alpha)$ if $\ell gt(\alpha) \leqslant n$, otherwise $\sigma_n(\alpha) = \delta_\perp$. For $a \in \Sigma_\tau$, we write $s \xrightarrow{a}_c \mu$ if there is a scheduler σ of length 1 such that $s \overset{\hat{a}}{\Longrightarrow}_\sigma \mu$. Schedulers of length 1 induce the notion of a *one-step transition*.

▶ **Definition 5.** Let $\mu, \mu' \in \mathit{Disc}(S)$ and $a \in \Sigma_\tau$. For $a \neq \tau$, a *one-step transition* $\mu \xrightarrow{a}_c \mu'$ is a weak hyper-transition $\mu \overset{a}{\Longrightarrow}_c \mu'$ for $\{s \overset{a}{\Longrightarrow}_c \mu_s\}_{s \in spt(\mu)}$ such that $s \xrightarrow{a}_c \mu_s$. A *one-step transition* $\mu \xrightarrow{\tau}_c \mu'$ is a weak hyper-transition $\mu \overset{\varepsilon}{\Longrightarrow}_c \mu'$ for $\{s \overset{\varepsilon}{\Longrightarrow}_c \mu_s\}_{s \in spt(\mu)}$ such that either $\mu_s = \delta(s)$ or $s \xrightarrow{\tau}_c \mu_s$, for $s \in spt(\mu)$, and $s \xrightarrow{\tau}_c \mu_s$ for at least one $s \in spt(\mu)$.

The definition of a one-step transition for a visible action requires that each state in the support of μ executes the visible action. On the other hand, if the action is not visible, we require that at least one state executes a τ-transition.

▶ **Example 6.** Consider again Example 3. Because of $s_1 \xrightarrow{a}_c \nu_1$ and $s_2 \xrightarrow{a}_c \mu_2$, $0.5\delta(s_1) + 0.5\delta(s_2) \xrightarrow{a}_c 0.5\nu_1 + 0.5\mu_2$. Since $s_1 \xrightarrow{\tau}_c 0.5\delta(s_1) + 0.5\mu_1$, we have $\delta(s_1) \xrightarrow{\varepsilon}_c 0.5\delta(s_1) + 0.5\mu_1$. Notice that this target distribution cannot be reached from $\delta(s_1)$ by a one-step transition with a deterministic scheduler. Definition 5 does not allow to split the state s_1. Finally, notice that $\delta(s_1) \overset{\varepsilon}{\Longrightarrow}_c \delta(s_1)$ but $\delta(s_1) \xrightarrow{\tau}_c \delta(s_1)$ is not a valid one-step hyper-transition.

We have used the notion of a hyper-transition of [14] to define transitions over distributions because of its clear operational intuition.

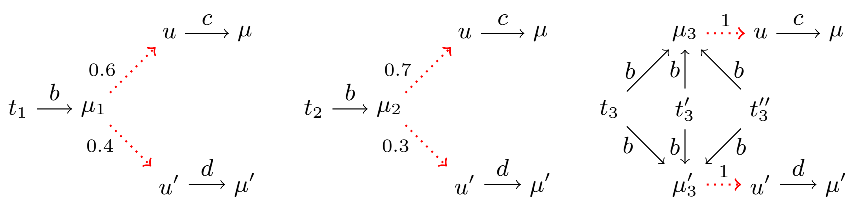
3 Semantics for Transitions Over Distributions and their Logical Characterizations

In this section we present four different semantics. First we treat probabilistic strong bisimulation. Here we present the main results to deal with probabilities. The second semantics is probabilistic weak bisimulation. The key point is how to define a logic that characterizes the relation in such a way that the modal operators can be reused for characterizing other semantics that abstract from internal behavior. For this we follow the framework defined by van Glabbeek in [21]. Additionally, we have to introduce a number of properties satisfied by our definitions of combined and hyper-transitions (Lemma 19). We also recall why probabilistic weak bisimulation may be considered too tight. Next we cover probabilistic branching bisimulation. Defining a logic and proving that it characterizes the process equivalence is straightforward given the definitions and results gathered already. We also provide a stuttering lemma for the probabilistic context. Finally, we discuss weak distribution bisimulation [9] and its logical characterization.

3.1 Probabilistic strong bisimulation

Generally, approaches to define behavioral equivalences for probabilistic transition systems provide a relation over states and a lifting to distributions over states, by means of, for example, *weight functions* [20] or *closed sets* [3]. We follow a different approach and will directly define relations over distributions that satisfy the *decomposability* condition of [10].

▶ **Definition 7.** A symmetric relation $\mathcal{R} \subseteq \mathit{Disc}(S) \times \mathit{Disc}(S)$ is *decomposable* if $\mu \,\mathcal{R}\, \nu$ and $\mu = \mu_1 \oplus_p \mu_2$ imply there are $\nu_1, \nu_2 \in \mathit{Disc}(S)$ s.t. $\nu = \nu_1 \oplus_p \nu_2$, $\mu_1 \,\mathcal{R}\, \nu_1$ and $\mu_2 \,\mathcal{R}\, \nu_2$.



$t_1 \xrightarrow{b} \mu_1$
0.6
0.4
$u \xrightarrow{c} \mu$
$u' \xrightarrow{d} \mu'$
$t_2 \xrightarrow{b} \mu_2$
0.7
0.3
$u \xrightarrow{c} \mu$
$u' \xrightarrow{d} \mu'$
t_3
t'_3
t''_3
μ_3
μ'_3
1
1
b
$u \xrightarrow{c} \mu$
$u' \xrightarrow{d} \mu'$

Figure 2 The decomposability condition is needed – Only considering deterministic schedulers is too strong.

In Definition 8, we introduce the notion of a *probabilistic strong bisimulation* for transition relations over probability distributions. Then we will explain why the decomposability condition is needed. We also show that a variant of strong bisimulation for transitions over distributions that only considers deterministic schedulers is too strong. For this reason we will not consider one-step transitions for deterministic schedulers further.

▶ **Definition 8.** A decomposable relation $\mathcal{R} \subseteq \mathit{Disc}(S) \times \mathit{Disc}(S)$ is called a *probabilistic strong bisimulation* if, for every $a \in \Sigma_\tau$, $\mu \,\mathcal{R}\, \nu$ and $\mu \xrightarrow{a}_c \mu'$ imply $\nu \xrightarrow{a}_c \nu'$ and $\mu' \,\mathcal{R}\, \nu'$ for some ν'. *Probabilistic strong bisimilarity*, notation $\approx_{ps}$, is defined as the union of all probabilistic strong bisimulations.

Figure 2 illustrates the need for the decomposability condition. Suppose we remove the condition, therefore μ_1 and μ_2 should be consider equivalent since they execute no transition, see Def. 5; therefore $\delta(t_1)$ and $\delta(t_2)$ are also equivalent. The condition ensures that distributions that are related by $\approx_{ps}$ assign the same weight to equivalence classes. Note that $\mu_1 = 0.6\delta(u) + 0.4\delta(u')$ and there are no μ'_2 and μ''_2 such that $\mu_2 = 0.6\mu'_2 + 0.4\mu''_2$, $\delta(u) \approx_{ps} \mu'_2$ and $\delta(u') \approx_{ps} \mu''_2$. Thus $\mu_1 \not\approx_{ps} \mu_2$ and therefore $\delta(t_1) \not\approx_{ps} \delta(t_2)$.

We explain the problem with deterministic schedulers. Consider state t_3, t'_3 and t''_3 in Figure 2. It is clear that $\delta(t_3) \approx_{ps} \delta(t'_3)$, $\delta(t'_3) \approx_{ps} \delta(t''_3)$ and $\delta(t_3) \approx_{ps} \delta(t''_3)$. If we would consider only deterministic schedulers to define one-step transitions we have the transition $0.5\delta(t'_3) + 0.5\delta(t''_3) \xrightarrow{b} 0.5\mu_3 + 0.5\mu'_3$ and this transition cannot be mimicked by $\delta(t_3)$ in the restricted setting. However, since we consider arbitrary schedulers, $\delta(t_3) \xrightarrow{b}_c 0.5\mu_3 + 0.5\mu'_3$.

Definition 9 introduces a logic that characterizes $\approx_{ps}$ (Theorem 15).

▶ **Definition 9.** The logic $\mathcal{L}_{ps}$ is defined by

$$\psi := \top \mid \bigwedge_{i\in I} \psi_i \mid \neg\psi \mid a\psi \mid \tau\psi \mid [\psi]_{\geq q}$$

for $a \in \Sigma$, $q \in \mathbb{Q}$ and possibly infinite index sets I. The *satisfiability of an $\mathcal{L}_{ps}$-formula for $\mu \in \mathit{Disc}(S)$* is defined by the following clauses:

$(\top)$	$\mu \models \top$	for all μ	(a)	$\mu \models a\psi$	if $\mu \xrightarrow{a}_c \mu'$ and $\mu' \models \psi$
$(\wedge)$	$\mu \models \bigwedge_{i\in I} \psi_i$	if $\mu \models \psi_i$ for all $i \in I$	(τ)	$\mu \models \tau\psi$	if $\mu \xrightarrow{\tau}_c \mu'$ and $\mu' \models \psi$
$(\neg)$	$\mu \models \neg\psi$	if $\mu \not\models \psi$	$(\geq q)$	$\mu \models [\psi]_{\geq q}$	if $\mu(\{s \in S \mid \delta(s) \models \psi\}) \geqslant q$

We denote by $\models_{ps}$ the satisfiability relation of $\mathcal{L}_{ps}$.

Recall that a one-step transition with an action a different from τ requires that all states in the support of the distribution execute the action a. This is not the case for a τ action. See Definition 5.

In [10], Hennessy introduces the logic pHML that also characterizes probabilistic strong bisimulation. The difference with this logic is the modality used to measure probabilities. The logic pHML uses a modal operator $\oplus_q$ and its validity, in contrast to $[\psi]_{\geq q}$, does not depend on the support of the distribution. Instead, it depends on how the distribution can be decomposed. The quantitative modal operators of [7, 18] are defined similarly. We do not follow this approach because it does not fit well for weak semantics. We explain this in more detail in the next subsection.

Parma and Segala [17] have introduced a distribution-based logic to characterize state-based probabilistic strong bisimulation, the logic $\mathcal{L}_p^N$. In this setting, states s and t are bisimilar iff $\delta(s)$ and $\delta(t)$ satisfy the same set of formulas. If we compare $\models_{ps}$ with $\mathcal{L}_p^N$, we see that clause (a) of Definition 9 is different: they have $\mu \models a\psi$ if for all $s \in spt(\mu)$, $s \xrightarrow{a}_c \mu_s$ and $\mu_s \models \psi$. In this case, the modal operator refers to the transitions that can be executed by the states underlying the distribution instead of the transitions of the distribution itself.

Theorem 15 states that $\mathcal{L}_{ps}$ characterizes $\approx_{ps}$. To prove the theorem we need a number of auxiliary results.

▶ **Lemma 10.** *Let $\mathcal{R}$ be a decomposable relation. For all μ, ν with $\mu \, \mathcal{R} \, \nu$, there is a finite index set K such that*
1. $\mu = \sum_{k \in K} p_k \cdot \delta(s_k)$.
2. $\nu = \sum_{k \in K} p_k \cdot \delta(t_k)$.
3. $\delta(s_k) \, \mathcal{R} \, \delta(t_k)$ *for all* $k \in K$.

In the following three lemmas $\mathcal{L}$ indicates a sublogic (of the logic at hand, here $\mathcal{L}_{ps}$).

▶ **Lemma 11.** *Let $\hat{s} \in S$ and $\psi \in \mathcal{L}$. If $\delta(\hat{s}) \models \psi$ then it holds that $\delta(\hat{s})(\{ \, s \mid \delta(s) \models \psi \, \}) = 1$.*

Note that the last result does not work for arbitrary μ and ψ such that $\mu \models_s \psi$. Referring to Figure 1, $0.5\delta(s_2) + 0.5\delta(s_3) \models_s a[b]_{\geq 0.75}$. However, $\delta(s_3) \not\models_s a[b]_{\geq 0.75}$, and therefore it holds that $0.5\delta(s_2) + 0.5\delta(s_3)(\{ \, s \mid \delta(s) \models \psi \, \}) < 1$.

▶ **Lemma 12.** *Let $\mathcal{L}$ be a logic containing $\neg$ and $\bigwedge$. Then there is a formula ψ_C for each $C \in \{\pi \mid \pi \in Disc(S)$ is a Dirac distribution$\}/{\approx_{\mathcal{L}}}$ s.t. for all $s \in S$, $\delta(s) \models \psi_C$ iff $\delta(s) \in C$.*

▶ **Lemma 13.** *Let $\mathcal{L}$ be a logic containing $\bigwedge$, $\neg$, and $[\cdot]_{\geq q}$ for $q \in \mathbb{Q}$. For all $\mu, \nu \in Disc(S)$ with $\mu \approx_{\mathcal{L}} \nu$ there is a finite index set K such that*
1. $\mu = \sum_{k \in K} p_k \cdot \delta(s_k)$.
2. $\nu = \sum_{k \in K} p_k \cdot \delta(t_k)$.
3. $\delta(s_k) \approx_{\mathcal{L}} \delta(t_k)$ *for all* $k \in K$.

▶ **Lemma 14.** *Suppose $\sum_{k \in K} p_k = 1$ for some index finite set K. If $\mu_k \approx_{\mathcal{L}_{ps}} \nu_k$ for $k \in K$, then $\sum_{k \in K} p_k \mu_k \approx_{\mathcal{L}_{ps}} \sum_{k \in K} p_k \nu_k$.*

▶ **Theorem 15.** *Let $\mu, \nu \in Disc(S)$, then $\mu \approx_{ps} \nu$ iff $\mu \approx_{\mathcal{L}_{ps}} \nu$.*

Sketch. To prove ($\Rightarrow$) we show that $\mu \approx_{ps} \nu$ and $\mu \models_{ps} \psi$ implies $\nu \models_{ps} \psi$. This goes by structural induction on ψ. Cases $\top, \bigwedge_{i \in I} \psi_i$, $\neg\psi$ and $a\psi$ follow [21]. The case $[\psi]_{\geq q}$ follows by Lemmas 10 and 11. To prove ($\Leftarrow$) we show that $\approx_{\mathcal{L}_{ps}}$ is a probabilistic strong bisimulation. The check of the transfer property follows [21]. To prove that $\approx_{\mathcal{L}_{ps}}$ is decomposable we use Lemmas 13 and 14. ◀

3.2 Probabilistic weak bisimulation

Transition relations over distributions allow to introduce straightforwardly a notion of weak bisimulation. Moreover, the discussion of the previous subsection applies here as well. Then, after the definition of weak bisimulation, we can focus on defining a corresponding logic.

▶ **Definition 16.** A decomposable relation $\mathcal{R} \subseteq \mathit{Disc}(S) \times \mathit{Disc}(S)$ is a *probabilistic weak bisimulation* if given $\mu \, \mathcal{R} \, \nu$, for every $a \in \Sigma_\tau$, $\mu \xrightarrow{a}_c \mu'$ implies there is ν' such that $\nu \overset{\hat{a}}{\Longrightarrow}_c \nu'$ and $\mu' \, \mathcal{R} \, \nu'$. *Probabilistic weak bisimilarity*, notation $\approx_{pw}$, is defined as the union of all probabilistic weak bisimulations.

The logic that characterizes $\approx_{pw}$ uses many of the operators of the logic for $\approx_{ps}$, but also adds new features to deal with internal behavior. The logic $\mathcal{L}_{pw}$ will be defined using a new modality ε, and two new clauses, $(\vec{\varepsilon})$ and $(\overleftarrow{\varepsilon})$ in the satisfiability relation. Because internal transitions cannot be observed, the clause (τ) is removed. First, we introduce the syntax and explain the intuition of the modality ε.

▶ **Definition 17.** The *logic* $\mathcal{L}_{pw}$ is defined by

$$\psi := \top \mid \bigwedge_{i \in I} \psi_i \mid \neg\psi \mid a\varepsilon\psi \mid \varepsilon\psi \mid [\psi]_{\geq q}$$

for $a \in \Sigma$, $q \in \mathbb{Q}$ and possibly infinite index sets I.

The modality ε is introduced to encode that internal behavior (zero or more τ's) can happen. In addition, we shall assume that some internal behavior can happen before the execution of any action. For example, $\varepsilon[c\varepsilon\top]_{\geq 0.5}$ encodes that after some internal behavior, with probability at least 0.5, an action c can be executed and the observation terminates; because the assumption and the modality ε, before and after the execution of c, some internal behavior can happen. This behavior is present in state s_1 in Figure 1. Also in Figure 1, notice that states s_2 and s_3 satisfy $a\varepsilon\top$. Because s_1 can reach both states with probability 1 via internal behavior, then $\delta(s_1)$ should also satisfy $a\varepsilon\top$. These ideas are modeled by clauses $(\vec{\varepsilon})$ and $(\overleftarrow{\varepsilon})$, which follow a phrasing of [21], in the following definition.

▶ **Definition 18.** The *satisfiability* of an $\mathcal{L}_{pw}$-formula is defined by the clauses $(\top)$, $(\bigwedge)$, $(\neg)$, $[\cdot]_{\geq q}$, (a) together with the following two:

$(\vec{\varepsilon})$ $\mu \models \varepsilon\psi$ if $\mu \models \psi$. $(\overleftarrow{\varepsilon})$ $\mu \models \psi$ if $\mu \overset{\varepsilon}{\Longrightarrow}_c \mu'$, $\mu' \models \psi$ and the outermost operator of ψ is neither $\neg$, $\bigwedge$ nor $[\cdot]_{\geq q}$.

We write $\models_{pw}$ to denote the satisfiability relation of $\mathcal{L}_{pw}$.

Figures 3 and 4, corresponding to the example in Figure 1, illustrates these clauses. Moreover, Figure 4 shows their interaction. Notice that it is possible to infer that some internal behavior can or cannot happen in a state. For instance, $\delta(s_1) \models \varepsilon[c\varepsilon\top]_{\geq 0.5}$, but $\delta(s_1) \not\models [c\varepsilon\top]_{\geq 0.5}$. The two formulas confirm that an internal transition for s_1 will change the equivalence class of the process for s_1 of Figure 1.

The condition "*the outermost operators of ψ is not $\neg$, $\bigwedge$ nor $[\cdot]_{\geq q}$*" of Definition 18 is needed for $(\overleftarrow{\varepsilon})$ because operators $\neg$ and $[\cdot]_{\geq q}$ give information about the current distribution. We use an example to explain this. See Figure 1: distribution μ_1 is such that $\mu_1 \models_{pw} \neg b\varepsilon\top$. If the restriction is not present, given that $\delta(s_1) \overset{\varepsilon}{\Longrightarrow} \mu_1$, one has $\delta(s_1) \models_{pw} \neg b\varepsilon\top$. This is inconsistent with the fact that $s_1 \xrightarrow{b}$. Similar reasoning is in place for the modality $[\cdot]_{\geq q}$. Operator $\bigwedge$ is also restricted to take into account the recursive case, for example, $\neg b\varepsilon\top \wedge \top$.

1. $\delta(s_2) \models_{pw} a\varepsilon\top$ and $\delta(s_3) \models_{pw} a\varepsilon\top$
2. $0.5\delta(s_2) + 0.5\delta(s_3) \models_{pw} a\varepsilon\top$ by (a)
3. $\delta(s_1) \xRightarrow{\varepsilon}_c 0.5\delta(s_2) + 0.5\delta(s_3)$
4. $\delta(s_1) \models_{pw} a\varepsilon\top$ by $(\overleftarrow{\varepsilon})$ and 3.

Figure 3 $(\overleftarrow{\varepsilon})$ allows to take into account the formula that are satisfied after an internal hyper-transition with a probabilistic scheduler.

(i) $\delta(s_2) \models_{pw} c\varepsilon\top$.
(ii) $0.5\delta(s_2) + 0.5\delta(s_3) \models_{pw} [c\varepsilon\top]_{\geq 0.5}$ by $(\geq q)$
(iii) $0.5\delta(s_2) + 0.5\delta(s_3) \models_{pw} \varepsilon[c\varepsilon\top]_{\geq 0.5}$ by $(\overrightarrow{\varepsilon})$
(iv) $\delta(s_1) \models_{pw} \varepsilon[c\varepsilon\top]_{\geq 0.5}$ by $(\overleftarrow{\varepsilon})$, 3. and 3

Figure 4 $(\overrightarrow{\varepsilon})$ concerns the current probability measure. Then $(\overleftarrow{\varepsilon})$ can be used for backward propagation. Notice $\delta(s_1) \not\models_{pw} [c\varepsilon\top]_{\geq 0.5}$.

In addition, in the following sections, we will use the same definition to define other logics that characterize other weak semantics.

In [17], Parma and Segala also introduce a logic that characterizes probabilistic weak bisimulation, the logic $\mathcal{L}_w^N$. This logic is the logic $\mathcal{L}_p^N$ where the modality a is replaced by a modality that considers weak combined transitions. Similarly, [7, 18] consider weak hyper-transitions. In comparison with other logics characterizing probabilistic weak bisimulation, $\mathcal{L}_{pw}$ looks more complex because of the modalities ε and the clauses $(\overleftarrow{\varepsilon})$ and $(\overrightarrow{\varepsilon})$. This complexity is needed to extend the logic to other semantics. See the next section.

It has been argued that probabilistic weak bisimulation is too strong [9, 5]: Consider Figure 5 and assume that states $\frownie$ and $\smiley$ are such that $\frownie \not\approx_{pw} \smiley$. State s_1 does not add any behavior to the system represented by the state s, and the probabilities of reaching states $\frownie$ and $\smiley$ from s are, respectively, 0.25 and 0.75. Then it is plausible to consider the distributions $\delta(s)$ and $\delta(t)$ weakly bisimilar, but in fact they are not. Notice there is no matching for the transition $\delta(s) \xrightarrow{\tau} \mu_s$. In [9], a variant of weak bisimulation is introduced to deal with this problem. We study the logic characterization of this variant in Subsection 3.4.

We explain why the approach of Hennessy [10] to define the measure modality does not fit well for weak bisimilarity. Let $\psi_{\frownie}$ and $\psi_{\smiley}$ be the characteristic formula of $\frownie$ and $\smiley$. Then

$$\delta(s_1) \models_{pw} \varepsilon([\psi_{\frownie}]_{\geq 0.5} \wedge [\psi_{\smiley}]_{\geq 0.5}) \qquad \delta(s) \models_{pw} \varepsilon([\varepsilon([\psi_{\frownie}]_{\geq 0.5} \wedge [\psi_{\smiley}]_{\geq 0.5}]_{\geq 0.5} \wedge [\psi_{\smiley}]_{\geq 0.5}))$$

Let $\hat{\psi}$ be the last formula, then $\delta(t) \not\models_{pw} \hat{\psi}$. In case we had used the approach used by Hennessy [10], we would replace the measure modality $[\,\cdot\,]_{\geq q}$ by $\oplus_q$ with clause

$$(\oplus) \quad \mu \models \psi_1 \oplus_q \psi_2 \quad \text{if } \mu = \mu_1 \oplus_q \mu_2 \text{ and } \mu_i \models \psi_i \text{ for } i = 1, 2$$

In the new setting, the formula analogous to $\hat{\psi}$ is $\varepsilon((\varepsilon(\psi_{\frownie} \oplus_{0.5} \psi_{\smiley})) \oplus_{0.5} \psi_{\smiley})$. This formula is satisfied by $\delta(t)$ because $\mu_t = (\frownie \oplus_{0.5} \smiley) \oplus_{0.5} \smiley$. Then $\delta(s)$ and $\delta(t)$ would not be distinguished by the logic in the new setting.

Theorem 21 states $\mathcal{L}_{pw}$ characterizes $\approx_{pw}$. To prove the result, we reuse the results for probabilistic strong bisimulation regarding probabilities. In addition, we need to introduce technical properties of weak transitions, see Lemma 19, and to recast Lemma 14 for $\mathcal{L}_{pw}$.

▶ **Lemma 19.** *Let $s \in S$, $\mu, \mu', \nu, \mu_i, \mu_i' \in Disc(S)$, where $i \in I$, and σ be a scheduler. Then*
1. *$\mu \xRightarrow{\varepsilon}_c \nu$ and $\nu \xRightarrow{\varepsilon}_c \mu'$ imply $\mu \xRightarrow{\varepsilon}_c \mu'$. [14, Prop. 3.6]*
2. *$s \xRightarrow{\hat{a}}_c \mu_i$, $i \in I$, $a \in \Sigma_\tau$ and $\sum_{i \in I} p_i = 1$ imply $s \xRightarrow{\hat{a}}_c \sum_{i \in I} p_i\mu_i$ [14, Prop. 3.4].*

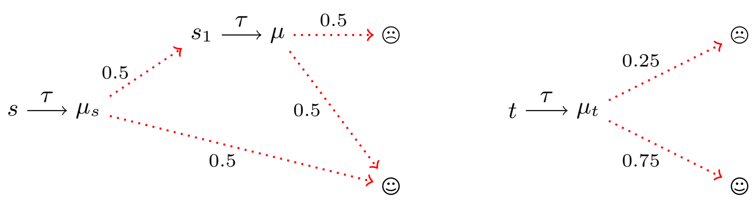


Figure 5 Probabilistic weak bisimulation is too strong.

3. $\mu_i \overset{\hat{a}}{\Longrightarrow}_c \mu_i'$, $i \in I$, $a \in \Sigma_\tau$ *and* $\sum_{i\in I} p_i = 1$ *imply* $\sum_{i\in I} p_i\mu_i \overset{\hat{a}}{\Longrightarrow} \sum_{i\in I} p_i\mu_i'$.
4. *For* $a \in \Sigma$, $\mu \overset{a}{\Longrightarrow}_c \mu'$ *iff there are* $\nu, \nu' \in Disc(S)$ *such that* $\mu \overset{\varepsilon}{\Longrightarrow}_c \nu \xrightarrow{a}_c \nu' \overset{\varepsilon}{\Longrightarrow}_c \mu'$.
5. $\sigma = \lim_{n\to\infty} \sigma_n$ *pointwise. Moreover,* $\mu_{s,\sigma}(\alpha) = \lim_{n\to\infty} \mu_{s,\sigma_n}(\alpha)$ *[20, Prop. 5.3.22].*
6. *If* $\ell gt(\sigma) > n$, $s \overset{\varepsilon}{\Longrightarrow}_{\sigma_n} \mu_n$ *and* $s \overset{\varepsilon}{\Longrightarrow}_{\sigma_{n+1}} \mu_{n+1}$ *then* $\mu_n \xrightarrow{\tau}_c \mu_{n+1}$.
7. *If* $\mu \approx_{pw} \nu$ *and* $\mu \overset{\varepsilon}{\Longrightarrow}_c \mu'$, *there is* $\nu' \in Disc(S)$ *such that* $\nu \overset{\varepsilon}{\Longrightarrow}_c \nu'$ *and* $\mu' \approx_{pw} \nu'$.
8. *For* $a \in \Sigma$, *if* $\mu \approx_{pw} \nu$ *and* $\mu \overset{a}{\Longrightarrow}_c \mu'$, *there is* $\nu' \in Disc(S)$ *such that* $\nu \overset{a}{\Longrightarrow}_c \nu'$ *and* $\mu' \approx_{pw} \nu'$.

▶ **Lemma 20.** *Suppose* $\sum_{k\in K} p_k = 1$, *for some finite index set* K. *If* $\mu_k \approx_{\mathcal{L}_{pw}} \nu_k$ *for* $k \in K$, *then* $\sum_{k\in K} p_k\mu_k \approx_{\mathcal{L}_{pw}} \sum_{k\in K} p_k\nu_k$.

▶ **Theorem 21.** *Let* $\mu, \nu \in Disc(S)$. *Then* $\mu \approx_{pw} \nu$ *iff* $\mu \approx_{\mathcal{L}_{pw}} \nu$.

The proofs of Lemma 20 and Theorem 21 strongly depend on the properties of Lemma 19. The proof of these statements are intricate, because of the definitions of combined and hyper-transitions considering schedulers of infinite length. These chedulers are needed, e.g., to distinguish between distributions $\delta(s_7)$ and $\delta(s_8)$ in Figure 1, i.e. $\delta(s_7) \approx_{pw} \delta(s_8)$. Note, state s_8 executes a transition with action b and this transition can be mimicked by s_7 only using a weak hyper-transition defined by a scheduler of infinite length. The variant of weak bisimulation of [7, 18] does not relate $\delta(s_7)$ and $\delta(s_8)$, because the relation $\overset{\varepsilon}{\Longrightarrow}_c$ is defined as the reflexive and transitive closure of $\xrightarrow{\hat{\tau}}_c$. This way of defining hyper-transition is sufficient in the context of [7, 18], because they deal with divergence-free PA. Notice, $\mathcal{A}$ in Figure 1 is not divergence-free.

3.3 Probabilistic branching bisimulation

We discuss probabilistic branching bisimilarity and its logical characterization. We remark that for the correspondence result we only need to add one new auxiliary result, a lemma analogous to Lemma 20.

▶ **Definition 22.** A decomposable relation $\mathcal{R} \subseteq Disc(S) \times Disc(S)$ is a *probabilistic branching bisimulation* if given $\mu \,\mathcal{R}\, \nu$, for every $a \in \Sigma_\tau$, $\mu \xrightarrow{a}_c \mu'$ implies

- $a = \tau$ and $\mu' \,\mathcal{R}\, \nu$, or
- there are $\tilde{\nu}$ and ν' such that $\nu \overset{\varepsilon}{\Longrightarrow}_c \tilde{\nu} \xrightarrow{a}_c \nu'$ with $\mu \,\mathcal{R}\, \tilde{\nu}$ and $\mu' \,\mathcal{R}\, \nu'$.

Probabilistic branching bisimilarity, notation $\approx_{pb}$, is defined as the union of all probabilistic branching bisimulations.

For the logic $\mathcal{L}_{pb}$ for probabilistic branching bisimulation we include binary operators $\_a\_$, for $a \in \Sigma$, and $\_\tau\_$ replacing $a\varepsilon$ and ε.

▶ **Definition 23.** The *logic* $\mathcal{L}_{pb}$ is defined by

$$\psi := \top \mid \bigwedge_{i\in I}\psi_i \mid \neg\psi \mid \psi a\psi' \mid \psi\tau\psi' \mid [\psi]_{\geq q}$$

for $a \in \Sigma$, $q \in \mathbb{Q}$ and possibly infinite index sets I. The satisfiability of an $\mathcal{L}_{pb}$-formula is defined by the clauses ($\top$), ($\bigwedge$), ($\neg$), ($\geq q$), ($\overleftarrow{\varepsilon}$) and

(η) $\quad \mu \quad \models \psi a\psi'$ $\quad$ if $\mu \xrightarrow{a}_c \mu'$, $\mu \models \psi$ and $\mu' \models \psi'$.

(η_τ) $\quad \mu \quad \models \psi\tau\psi'$ $\quad$ if, $\mu = \mu'$ or $\mu \xrightarrow{\tau}_c \mu'$, $\mu \models \psi$ and $\mu' \models \psi'$.

As in the non-probabilistic context [22, 21], the modality $\psi a\psi'$ (based on the notion of η-replication) allows to observe ψ' after an execution of action a that is preceded by the observation of ψ. This modality generalizes $a\psi$. A similar meaning for $\psi\tau\psi'$. These two modalities allow to check the branching of a process. Notice that _a_ does not force using ε, because of this, it is possible to check the branching after the execution of the action a.

▶ **Lemma 24.** *Suppose $\sum_{k\in K} p_k = 1$ for some finite index set K.*

1. *If $\mu_k \approx_{pb} \nu_k$ for $k \in K$, then $\sum_{k\in K} p_k\mu_k \approx_{pb} \sum_{k\in K} p_k\nu_k$.*
2. *If $\mu_k \approx_{\mathcal{L}_{pb}} \nu_k$ for $k \in K$, then $\sum_{k\in K} p_k\mu_k \approx_{\mathcal{L}_{pb}} \sum_{k\in K} p_k\nu_k$.*

▶ **Theorem 25.** *Let $\mu, \nu \in Disc(S)$, $\mu \approx_{pb} \nu$ iff $\mu \approx_{\mathcal{L}_{pb}} \nu$.*

3.4 Probabilistic weak bisimulation with sloppy probabilities

We have argued that weak bisimulation may be considered too strong. To deal with this problem, Eisentraut and co-workers introduced in [9] a notion called *weak distribution bisimulation*. We recall this process equivalence in Definition 27. Our presentation slightly differs from the original because we build on the notion of a weak decomposable relation (see Definition 26). In line with the nomenclature used in [21] for the global testing variants, we refer to our notion as *probabilistic weak bisimulation with sloppy probabilities*. We will motivate this further after the presentation of the logic that characterizes the semantics.

▶ **Definition 26.** A symmetric relation $\mathcal{R} \subseteq Disc(S) \times Disc(S)$ is called a *weak decomposable relation* if $\mu\,\mathcal{R}\,\nu$ and $\mu = \mu_1 \oplus_p \mu_2$ implies there are ν_1 and ν_2 such that $\nu \overset{\varepsilon}{\Longrightarrow}_c \nu_1 \oplus_p \nu_2$, $\mu_1\mathcal{R}\nu_1$ and $\mu_2\mathcal{R}\nu_2$.

Next we define for a weak decomposable relation when it is called a probabilistic weak bisimulation with sloppy probabilities.

▶ **Definition 27.** A weak decomposable relation $\mathcal{R} \subseteq Disc(S)\times Disc(S)$ is a called *probabilistic weak bisimulation with sloppy probabilities* if, for every $a \in \Sigma_\tau$, $\mu\mathcal{R}\nu$ and $\mu \xrightarrow{a}_c \mu'$ imply there is $\nu' \in Disc(S)$ s.t. $\nu \overset{\hat{a}}{\Longrightarrow}_c \nu'$ and $\mu'\,\mathcal{R}\,\nu'$. *Probabilistic weak bisimilarity with sloppy probabilities*, notation $\approx_{spw}$, is defined as the union of all the probabilistic weak bisimulations with sloppy probabilities.

The single difference of Definition 16 and Definition 27 is that the former uses a decomposable relation while the latter requires it to be weakly decomposable. However, this change is sufficient to capture $\delta(s) \approx_{spw} \delta(t)$ in Figure 5. Notice, $\mu_s \approx_{spw} \mu_t$ and these distributions are weakly decomposable because $\delta(s) \approx_{spw} \mu$.

In Figure 5 we have seen how the modal operator $[\cdot]_{\geq q}$ can be used to distinguish $\delta(s)$ and $\delta(t)$. We have argued that the approach of Hennessy does not differentiate between $\delta(s)$ and $\delta(t)$. However, to characterize the new semantics we only need to push his approach a little forward. The extra subtlety is this: Recall $\mu_s \approx_{spw} \mu_t$ and take into account the operator $\oplus_p$ with clause ($\oplus$). Then $\mu_t \models \psi_☹ \oplus_{0.25} \psi_☺$, but there are no μ_1 and μ_2 such that

$\mu_s = \mu_1 \oplus_{0.25} \mu_2$, $\mu_1 \models \psi_{☹}$ and $\mu_2 \models \psi_{☺}$. On the other hand, $\mu_s \overset{\varepsilon}{\Longrightarrow}_c 0.5\mu + 0.5\delta(☺)$ and $0.5\mu + 0.5\delta(☺) \models \psi_{☹} \oplus_{0.25} \psi_{☺}$. Therefore, in order to achieve $\mu_s \models \psi_{☹} \oplus_{0.25} \psi_{☺}$, we will allow a distribution to observe the measuring that is done after some internal behavior. This can be arranged 'for free', because the constraint that "*the outermost operator of ψ is not $\neg$, $\bigwedge$ nor $[\cdot]_{\geq q}$*" in the clause $(\overleftarrow{\varepsilon})$ does not concern the operator $\oplus_p$.

▶ **Definition 28.** The logic $\mathcal{L}_{spw}$ is defined by

$$\psi := \top \mid \bigwedge_{i \in I} \psi_i \mid \neg\psi \mid a\varepsilon\psi \mid \varepsilon\psi \mid \psi_1 \oplus_q \psi_2$$

for $a \in \Sigma$ and $q \in \mathbb{Q}$. The *satisfiability* of an $\mathcal{L}_{spw}$ formula is defined by clauses $(\top)$, $(\bigwedge)$, $(\neg)$, (a), $(\overleftarrow{\varepsilon})$, $(\overrightarrow{\varepsilon})$, and

$(\oplus)$ $\quad \mu \models \psi_1 \oplus_p \psi_2 \quad$ if $\mu = \mu_1 \oplus_p \mu_2$ and $\mu_i \models \psi_i$ for $i = 1, 2$.

We write $\models_{spw}$ to denote the satisfiability relation of $\mathcal{L}_{spw}$.

▶ **Theorem 29.** *For $\mu, \nu \in Disc(S)$, it holds that $\mu \approx_{spw} \nu$ iff $\mu \approx_{\mathcal{L}_{spw}} \nu$.*

4 Concluding remarks

In this paper we studied various behavioral equivalences for transitions systems over distributions in the presence of internal actions. An important contribution of our work is that we have consider weak hyper-transitions that deal with schedulers of infinite length. This allows to avoid the divergence-free condition for processes. Led by van Glabbeek's framework for the non-deterministic setting, we considered various ways to deal with τ-moves and provide logical characterizations for distribution-based probabilistic bisimulations. Moreover, we gave new characterization results following a uniform framework. The logics and axioms derive from the step-based behaviour encounter in the transfer conditions of the underlying bisimulation relation. The approach to prove correspondence results is the same for all notions of bisimulations considered. Crucial is the technical treatment of decomposable relations for weak combined and weak hyper-transitions (see Lemma 19). The uniform set-up allows to extend the results presented here to other semantics of the probabilistic branching-time spectrum without significantly more effort. Examples of this are η-bisimulation [1, 22] and delay bisimulation [22, 20].

▶ **Definition 30.** Let $\mathcal{R} \subseteq Disc(S) \times Disc(S)$ be a decomposable relation.

- $\mathcal{R}$ is a *probabilistic η-bisimulation* if given $\mu \mathrel{\mathcal{R}} \nu$, for every $a \in \Sigma_\tau$, $\mu \xrightarrow{a} \mu'$ implies
  1. $a = \tau$ and $\mu' \mathrel{\mathcal{R}} \nu$, or
  2. there are $\tilde{\nu}$, $\hat{\nu}$ and ν' such that $\nu \overset{\varepsilon}{\Longrightarrow}_c \tilde{\nu} \xrightarrow{a}_c \hat{\nu} \overset{\varepsilon}{\Longrightarrow}_c \nu'$ with $\mu \mathrel{\mathcal{R}} \tilde{\nu}$ and $\mu' \mathrel{\mathcal{R}} \nu'$.
- $\mathcal{R}$ is a *probabilistic delay bisimulation* if given $\mu \mathcal{R} \nu$, for every $a \in \Sigma_\tau$, $\mu \xrightarrow{a} \mu'$ implies
  1. $a = \tau$ and $\mu' \mathrel{\mathcal{R}} \nu$, or
  2. there are $\tilde{\nu}$ and ν' such that $\nu \overset{\varepsilon}{\Longrightarrow}_c \tilde{\nu} \xrightarrow{a}_c \nu'$ with $\mu' \mathrel{\mathcal{R}} \nu'$.

Probabilistic η-bisimilarity is defined as the union of all probabilistic η-bisimulations. *Probabilistic delay bisimilarity* is defined as the union of all probabilistic delay bisimulations.

We claim that the logics that characterize probabilistic η-bisimilarity and delay bisimilarity have the following syntax (using formulas ψ and φ for η and delay bisimulation, respectively).

$$\psi := \top \mid \textstyle\bigwedge_{i \in I} \psi_i \mid \neg\psi \mid \psi a \varepsilon \psi' \mid \psi \tau \psi' \mid [\psi]_{\geq q} \qquad \varphi := \top \mid \textstyle\bigwedge_{i \in I} \varphi_i \mid \neg\varphi \mid a\varphi \mid [\varphi]_{\geq q}$$

In the first logic, adding the modality ε in $\psi a \varepsilon \psi'$ does not allow anymore to check the branching of a process after the execution of a visible action. In the second logic, because the modalities $\psi a \psi'$ and $\psi \tau \psi'$ are removed, it is no longer possible to check the branching of a process before the execution of a visible action. We also remark that the problem presented in Figure 5 for probabilistic weak bisimulation is also present for these two new semantics. Then, one may also consider variants of the semantics with sloppy probabilities.

A decomposable relation $\mathcal{R} \subseteq \mathit{Disc}(S) \times \mathit{Disc}(S)$ straightforwardly induces a relation $\mathcal{S}$ over states just by putting $\mathcal{S} = \{ \, (s,t) \mid \delta(s) \, \mathcal{R} \, \delta(t) \, \}$. For distributions μ and ν with $\mu \mathcal{R} \nu$ we can define a weight function w for μ and ν with respect to $\mathcal{S}$ using the decompositions of μ and ν given by Lemma 10. Then the lifting of $\mathcal{S}$ to distributions agrees with $\mathcal{R}$. On the other hand, we expect that the approach based on *preserving* transitions used in [9] to give a state-based characterization of weak bisimulation with sloppy probabilities can be generalized to any weak decomposable relation. We have not studied this so far.

The probabilistic linear-time branching-time spectrum contains many more equivalences besides the ones discussed above. In [20] different types of combined transitions have been defined, each of which may potentially yield a new variant of a particular weak semantics. Alternatively, one can relax the condition over distributions, such as the variant of weak bisimulation of [19], or the variants of abstract probability bisimulation of [4]. The study of transition relations over distributions with internal actions can also be extended in other directions. We have considered image-finite relations, but this condition could be dropped, cf. [12]. Another interesting direction of future work considers relations over distributions with internal actions for uncountable state spaces, both regarding states and labels, as studied in [11]. Finally, it would be interesting to study how other probabilistic logics (like PCTL* [19] or variants of the probabilistic μ-calculus [15, 16]) behave in the distribution-based approach.

References

1 J.C.M. Baeten and R. J. van Glabbeek. Another look at abstraction in process algebra. In T. Ottmann, editor, *Proc. ICALP'87*, pages 84–94. LNCS 267, 1987.

2 S. Crafa and F. Ranzato. Logical characterizations of behavioral relations on transition systems of probability distributions. *TOCL*, 16(1):2:1–24, 2014.

3 P.R. D'Argenio and M.D. Lee. Probabilistic transition system specification: Congruence and full abstraction of bisimulation. In L. Birkedal, editor, *Proc. FOSSACS 2012*, 2012.

4 P.R. D'Argenio, M.D. Lee, and D. Gebler. SOS rule formats for convex and abstract probabilistic bisimulations. In S. Crafa et al., editor, *Proc. EXPRESS/SOS 2015*, 2015.

5 Y. Deng and M. Hennessy. On the semantics of Markov automata. *Information and Computation*, 222:139–168, 2013.

6 Y. Deng, R. J. van Glabbeek, M Hennessy, and C. Morgan. Characterising testing preorders for finite probabilistic processes. *Logical Methods in Computer Science*, 4(4), 2008.

7 Y. Deng and R.J. van Glabbeek. Characterising probabilistic processes logically. *CoRR*, abs/1007.5188, 2010.

8 J. Desharnais, V. Gupta, R. Jagadeesan, and P Panangaden. Weak bisimulation is sound and complete for PCTL*. *Information and Computation*, 208(2):203–219, 2010.

9 C. Eisentraut, H. Hermanns, J. Krämer, A. Turrini, and L. Zhang. Deciding bisimilarities on distributions. In Joshi, K.R. et al., editor, *Proc. QEST 2013*, pages 72–88, 2013.

10 M. Hennessy. Exploring probabilistic bisimulations, part I. *Formal Aspects of Computing*, 24(4-6):749–768, 2012.

11 H. Hermanns, J. Krcál, and J. Kretínský. Probabilistic bisimulation: Naturally on distributions. In P. Baldan and D. Gorla, editors, *Proc. CONCUR 2014*, pages 249–265, 2014.

12 H. Hermanns, A. Parma, R. Segala, B. Wachter, and L. Zhang. Probabilistic logical characterization. *Information and Computation*, 209(2):154–172, 2011.

13 K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

14 N.A. Lynch, R. Segala, and F.W. Vaandrager. Observing branching structure through probabilistic contexts. *SIAM Journal of Computing*, 37(4):977–1013, 2007.

15 M. Mio. Upper-expectation bisimilarity and Łukasiewicz μ-calculus. In A. Muscholl, editor, *Proc. FoSSaCS*, pages 335–350, 2014. `doi:10.1007/978-3-642-54830-7_22`.

16 M. Mio and A.K. Simpson. Łukasiewicz μ-calculus. *CoRR*, abs/1510.00797, 2015.

17 A. Parma and R. Segala. Logical characterizations of bisimulations for discrete probabilistic systems. In H. Seidl, editor, *Proc. FOSSACS 2007*, pages 287–301. LNCS 4423, 2007.

18 J. Sack and L. Zhang. A general framework for probabilistic characterizing formulae. In *VMCAI 2012, Philadelphia, USA, January, 2012. Proceedings*, pages 396–411, 2012.

19 L. Song, L. Zhang, and J.C. Godskesen. Bisimulations meet PCTL equivalences for probabilistic automata. *Logical Methods in Computer Science*, 9(2), 2013.

20 M.I.A. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems.* PhD thesis, Radboud Universiteit Nijmegen, 2002.

21 R. J. van Glabbeek. The linear time–branching time spectrum II. In E. Best, editor, *Proc. CONCUR '93*, pages 66–81. LNCS 715, 1993.

22 R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

Ackermannian Integer Compression and the Word Problem for Hydra Groups*

Will Dison[1], Eduard Einstein[2], and Timothy R. Riley[3]

1 **Bank of England**
Threadneedle Street, London, EC2R 8AH, United Kingdom
william.dison@gmail.com
2 **Department of Mathematics, Cornell University**
310 Malott Hall, Ithaca, NY 14853, USA
ee256@cornell.edu
3 **Department of Mathematics, Cornell University**
310 Malott Hall, Ithaca, NY 14853, USA
tim.riley@math.cornell.edu

Abstract

For a finitely presented group, the word problem asks for an algorithm which declares whether or not words on the generators represent the identity. The Dehn function is a complexity measure of a direct attack on the word problem by applying the defining relations. Dison and Riley showed that a "hydra phenomenon" gives rise to novel groups with extremely fast growing (Ackermannian) Dehn functions. Here we show that nevertheless, there are efficient (polynomial time) solutions to the word problems of these groups. Our main innovation is a means of computing efficiently with enormous integers which are represented in compressed forms by strings of Ackermann functions.

1998 ACM Subject Classification F.2.2 Geometrical problems and computations, E.4 Data compaction and compression

Keywords and phrases Ackermann functions, hydra, word problem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.30

1 Ackermann functions, compressed integers, and our first theorem

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. For $i \in \mathbb{N}$, the Ackermann functions $A_i : \mathbb{N} \to \mathbb{N}$ are a family of recursively defined increasingly fast-growing functions:

(i) $A_0(n) = n + 1$ for all $n \in \mathbb{Z}$,
(ii) $A_1(n) = 2n$ for all $n \in \mathbb{Z}$,
(iii) $A_i(0) = 1$ for all $i \geq 2$, and
(iv) $A_{i+1}(n + 1) = A_i A_{i+1}(n)$ for all $n \geq 0$ and all $i \geq 1$.

The following table, showing some values of $A_i(n)$, can be constructed by first inserting the $i = 0, 1$ rows and then $n = 0$ column, and then filling in the subsequent rows left-to-right according to the recurrence relation.

* We gratefully acknowledge partial support from NSF grant DMS-1101651 (TR) and Simons Collaboration Grant 318301 (TR), and the hospitality of the Mathematical Institute, Oxford (EE & TR), and the Institute for Advanced Study, Princeton (TR).

	0	1	2	3	4	$\cdots$	n	$\cdots$
A_0	1	2	3	4	5	$\cdots$	$n+1$	$\cdots$
A_1	0	2	4	6	8	$\cdots$	$2n$	$\cdots$
A_2	1	2	4	8	16	$\cdots$	2^n	$\cdots$
A_3	1	2	4	16	65536	$\cdots$	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}n$	$\cdots$
A_4	1	2	4	65536	$\left.2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\right\}65536$	$\cdots$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$			

Due to the increasing nesting of the recursion, the A_i represent the successive graduations in a hierarchy of all primitive recursive functions due to Grzegorczyk (see e.g. [30]).

The functions A_i are all strictly increasing and hence injective, so have partial inverses:

(I) $A_0^{-1} : \mathbb{Z} \to \mathbb{Z}$ mapping $n \mapsto n-1$,
(II) $A_1^{-1} : 2\mathbb{Z} \to \mathbb{Z}$ mapping $n \mapsto n/2$, and
(III) $A_i^{-1} : \operatorname{Img} A_i \to \mathbb{N}$ for all $i > 1$.

Starting with zero and successively applying a few Ackermann functions and their inverses can produce an enormous integer. For example,

$$A_3A_0A_1^2A_0(0) \;=\; A_3A_0A_1^2(1) \;=\; A_3A_0A_1(2) \;=\; A_3A_0(4) \;=\; A_3(5) \;=\; 2^{65536}$$

because

$$A_3(5) \;=\; A_2^5A_3(0) \;=\; A_2^5(1) \;=\; 2^{2^{2^{2^{2}}}} \;=\; 2^{65536}.$$

Thus Ackermann functions give highly compact representations for some very large numbers.

More precisely, here is how a string w of Ackermann functions may represent an integer $w(0)$. For $x_1, \ldots, x_n \in \{A_0^{\pm 1}, \ldots, A_k^{\pm 1}\}$, we say the word $w = x_nx_{n-1}\cdots x_1$ is *valid* if $x_mx_{m-1}\cdots x_1(0)$ is defined for all $0 \le m \le n$. That is, if we evaluate $w(0)$ by starting with 0 and proceeding through w from right to left applying successive x_i, we never encounter the problem that we are trying to apply x_i to an integer outside its domain.

For example, $w := A_2^{-1}A_1A_1A_0$ is valid, and $w(0) = \log_2(2 \cdot 2 \cdot (0+1)) = 2$. But $A_2A_0^{-1}$ and $A_1A_1^{-1}A_0$ are not valid because $A_0^{-1}(0) = -1$ is not in $\mathbb{N}$ (the domain of A_2) and because $A_0(0) = 1$ is not in $2\mathbb{Z}$ (the domain of A_1^{-1}).

Motivated by applications in group theory that we will describe in the next section, we wish to compute with these representations in an efficient manner. (Our choices of $\mathbb{Z}$ as the domains for A_0 and A_1 and our definition of A_0 represent small variations on the standard definitions of Ackermann functions, which are convenient for our applications.) One could just evaluate $w(0)$ using standard integer arithmetic, but this can be monumentally inefficient because of the sizes of the integers involved. Our first theorem is that is it possible to calculate efficiently in a rudimentary way with these representations of integers:

▶ **Theorem 1.** *Fix an integer $k \ge 0$. There is a polynomial-time algorithm, which on input a word w on $A_0^{\pm 1}, \ldots, A_k^{\pm 1}$, declares whether or not $w(0)$ represents an integer, and if so whether $w(0) < 0$, $w(0) = 0$ or $w(0) > 0$.*

(In fact our algorithm halts in time bounded above by a polynomial of degree $4 + k$. We have not attempted to optimize the degrees of the polynomial bounds on time complexity here or elsewhere in this work.)

2 The word problem, Dehn functions, and our second theorem

Elements of a group Γ with a generating set A can be represented by words—that is, products of elements of A and their inverses. To work with Γ, it is useful to have an algorithm which, on input a word, declares whether that word represents the identity element in Γ. After all, if we can recognize when a word represents the identity, then we can recognize when two words represent the same group element, and thereby begin to compute in Γ. The issue of whether there is such an algorithm is known as the *word problem* for (Γ, A) and was first posed by Dehn [9, 10] in 1912. (He did not precisely ask for an algorithm, of course, rather '*eine Methode angeben, um mit einer endlichen Anzahl von Schritten zu entscheiden...*'—that is, '*specify a method to decide in a finite number of steps...*')

Suppose a group Γ has a finite presentation $\langle a_1, \ldots, a_m \mid r_1, \ldots, r_n \rangle$. The *Dehn function* Area : $\mathbb{N} \to \mathbb{N}$ quantifies the difficulty of a *direct attack* on the word problem: roughly speaking Area(n) is the minimal N such that if a word of length at most n represents the identity, then it does so 'as a consequence of' at most N defining relations.

Here is some notation that we will use to make this more precise. Associated to a set $\{a_1, a_2, \ldots\}$ (an *alphabet*) is the set of inverse letters $\{a_1^{-1}, a_2^{-1}, \ldots\}$. The inverse map is the involution defined on $\{a_1^{\pm 1}, a_2^{\pm 1}, \ldots\}$ that maps $a_i \mapsto a_i^{-1}$ and $a_i^{-1} \mapsto a_i$ for all i. The inverse map extends to words by sending $w = x_1 \cdots x_s \mapsto x_s^{-1} \cdots x_1^{-1} = w^{-1}$ when each $x_i \in \{a_1^{\pm 1}, a_2^{\pm 1}, \ldots\}$. Words u and v are *cyclic conjugates* when $u = \alpha\beta$ and $v = \beta\alpha$ for some subwords α and β. *Freely reducing* a word means removing all $a_j^{\pm 1} a_j^{\mp 1}$ subwords. For Γ presented as above, *applying a relation* to a word $w = w(a_1, \ldots, a_m)$ means replacing some subword τ with another subword σ such that some *cyclic conjugate* of $\tau\sigma^{-1}$ is one of $r_1^{\pm 1}, \ldots, r_n^{\pm 1}$.

For a word w representing the identity in Γ, Area(w) is the minimal $N \geq 0$ such that there is a sequence of *freely reduced* words $w_0, \ldots, w_N$ with w_0 the freely reduced form of w, and w_N is the empty word, such that for all i, w_{i+1} can be obtained from w_i by *applying a relation* and then *freely reducing*. The *Dehn function* Area : $\mathbb{N} \to \mathbb{N}$ is defined by

$$\mathrm{Area}(n) \; := \; \max \{ \, \mathrm{Area}(w) \mid \text{words } w \text{ with } \ell(w) \leq n \text{ and } w = 1 \text{ in } \Gamma \, \}.$$

This is one of a number of equivalent definitions of the Dehn function. While a Dehn function is defined for a particular finite presentation for a group, its growth type—quadratic, polynomial, exponential etc.—does not depend on this choice. Dehn functions are important from a geometric point-of-view and have been studied extensively. There are many places to find background, for example [4, 5, 6, 10, 15, 16, 29, 31].

If Area(n) is bounded above by a recursive function $f(n)$, then it is possible to solve the word problem by an exhaustive search: to tell whether or not a given word w represents the identity, try all the possible ways of applying at most $f(n)$ defining relations and see whether one reduces w to the empty word. (There are finitely presented groups for which there is no algorithm to solve the word problem [3, 27].) Conversely, when a finitely presented group admits an algorithm to solve its word problem, Area(n) is bounded above by a recursive function (in fact Area(n) *is* a recursive function) [14].

There are finitely presented groups for which an extrinsic algorithm is far more efficient than this intrinsic brute-force approach. A simple example is $\mathbb{Z}^2 = \langle a, b \mid ab = ba \rangle$ (which has Dehn function Area$(n) \simeq n^2$). Given a word on $a^{\pm 1}, b^{\pm 1}$, the extrinsic approach amounts to searching exhaustively through all the ways of shuffling letters $a^{\pm 1}$ past letters $b^{\pm 1}$ to see if there is one which brings each $a^{\pm 1}$ together with an $a^{\mp 1}$ to be cancelled, and likewise each $b^{\pm 1}$ together with a $b^{\mp 1}$. It is much more efficient to read through the word and check that the number of a is the same as the number of a^{-1}, and the number of b is the same as the number of b^{-1}.

There are more dramatic examples of groups where Area(n) is a fast growing recursive function (so the 'brute force' algorithm succeeds but is extremely inefficient), but there are efficient ways to solve the word problem. Cohen, Madlener & Otto built extraordinary examples in a series of papers [7, 8, 25] where Dehn functions were first introduced (under then name *derivational complexity*). They designed their groups in such a way that the 'intrinsic' method of solving the word problem involves running a very slow algorithm which has been suitably 'embedded' in the presentation. But running this algorithm to see whether it halts on a given input is pointless as it is constructed to halt (eventually) on all inputs and so presents no obstacle to the word representing the identity. Their examples all admit algorithms to solve the word problem in running times that are at most $n \mapsto \exp^{(\ell)}(n) := \underbrace{\exp(\exp(\ldots\exp(n)))}_{\ell \text{ compositions of } \exp}$ for some ℓ. But for each $k \in \mathbb{N}$ they have examples which have Dehn functions growing like $n \mapsto A_k(n)$. Indeed, better, they have examples with Dehn function growing like $n \mapsto A_n(n)$.

Recently, yet more extreme examples were constructed by Kharlampovich, Miasnikov & Sapir [20]. By simulating Minsky machines in groups, for every recursive function $f : \mathbb{N} \to \mathbb{N}$, they construct a finitely presented group (which also happens to be residually finite and solvable of class 3) with Dehn function growing faster than f, but with word problem solvable in polynomial time.

There are also 'naturally arising' groups which have fast growing Dehn function but an efficient (that is, polynomial-time) solution to the word problem. A first example is $\langle\, a, b \mid b^{-1}ab = a^2 \,\rangle$. Its Dehn function grows exponentially (see, for example, [4]), but the group admits a faithful matrix representation

$$a \mapsto \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \qquad b \mapsto \begin{pmatrix} 1/2 & 0 \\ 0 & 1 \end{pmatrix},$$

so it is possible to check efficiently when a word on $a^{\pm 1}$ and $b^{\pm 1}$ represents the identity by multiplying out the corresponding string of matrices.

A celebrated 1-relator group due to Baumslag [1] provides a more dramatic example:

$$\langle\ a, b \ \mid\ (b^{-1}a^{-1}b)\, a\, (b^{-1}ab) = a^2\ \rangle.$$

Platonov [28] proved its Dehn function grows like $n \mapsto \overbrace{\exp_2(\exp_2 \cdots (\exp_2}^{\lfloor \log_2 n \rfloor}(1)) \cdots)$, where $\exp_2(n) := 2^n$. (Earlier results in this direction are in [2, 14, 15].) Nevertheless, Miasnikov, Ushakov & Won [26] solve its word problem in polynomial time. (In unpublished work I. Kapovich and Schupp showed it is solvable in exponential time [33].)

Higman's group

$$\langle\ a, b, c, d \ \mid\ b^{-1}ab = a^2,\ c^{-1}bc = b^2,\ d^{-1}cd = c^2,\ a^{-1}da = d^2\ \rangle$$

from [19] is another example. Diekert, Laun & Ushakov [11] recently gave a polynomial time algorithm for its word problem and, citing a 2010 lecture of Bridson, claim it too has Dehn function growing like a tower of exponentials.

The groups we focus on here are yet more extreme 'natural examples.' They arose in the study of *hydra groups* by Dison & Riley [13] . Let $\theta : F(a_1, \ldots, a_k) \to F(a_1, \ldots, a_k)$ be the automorphism of the free group of rank k such that $\theta(a_1) = a_1$ and $\theta(a_i) = a_i a_{i-1}$ for $i = 2, \ldots, k$. The family

$$G_k \ := \ \langle\ a_1, \ldots, a_k, t \ \mid\ t^{-1}a_i t = \theta(a_i)\ \ \forall i > 1\ \rangle,$$

are called *hydra groups*. Define

$$\Gamma_k \ := \ \langle\, a_1, \ldots, a_k, t, p \ \mid \ t^{-1}a_it = \theta(a_i), \ [p, a_it] = 1 \ \forall i \geq 1 \,\rangle,$$

which is an *HNN-extension* of G_k in which an additional *stable letter* p commutes with all elements of the subgroup $H_k := \langle a_1t, \ldots, a_kt\rangle$. It is shown in [13] that for $k = 1, 2, \ldots$, the subgroup H_k is free of rank k and Γ_k has Dehn function growing like $n \mapsto A_k(n)$. Our second theorem is that nevertheless:

▶ **Theorem 2.** *For all k, the word problem of Γ_k is solvable in polynomial time.*

(In fact, our algorithm halts in time at most a polynomial of degree $3k^2 + k + 2$.)

3 The membership problem, subgroup distortion, and our third theorem

A geometric feature known as *distortion* is the root cause of the Dehn function of the group Γ_k of the previous section growing like $n \mapsto A_k(n)$. The massive gap described in Theorem 2 between Dehn function and the time-complexity of the word problem for Γ_k is attributable to a similarly massive gap between a *distortion function* and the time-complexity of a *membership problem*. Here are more details.

Suppose H is a subgroup of a group G and G and H have finite generating sets S and T, respectively. So G has a *word metric* $d_S(g, h)$, the length of a shortest word on $S^{\pm1}$ representing $g^{-1}h$, and H has a word metric d_T similarly. The *distortion* of H in G is

$$\mathrm{Dist}_H^G(n) \ := \ \max\{\, d_T(1, g) \mid g \in H \text{ with } d_S(1, g) \leq n \,\}.$$

(Distortion is defined here with respect to specific S and T, but their choices do not affect the qualitative growth of $\mathrm{Dist}_H^G(n)$.) A fast growing distortion function signifies that H 'folds back on itself' dramatically as a metric subspace of G.

The *membership problem* for H in G is to find an algorithm which, on input of a word on $S^{\pm1}$, declares whether or not it represents an element of H.

If the word problem of G is decidable (as it is for all G_k, because, for instance, they are free-by-cyclic) and we have a recursive upper bound on $\mathrm{Dist}_H^G(n)$, then there is a brute-force solution to the membership problem for H in G. If the input word w has length n, then search through all words on $T^{\pm1}$ of length at most $\mathrm{Dist}_H^G(n)$ for one representing the same element as w. This is, of course, likely to be extremely inefficient, and especially so for H_k in G_k as the distortion $\mathrm{Dist}_{H_k}^{G_k}$ grows like $n \mapsto A_k(n)$. Nevertheless:

▶ **Theorem 3.** *For all k, the membership problem for H_k in G_k is solvable in polynomial time.*

(The algorithm we construct to prove this halts in time at most polynomial of degree $3k^2 + k$.)

Reducing Theorem 2 to Theorem 3 is straight-forward, requiring little more than a standard result about HNN-extensions. We detail this in Section 5 of [12].

4 Comparing our methods for Theorem 1 with power circuits and straight-line programs

Our strategy compares and contrasts with those used to solve the word problem for Baumslag's group in [26] and Higman's group in [11], where *power circuits* are the key tool. Power circuits provide concise representations of integers: power circuits of 'size' n represent (some) integers up to a height-n tower of powers of 2. There are efficient algorithms to perform addition, subtraction, and multiplication and division by 2 with power-circuit representations of integers, and to declare which of two power circuits represents the larger integer.

We too use concise representations of large integers, but in place of power circuits we use strings of Ackermann functions. These have the advantage that they may represent much larger integers. After all, $A_3(n) = \exp_2^{(n-1)}(1)$ already produces a tower of exponents, and the higher rank Ackermann functions grow far faster. However, we are aware of fewer efficient algorithms to perform operations with strings of Ackermann functions than are available for power circuits: we only have Theorem 1.

Our methods also bear comparison with the work of Lohrey, Schleimer and their coauthors [17, 18, 21, 22, 23, 24, 32] on efficient computation in groups and monoids where words are given in compressed forms using *straight-line programs* and are compared and manipulated using polynomial-time algorithms due to Hagenah, Plandowski and Lohrey. For instance Schleimer obtained polynomial-time algorithms solving the word problem for free-by-cyclic groups and automorphism groups of free groups and the membership problem for the handlebody subgroup of the mapping class group in [32].

5 The hydra phenomenon: connecting the group theory to Ackermann's functions

The reason G_k are named *hydra groups* is that the extreme distortion of H_k in G_k stems from a string-rewriting phenomenon which is a reimagining of the battle between Hercules and the Lernean Hydra, a mythical beast which grew two new heads for every one Hercules severed. Think of a *hydra* as a word w on $a_1, a_2, a_3, \ldots$. Hercules fights w as follows. He removes its first letter, then the remaining letters regenerate in that for all $i > 1$, each remaining a_i becomes $a_i a_{i-1}$ (and each remaining a_1 is unchanged). This repeats. An induction on the highest index present shows that every hydra eventually becomes the empty word. (Details are in [13].) Hercules is then declared victorious. For example, the hydra $a_2a_3a_1$ is annihilated in 5 steps:

$$a_2a_3a_1 \;\to\; a_3a_2a_1 \;\to\; a_2a_1a_1 \;\to\; a_1a_1 \;\to\; a_1 \;\to\; \textit{empty word.}$$

Define $\mathcal{H}(w)$ to be the number of steps required to reduce a hydra w to the empty word. (So $\mathcal{H}(a_3a_3a_1) = 5$.) Then, for $k = 1, 2, \ldots$, define functions $\mathcal{H}_k : \mathbb{N} \to \mathbb{N}$ by $\mathcal{H}_k(n) = \mathcal{H}(a_k^n)$. It is shown in [13] that $\mathcal{H}_k$ and A_k grow at the same rate for all k, since the two families of functions exhibit a similar recursion relation.

Here is an outline of the argument from [13] as to why $\mathrm{Dist}_{H_k}^{G_k}$ grows at least as fast as $n \mapsto \mathcal{H}_k(n)$ (and so as fast as $n \mapsto A_k(n)$). When $k \geq 2$ and $n \geq 1$, there is a reduced word $u_{k,n}$ on $\{a_1t, \ldots, a_kt\}^{\pm1}$ of length $\mathcal{H}_k(n)$ representing $a_k^n t^{\mathcal{H}_k(n)}$ in G_k on account of the hydra phenomenon. (For example, $u_{2,3} = (a_2t)^2(a_1t)(a_2t)(a_1t)^3$ equals $a_2^3t^7$ in G_2 since a_2, a_2, a_1, a_2, a_1, a_1, and a_1 are the $\mathcal{H}_2(3) = 7$ initial letters removed by Hercules as he vanquishes the hydra a_2^3.) It follows that in G_k

$$a_k^n a_2 \; ta_1 \; a_2^{-1} a_k^{-n} \;=\; u_{k,n} \, (a_2t) \, (a_1t) \, (a_2t)^{-1} \, {u_{k,n}}^{-1}.$$

The word on the left is a product of length $2n + 4$ of the generators $a_1^{\pm1}, \ldots, a_n^{\pm1}, t^{\pm1}$ of G_k and that on the right is a product of length $2\mathcal{H}_k(n) + 3$ of the generators $(a_1t)^{\pm1}, \ldots, (a_kt)^{\pm1}$ of H_k. As H_k is free of rank k and this word is reduced, it is not equal to any shorter word on these generators.

Hydra functions and Ackermann functions grow at the same rates, but do not precisely agree. So for Theorem 3 we, in fact, need a variation of Theorem 1, namely Proposition 3.4 in [12] which concerns a recursively defined family of functions ψ_i we call *ψ-functions*. Like strings of Ackermann functions, strings of ψ-functions (which we call *ψ-words*) can concisely represent extremely large integers. We do not have a direct proof of the equivalence of this proposition to Theorem 1, but they can be proved in essentially the same ways as the defining recurrence for the ψ_i is very similar to that for the A_i. We prefer to highlight Theorem 1 here because Ackermann functions have a long history and so are of intrinsic interest.

6 An outline of our strategy for Theorem 1

Here is a sketch of the algorithm we construct in Section 2 of [12] to prove Theorem 1. A more detailed high-level description is in Section 2.2 of [12].

Suppose we have a word w on $A_0^{\pm 1}, \ldots, A_k^{\pm 1}$ and we seek to determine in polynomial time whether it is valid and, if so, whether the integer $w(0)$ is negative, zero, or positive.

We will attempt to pass to successive new words $w = w_0, w_1, \ldots$ that are *equivalent* to w (denoted $w \sim w_j$) in that each w_j is valid if and only if w is, and when they both are, $w(0) = w_j(0)$. These words are obtained by making substitutions such as replacing a letter A_{i+1} in w by a subword $A_i A_{i+1} A_0^{-1}$ (the recursion defining the Ackermann functions), or deleting a subword $A_i A_i^{-1}$ or $A_i^{-1} A_i$. The lengths of these w_j will all be at most a constant times the length of w, which is important for our proof that our algorithm halts in polynomial time. The aim of the substitutions is to reach a $w' \sim w$ which contains no $A_1^{-1}, \ldots, A_k^{-1}$. Eliminating these letters represents progress because they denote functions which have sparse domains and so present the greatest obstacle to checking whether a word is valid.

We will look at how to make these substitutions momentarily, but first here's what happens when we have reached such a w'. Consider calculating a succession of integers beginning with 0 and ending with $w'(0)$ by evaluating $w'(0)$ letter-by-letter starting from the right. Only $A_0^{\pm 1}$ can trigger decreases in absolute value. So, to determine the sign of $w'(0)$, we can stop our evaluation if the integer calculated ever exceeds the length of w': after all, whatever sign our evaluation then has will be the sign of $w'(0)$. This threshold for the integers in our calculation allow for a polynomial time bound.

So how do we reach this w'? The rough idea is to 'cancel' each A_i^{-1} (where $i \geq 1$) in w with some A_i (if present) further to the right in w'. We do this inductively on i by manipulating suffixes of the form $\sigma = A_i^{-1} u A_i v$ such that u is a word on $A_0^{\pm 1}, \ldots, A_{i-1}^{\pm 1}$ and v a word on $A_0, \ldots, A_k$. A number of complications may arise. For instance, there are exceptional cases when substituting a A_{i+1} with $A_i A_{i+1} A_0^{-1}$ fails to preserve validity. Another issue is that we may have to introduce an A_i 'artificially' to cancel with an A_i^{-1}.

It is only possible to give a few details of our algorithm in the space available here. We choose to present a subroutine **BasePinch**, which serves as the base case of this inductive process of manipulating suffixes (the instance where u only contains letters $A_0^{\pm 1}$). It displays the crucial idea that allows us to operate within polynomial time: because the gaps between elements of $\mathrm{Img} A_i$ are large, we can either recognize efficiently that σ (and hence w) is invalid on account of u not being able to carry $A_i v(0) \in \mathrm{Img} A_i$ to another element of $\mathrm{Img} A_i$ (this is what the commentary on line 12 below is about), or σ is long enough that computing letter-by-letter by usual integer arithmetic is possible in polynomial time.

BasePinch will call two other subroutines (from Section 2.3 of [12]):

- **Bounds** which, on input $\ell \in \mathbb{N}$ (expressed in binary), returns in time $O(\ell)$ a list of all the (at most $(\log_2 \ell)^2$) triples of integers $(r, n, A_r(n))$ such that $r \geq 2$, $n \geq 3$, and $A_r(n) \leq \ell$.

Algorithm 1 BasePinch.

- Input a word $\sigma = A_r^{-1} u A_r v$ where $r \geq 1$, u is a word on $A_0^{\pm 1}$, and v is a word on $A_0^{\pm 1}, \ldots, A_k$.
- Either return that σ is invalid, or return a valid word $\sigma' = A_0^{l'} v \sim \sigma$ such that $\ell(\sigma') \leq \ell(\sigma) - 2$.
- Halt in time $O(\ell(\sigma)^4)$.

```
1    $l := u(0)$  (so  $A_r^{-1} A_0^l A_r v \sim w$)
     if Positive($A_r v$) $=$ Invalid, halt and return invalid
     run Positive($v$) to determine whether $v(0) < 0$
4    if $r \geq 2$ and $v(0) < 0$, halt and return invalid
     if $l = 0$, halt and return $\sigma' := v$
     if $r = 1$, halt and return $\sigma' := A_0^{l/2} v$ if $i$ is even or invalid otherwise
7
     we now have $l \neq 0$ and $r > 1$
     run Positive($A_0^l A_r v$) to determine if $A_0^l A_r v(0) \leq 0$ (so $\notin$ domain of $A_r^{-1}$)
10           if so, halt and return invalid
     run Positive($A_0^{-2|l|} A_r v$) to determine whether $A_r v(0) > 2|l|$
             if so, halt and return invalid
13
     we now have that $0 \leq v(0) \leq |l|$ and $0 < A_r v(0) \leq 2|l|$ and $A_r v(0) + l \leq 3|l|$
     calculate $v(0)$ by running Positive($A_0^{-i} v$) for $i = 0, 1, \ldots, |l|$
16   run Bounds($3\,|l|$)
     search the output of Bounds($3\,|l|$) to find $A_r v(0)$
     set $m := A_r v(0) + l$
19   search the output of Bounds($3\,|l|$) for $c$ with $A_r(c) = m$
                                                (so $c = A_r^{-1} A_0^l A_r v(0) = \sigma(0)$)
             if such a $c$ exists, halt and return $\sigma' := A_0^{c - v(0)} v$
22           else halt and return invalid
```

- **Positive** which, on input a word w on $A_0^{\pm 1}, A_1, \ldots, A_k$ in time $O(\ell(w)^3)$ either declares w invalid or declares whether $w(0) < 0$, $w(0) = 0$, or $w(0) > 0$.

  We use these properties of Ackermann functions:

Lemma 4.

$$A_i(n) + m \leq A_i(n+m) \qquad \forall i, n, m \geq 0, \tag{1}$$

$$|A_i(n) - A_i(m)| \geq \frac{1}{2} A_i(n) \qquad \forall i \geq 2 \textit{ and } n \neq m. \tag{2}$$

The proofs follow by inductive arguments applied to the definition of an Ackermann function. Refer to Lemma 2.1 of [12] for details.

Correctness of BasePinch. Here are the salient points line-by-line.

4: If $v(0) < 0$, then σ is invalid.

5: If $r < 2$ or $v(0) \geq 0$, $A_r^{-1} A_r v \sim v$.

6: Since A_1 is the function $n \mapsto 2n$, the parity of $A_0^l A_r v(0)$ is the parity of l when $r = 1$, and determines the validity of σ.

9, 11: We know $A_0^l A_r v$ and $A_0^{-2|l|} A_r v$ are valid at these points because $A_r v$ is valid.

12: Let $q = v(0)$. For all $p \neq q$ we have $|A_r(q) - A_r(p)| \geq \frac{1}{2} A_r(q)$ by Lemma 4, and so $|A_r(q) - A_r(p)| > |l|$. If $A_r^{-1} A_0^l A_r v$ is valid, then there exists $p \in \mathbb{N}$ such that $A_r(p) = A_0^l A_r v(0) = l + A_r(q)$, but then $|A_r(p) - A_r(q)| = |l|$ for some $p \neq q$ (since $l \neq 0$), contradicting $|A_r(q) - A_r(p)| > l$. Thus w is invalid.

14: The reason $0 < A_r v(0)$ is that $r > 1$ and so $\mathrm{Img} A_r$ contains only positive integers. And $A_r v(0) \leq 2\,|l|$ because of lines 11 and 12. It follows that $v(0) \leq |l|$ because $2v(0) = A_1 v(0) \leq A_r v(0) \leq 2\,|l|$. And $v(0) \geq 0$ since $v(0)$ is in the domain of A_r, which is $\mathbb{N}$ when $r > 1$. We have $A_0^l A_r v(0) \leq 3|l|$ here because $A_r v(0) \leq 2\,|l|$ and so $A_0^l A_r v(0) \leq l + 2\,|l|$.
20: If $m = A_r v(0) + l = A_0^l A_r v(0)$ is in the domain of A_r^{-1}, then $m > 0$. And, from line 14, we know $m \leq 3\,|l|$, so this will find c if it exists. If no such c exists, σ is invalid.
21: $A_0^{c-v(0)} v(0) = c = A_r^{-1}(l + A_r v(0)) = A_r^{-1} A_0^l A_r v(0)$.

We must show that $\ell(\sigma') \leq \ell(\sigma) - 2$. In the cases of lines 5 and 6, this is immediate, so suppose $r \geq 2$. As for line 21, by Lemma 4:

$$|c - v(0)| \leq |A_r(v(0) + c - v(0)) - A_r v(0)| = |A_r(c) - A_r(v(0)| = |l|$$

from which $\ell(\sigma') \leq \ell(\sigma) - 2$ follows immediately.

The integer calculations performed by the algorithm involve integers of absolute value at most $3\ell(\sigma)$. See [12] for details.

That **BasePinch** halts in time $O(\ell(\sigma)^4)$ follows the following. **Positive** and **Bounds** halt in cubic and linear time, respectively. **BasePinch** may add a pair of positive binary numbers each at most $2\ell(\sigma)$, may determine the parity of a number of absolute value at most $\ell(\sigma)$, and may halve an even positive number less than $\ell(\sigma)$. It calls **Positive** at most $|l| + 3 \leq \ell(\sigma) + 3$ times, always on a word of length at most $2\ell(\sigma)$. It calls **Bounds** at most once and on a non-negative integer that is at most $3\ell(\sigma)$. The output of **Bounds** is then searched at most twice and has size $O((\log_2 \ell(\sigma))^2)$. ◀

7 An outline of our strategy for Theorem 3

Here is an outline of our algorithm solving the membership problem for H_k in G_k from Section 4 of [12], proving Theorem 3. For a more detailed high-level description, see Section 4.1 of [12].

Suppose w is a word on $a_1^{\pm1}, \ldots, a_k^{\pm1}, t^{\pm1}$, so represents an element of G_k. To tell whether or not w represents an element of H_k, first collect all the $t^{\pm1}$ at the front by shuffling them to the left through the word, applying $\theta^{\pm1}$ as appropriate to the intervening a_i so that the element of G_k represented does not change. The result is a word $t^r v$ where $|r| \leq \ell(w)$ and v, a word on $a_1^{\pm1}, \ldots, a_k^{\pm1}$ has length at most a constant times $\ell(w)^k$ since θ is a free group automorphism of such polynomial growth.

Here is an example (one of a number in Section 4.2 of [12]). Suppose $w = a_3^4 a_2 t a_1 a_2^{-1} a_3^{-4}$. This equals tv in G_3 where $v = (a_3 a_2)^4 a_2 a_1^2 a_2^{-1} a_3^{-4}$ because $a_2 t = a_2 a_1$ and $a_3 t = a_3 a_2$.

We next look to carry the t^r back through v working from left to right, converting (if possible) what lies to the left of the power of t to a word on the generators $(a_1 t)^{\pm1}, \ldots, (a_k t)^{\pm1}$ of H_k. However the power of t being carried along will vary as this proceeds and, in fact, can get extremely large as a result of the hydra phenomenon. Similarly, the length of the word on the generators of H_k appearing to the left can be impractically long. For instance, in our example, the calculation outlined in Section 5 shows that w equals an element of the subgroup H_3 of G_3 which has length $2^{47} \cdot 3 - 1$ as a reduced word on the generators $(a_1 t)^{\pm1}$, $(a_2 t)^{\pm1}$, $(a_3 t)^{\pm1}$ of H_3.

So, instead of keeping track of the power of t directly, we record it as a word on ψ-functions (the functions that are analogues of Ackermann functions, as we explained in Section 5). Roughly speaking, checking whether this process ever gets stuck (in which case $w \notin H_k$) amounts to checking whether an associated ψ-word is valid. If the end of the word is reached, we then have a word on $(a_1 t)^{\pm 1}, \ldots, (a_k t)^{\pm 1}$ times some power of t, where the power is represented by a ψ-word whose length is at most a polynomial function of the length of w. We then determine whether or not $w \in H_k$ by checking whether or not that ψ-word represents 0. Both tasks can be accomplished suitably efficiently thanks to Proposition 3.4 in [12] (a variation of Theorem 1 as we explained in Section 5).

A complication is that we do not carry the power of t through from left to right one letter at a time. Rather, we partition v into subwords we call *rank k-pieces* and are determined by the locations of the a_k and a_k^{-1} in v. Each contains at most one a_k and at most one a_k^{-1}, and if the a_k is present in a piece, it is the first letter of that piece, and if the a_k^{-1} is present, it is the last letter. For instance, in our example $k = 3$ and $v = (a_3a_2)(a_3a_2)(a_3a_2)(a_3a_2^2a_1^2a_2^{-1}a_3^{-1})(a_3^{-1})(a_3^{-1})(a_3^{-1})$. We look to carry the power of t through v one piece at a time. Lemma 6.2 of [13] details how $v \in \bigcup_{s\in\mathbb{Z}} H_k t^s$ if and only if this is possible.

Whether the power of t can be carried through a piece $a_k^{\varepsilon_1} u a_k^{-\varepsilon_2}$ (here, $\varepsilon_1, \varepsilon_2 \in \{0,1\}$ and u is a reduced word on $a_1^{\pm 1}, \ldots, a_{k-1}^{\pm 1}$) depends on u in a manner that can be recursively analyzed by decomposing u into pieces with respect to the locations of the $a_{k-1}^{\pm 1}$ it contains. The main technical result behind our algorithm is our ‘Piece Criterion’ (Proposition 4.10 in [12]). This determines whether a power t^r can pass through a piece π—that is, whether $t^r\pi \in H_k t^s$ for some $s \in \mathbb{Z}$—and, if it can, how to represent s by a ψ-word. The way this plays out in our example is:

$$\begin{aligned}
t(a_3a_2) &\in H_k t^{f_1(0)} &&\text{where} && f_1 = \psi_1\psi_1^{-1},\\
t^{f_1(0)}(a_3a_2) &\in H_k t^{f_2(0)} &&\text{where} && f_2 = \psi_2\psi_3 f_1,\\
t^{f_2(0)}(a_3a_2) &\in H_k t^{f_3(0)} &&\text{where} && f_3 = \psi_2\psi_3 f_2,\\
t^{f_3(0)}(a_3a_2^2a_1^2a_2^{-1}a_3^{-1}) &\in H_k t^{f_4(0)} &&\text{where} && f_4 = \psi_3^{-1}\psi_2^{-1}(\psi_1)^2\psi_2^2\psi_3 f_3.\\
t^{f_4(0)}(a_3^{-1}) &\in H_k t^{f_5(0)} &&\text{where} && f_5 = \psi_3^{-1} f_4,\\
t^{f_5(0)}(a_3^{-1}) &\in H_k t^{f_6(0)} &&\text{where} && f_6 = \psi_3^{-1} f_5,\\
t^{f_6(0)}(a_3^{-1}) &\in H_k t^{f_7(0)} &&\text{where} && f_7 = \psi_3^{-1} f_6.
\end{aligned}$$

(The integers encoded here are $f_1(0) = 0$, $f_2(0) = -3$, $f_3(0) = -45$, $f_4(0) = -46$, $f_5(0) = -4$, $f_6(0) = -1$, and $f_7(0) = 0$. The conclusion is that $w \in H_3$ since $f_7(0) = 0$.)

Like in the previous section, we do not have space here to present many of the details, and so will only give an illustrative subroutine, namely ‘$\mathbf{Back}_m$.’ This attempts to pass a power t^r through a rank m-piece which has the special form $ua_m^{-\epsilon_2}$ where $\epsilon_2 \in \{0,1\}$, u is a word $a_1^{\pm 1} \cdots a_{m-1}^{\pm 1}$ and $m \geq 3$. There are several precursors to the construction of $\mathbf{Back}_m$:

- The construction is inductive on m. $\mathbf{Back}_m$ calls an algorithm $\mathbf{Push}_{m-1}$ (of Section 4.5 of [12]) which takes as input a word v on $a_0^{\pm 1}, \ldots, a_{m-1}^{\pm 1}$ and a ψ-word f representing an integer, and declares whether $t^{f(0)}v \in H_k t^s$ for some $s \in \mathbb{Z}$; if so, $\mathbf{Push}_{m-1}$ also returns a ψ-word g so that $t^{f(0)}v \in H_k t^{g(0)}$.
- The Piece Criterion (Proposition 4.10 in [12]) stipulates (in particular) that if $t^r ua_m^{-\epsilon_2} \in H_k t^s$ for some $s \in \mathbb{Z}$, exactly one of the following three conditions must hold:
  - **(a)** $\epsilon_2 = 0$ and $t^r ua_m^{-\epsilon_2} = t^r u \in H_k t^s$ (the trivial case).
  - **(b)** $\epsilon_2 = 1$, $s \leq 0$ and $t^r u \in H_k t^{\psi_m(s)}$.
  - **(c)** $\epsilon_2 = 1$, $s > 0$, $t^r ua_m^{-1}\theta^s(a_m) \in H_k t^{s-1}$ and $\theta^{s-1}(a_m^{-1})$ is a suffix of $ua_m^{-\epsilon_2}$.

  (Here, θ is the free group automorphism we defined in Section 2.)
- A routine $\mathbf{Prefix}_m$ (of Section 4.5 of [12]) inputs a rank-m piece $\pi = a_m u a_m^{-\epsilon_2}$ where $m \geq 3$. It returns the largest integer $i > 0$ (if any) such that $\theta^{i-1}(a_m)$ is a prefix of π and halts in time in $O(\ell(\pi)^2)$.

Algorithm 2 Back$_m$.

∘ Input a rank-m piece $\pi = ua_m^{-\epsilon_2}$ with $m \geq 3$ (so u is a reduced word on $a_1^{\pm 1}, \ldots, a_{m-1}^{\pm 1}$ and $\epsilon_2 \in \{0,1\}$) and a valid ψ-word f on $\psi_1^{\pm 1}, \ldots, \psi_k^{\pm 1}$. Let $r := f(0)$.
∘ Declare whether or not $t^r \pi \in \bigcup_{s \in \mathbb{Z}} H_k t^s$. And, if it is, return a valid ψ-word f' such that $t^{f(0)}\pi \in H_k t^{f'(0)}$, $\ell(f') \leq \ell(f) + 2(m-1)\ell(\pi) + 1$ and rank $(f') \leq \max\{\text{rank } (f), m\}$.
∘ Halt in time $O((\ell(\pi) + \ell(f))^{2m+k})$.

```
   run Push$_{m-1}(u,f)$ to test whether or not $t^r u \in \bigcup_{s\in\mathbb{Z}} H_k t^s$
2    if so, let $g$ be the valid $\psi$-word it outputs such that $t^r u \in H_k t^{g(0)}$
   if $\epsilon_2 = 0$,
     if $t^r u \in H_k t^{g(0)}$ (so, (a) of the Piece Criterion holds with $s = g(0)$),
5            return $f' := g$
     else declare  $t^r \pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$
     halt
8
   we now have that $\epsilon_2 = 1$
   run Psi($\psi_m^{-1} g$) to check validity of $\psi_m^{-1} g$ (so whether $g(0) \in$ Img $\psi_m$)
11   and, if so, to check  $\psi_m^{-1} g(0) \leq 0$
             (i.e. whether (2) of the Criterion holds with $s = \psi_m^{-1} g(0)$)
     if so, halt and return $f' := \psi_m^{-1} g$
14
   run Prefix$_m(\pi^{-1})$ to determine the maximum $i$ (if any)
         such that $a_{m-1}^{-1}\theta^{i-1}(a_m^{-1})$ is a suffix of $\pi$
17   if there is no such $i$, halt and declare $t^r \pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$
   for $s = 1$ to $i$:
     run Push$_{m-1}(u', f)$ where $u'$ is the reduced word representing $u a_m^{-1}\theta^s(a_m)$
20     if it outputs a $\psi$-word $h$, run Psi($\psi_1^{s-1} h$) to check if $h(0) = s - 1$
         if so halt and return $f' := \psi_1 h$
   declare that $t^{f(0)} w \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$
```

- **Psi** is our algorithm (of Section 3.3 of [12]) determining in polynomial time whether a ψ-word is valid and, if so, whether the integer it represents is negative, zero, or positive. We discussed its Ackermann-function analogue in the previous section.

Proof of correctness. Here is our justification line-by-line.

2: It follows from the workings of **Push**$_{m-1}$ (proved in Section 4.5 of [12]) that $\ell(g) \leq \ell(u) + \ell(f)$ and **rank** $(g) \leq \max\{$**rank** $(f), m\}$.

3–6: **Push**$_{m-1}$ in lines 1–2 tests whether or not $t^r u$ is in $\bigcup_{s\in\mathbb{Z}} H_k t^s$ and, if so, it identifies the s such that $t^r u \in H_k t^s$. The Piece Criterion then tells us that the answer to whether $t^r \pi \in \bigcup_{s\in\mathbb{Z}} H_k t^s$ is the same, and if affirmative the s agrees. By our comment on line 2, $\ell(f') \leq \ell(f) + \ell(u) = \ell(f) + \ell(\pi)$, and rank$(f') \leq \max\{\text{rank}(f), m\}$, as required.

10–13: Again, we refer back to lines 1–2 for whether or not $t^r u$ is in $\bigcup_{s_0\in\mathbb{Z}} H_k t^{s_0}$. Assuming that it is, in fact, in $H_k t^{g(0)}$, then Condition *2*, is satisfied if and only if $g(0) = \psi_m(s)$ for some $s \leq 0$. And that is checked in line 10. The Piece Criterion then tells us that the answer to this is the same as the answer to whether $t^r u \in \bigcup_{s\in\mathbb{Z}} H_k t^s$, and, if affirmative, the s agrees. By our comment on line 2, $\ell(f') = \ell(g) + 1 \leq \ell(f) + \ell(u) + 1 = \ell(f) + \ell(\pi)$ and rank$(f') \leq \max\{\text{rank}(f), m\}$, as required.

16–21: The aim here is to determine whether Condition *3* holds—that is, whether

$$t^r u a_m^{-1} \theta^s(a_m) \in H_k t^{s-1}$$

and $a_{m-1}^{-1}\theta^{s-1}(a_m^{-1})$ is a suffix of π for some $s > 0$—and, if so, output a ψ-word f' such that $f'(0) = s$. (This s must be unique, if it exists, because, by the Criterion, it is the s such that $t^r\pi \in H_k t^s$, and we know that is unique.)

The possibilities for s are limited to the range $1, \ldots, i$ by the suffix condition and the requirement that $s > 0$, where i is as found in line 16 and must be at most $\ell(\pi)$. If there is such a suffix $a_{m-1}^{-1}\theta^{i-1}(a_m^{-1})$ of π, then $a_{m-1}^{-1}\theta^{s-1}(a_m^{-1})$ is a suffix of π for all $s \in \{1, \ldots, i\}$. If there is no such suffix, then Condition *3* fails, and, as we know at this point that Conditions *1* and *2* also fail, we declare in line 17 that (by the Criterion), $t^r\pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$.

For each s in the range $1, \ldots, i$, lines 18–21 address the question of whether or not $t^r u a_m^{-1}\theta^s(a_m) \in H_k t^{s-1}$. First $\mathbf{Push}_{m-1}$ is called, which can be done because, on freely reducing $u a_m^{-1}\theta^s(a_m)$, the a_m^{-1} cancels with the a_m at the start of $\theta^s(a_m)$ to give a word of rank at most $m-1$. $\mathbf{Push}_{m-1}$ either tells us that $t^r u a_m^{-1}\theta^s(a_m) \notin \bigcup_{s'\in\mathbb{Z}} H_k t^{s'}$, or it gives a ψ-word h such that $t^r u a_m^{-1}\theta^s(a_m) \in H_k t^{h(0)}$. In the latter case, **Psi** is then used to test whether or not $h(0) = s - 1$.

By the specifications of $\mathbf{Push}_{m-1}$, $\ell(h) \le \ell(f) + 2(m-1)\ell(u')$. And, as $\pi = u a_m^{-1}$ has suffix $\theta^{s-1}(a_m^{-1})$, when we form u' by freely reducing $u a_m^{-1}\theta^s(a_m)$, at least half of $\theta^s(a_m) = \theta^{s-1}(a_m)\theta^{s-1}(a_{m-1})$ cancels into π. So $\ell(u') \le \ell(\pi)$, and

$$\ell(f') \;=\; \ell(h) + 1 \;\le\; \ell(f) + 2(m-1)\ell(u') + 1 \;\le\; \ell(f) + 2(m-1)\ell(\pi) + 1,$$

as required. Also, it is immediate that $\operatorname{rank}(f') \le \max\{\operatorname{rank}(f), m\}$, as required.

22: At this point, we know *1*, *2* and *3* fail for all $s \in \mathbb{Z}$, so $t^r\pi \notin \bigcup_{s\in\mathbb{Z}} H_k t^s$.

$\mathbf{Back}_m$ runs $\mathbf{Push}_{m-1}(u, f)$ once (with $\ell(u) \le \ell(\pi)$), $\mathbf{Psi}(\psi_m^{-1}g)$ at most once (with $\ell(g) \le \ell(\pi) + \ell(f)$), $\mathbf{Prefix}_m(\pi^{-1})$ at most once, $\mathbf{Push}_{m-1}(u', f)$ at most $i \le \ell(\pi)$ times (with $\ell(u') < \ell(\pi)$), and $\mathbf{Psi}(\psi_1^{s-1}h)$ at most $i \le \ell(\pi)$ times (with $1 \le s \le \ell(\pi)$ and $\ell(h) < \ell(f) + \ell(\pi)$). Other operations such as free reductions of words etc. do not contribute significantly to the running time. Referring to the specifications of $\mathbf{Push}_{m-1}$, **Psi**, and $\mathbf{Prefix}_m$, we see that they (respectively) contribute:

$$\begin{aligned}\ell(\pi)O((\ell(\pi) + \ell(f))^{2(m-1)+k+1}) + \ell(\pi)O((\ell(f) + 2\ell(\pi))^{4+k}) + O(\ell(\pi)^2)\\ = O((\ell(\pi) + \ell(f))^{2m+k})\end{aligned}$$

which is the claimed bound on the halting time of $\mathbf{Back}_m$. ◀

There is also an algorithm $\mathbf{Front}_m$ which takes a rank-m-piece π and a ψ-word f and determines (in a manner similar to $\mathbf{Back}_m$, see [12] Algorithm 4.2 for details) whether $t^{f(0)}$ can efficiently pass an initial a_m (if it exists) in π. If so, $\mathbf{Front}_m$ outputs a word of the form $u a_m^{-1}$ suitable for input into $\mathbf{Back}_m$ and a valid ψ word g such that checking whether $t^{f(0)}\pi \in Ht^s$ for some $s \in \mathbb{Z}$ is equivalent to checking whether $t^{g(0)} u a_m^{-1} \in Ht^s$ for some $s \in \mathbb{Z}$. If $t^{f(0)}$ does not pass through an initial a_m of π in one of three ways, Proposition 4.10 in [12] says that $t^{f(0)}\pi \notin H_k t^S$ for all $s \in \mathbb{Z}$. Putting together the algorithm $\mathbf{Front}_m$ with $\mathbf{Back}_m$ and implicitly $\mathbf{Push}_{m-1}$, we can construct the algorithm $\mathbf{Push}_m$. That way, given a word v on $a_1^{\pm1}, \ldots, a_m^{\pm1}$ and a ψ-word f, we have a polynomial time algorithm to determine whether or not $t^{f(0)}v \in H_k t^s$ and if so, to give a ψ-word g such that $g(0) = s$. We can then use **Psi** to determine whether g represents zero, and so whether $t^{f(0)}v$ represents an element of H_k.

A Reference to the technical details

The technical details are set out in full in *Taming the hydra: the word problem and extreme integer compression* [12], which is available from the arXiv repository at `http://arxiv.org/abs/1509.02557`.

References

1 G. Baumslag. A non–cyclic one–relator group all of whose finite quotients are cyclic. *J. Austral. Math. Soc.*, 10:497–498, 1969.

2 A. A. Bernasconi. *On HNN–extensions and the complexity of the word problem for one-relator groups*. PhD thesis, University of Utah, 1994. `http://www.math.utah.edu/~sg/Papers/bernasconi-thesis.pdf`.

3 W. W. Boone. Certain simple unsolvable problems in group theory I, II, III, IV, V, VI. *Nederl. Akad. Wetensch Proc. Ser. A.* **57**, 231–236, 492–497 (1954), **58**, 252–256, 571–577 (1955), **60**, 22-26, 227-232 (1957).

4 N. Brady, T. R. Riley, and H. Short. *The geometry of the word problem for finitely generated groups*. Advanced Courses in Mathematics CRM Barcelona. Birkhäuser–Verlag, 2007.

5 M. R. Bridson. The geometry of the word problem. In M. R. Bridson and S. M. Salamon, editors, *Invitations to Geometry and Topology*, pages 33–94. O.U.P., 2002.

6 M. R. Bridson and A. Haefliger. *Metric Spaces of Non-positive Curvature*. Number 319 in Grundlehren der mathematischen Wissenschaften. Springer Verlag, 1999.

7 D. E. Cohen. The mathematician who had little wisdom: a story and some mathematics. In *Combinatorial and geometric group theory (Edinburgh, 1993)*, volume 204 of *London Math. Soc. Lecture Note Ser.*, pages 56–62. Cambridge Univ. Press, Cambridge, 1995.

8 D. E. Cohen, K. Madlener, and F. Otto. Separating the intrinsic complexity and the derivational complexity of the word problem for finitely presented groups. *Math. Logic Quart.*, 39(2):143–157, 1993.

9 M. Dehn. Über unendliche diskontunuierliche Gruppen. *Math. Ann.*, 71:116–144, 1912.

10 M. Dehn. *Papers on group theory and topology*. Springer-Verlag, New York, 1987. Translated from the German and with introductions and an appendix by J. Stillwell, With an appendix by O. Schreier.

11 V. Diekert, J. Laun, and A. Ushakov. Efficient algorithms for highly compressed data: The Word Problem in Higman's group is in P. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 218–229, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. `doi:10.4230/LIPIcs.STACS.2012.218`.

12 W. Dison, E. Einstein, and T. R. Riley. Taming the hydra: the word problem and efficient calculation with ackermann-compressed integers. URL: `http://arxiv.org/abs/1509.02557`.

13 W. Dison and T. R. Riley. Hydra groups. *Comment. Math. Helv.*, 88(3):507–540, 2013. `doi:10.4171/CMH/294`.

14 S. M. Gersten. Isodiametric and isoperimetric inequalities in group extensions. Preprint, University of Utah, 1991.

15 S. M. Gersten. Isoperimetric and isodiametric functions of finite presentations. In G. Niblo and M. Roller, editors, *Geometric group theory I*, number 181 in LMS lecture notes. Camb. Univ. Press, 1993.

16 M. Gromov. Asymptotic invariants of infinite groups. In G. Niblo and M. Roller, editors, *Geometric group theory II*, number 182 in LMS lecture notes. Camb. Univ. Press, 1993.

17 N. Haubold and M. Lohrey. Compressed word problems in HNN-extensions and amalgamated products. *Theory Comput. Syst.*, 49(2):283–305, 2011. `doi:10.1007/s00224-010-9295-2`.

18 N. Haubold, M. Lohrey, and C. Mathissen. Compressed decision problems for graph products and applications to (outer) automorphism groups. *Internat. J. Algebra Comput.*, 22(8):1240007, 53, 2012. `doi:10.1142/S0218196712400073`.

19 G. Higman. A finitely generated infinite simple group. *J. London Math. Soc.*, 26:61–64, 1951.

20 O. Kharlampovich, A. Miasnikov, and M. Sapir. Algorithmically complex residually finite groups. `arXiv:1204.6506`.

21 M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210–1240, 2006. `doi:10.1137/S0097539704445950`.

22 M. Lohrey. Compressed word problems for inverse monoids. In *Mathematical foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Comput. Sci.*, pages 448–459. Springer, Heidelberg, 2011. `doi:10.1007/978-3-642-22993-0_41`.

23 M. Lohrey. *The compressed word problem for groups.* Springer Briefs in Mathematics. Springer, New York, 2014. `doi:10.1007/978-1-4939-0748-9`.

24 M. Lohrey and S. Schleimer. Efficient computation in groups via compression. In *Proc. Computer Science in Russia (CSR 2007)*, volume 4649 of *Lecture Notes in Computer Science*, pages 249–258. Springer, 2007.

25 K. Madlener and F. Otto. Pseudonatural algorithms for the word problem for finitely presented monoids and groups. *J. Symbolic Comput.*, 1(4):383–418, 1985.

26 A. G. Miasnikov, A. Ushakov, and D. W. Won. Power circuits, exponential algebra, and time complexity. *Internat. J. Algebra Comput.*, 22(6):1250047, 51, 2012. `doi:10.1142/S0218196712500476`.

27 P. S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudt Mat. Inst. Stkelov*, 44:1–143, 1955.

28 A. N. Platonov. An isoperimetric function of the Baumslag–Gersten group. *Vestnik Moskov. Univ. Ser. I Mat. Mekh.*, 3:12–17, 70, 2004. Translation in Moscow Univ. Math. Bull. 59 (2004).

29 T. R. Riley. What is a Dehn function? Chapter for the forthcoming *Office hours with a geometric group theorist*, Princeton University Press, M. Clay and D. Magalit, eds.

30 H. E. Rose. *Subrecursion: functions and hierarchies*, volume 9 of *Oxford Logic Guides*. The Clarendon Press Oxford University Press, New York, 1984.

31 M. Sapir. Asymptotic invariants, complexity of groups and related problems. *Bull. Math. Sci.*, 1(2):277–364, 2011. `doi:10.1007/s13373-011-0008-1`.

32 S. Schleimer. Polynomial-time word problems. *Comment. Math. Helv.*, 83(4):741–765, 2008. `doi:10.4171/CMH/142`.

33 P. Schupp. personal communication.

A Note on the Advice Complexity of Multipass Randomized Logspace*

Peter Dixon[1], Debasis Mandal[2], A. Pavan[3], and N. V. Vinodchandran[4]

1 **Department of Computer Science, Iowa State University**
`tooplark@cs.iastate.edu`

2 **Synopsys, Inc.**
`debasis.mandal@gmail.com`

3 **Department of Computer Science, Iowa State University**
`pavan@cs.iastate.edu`

4 **Department of Computer Science and Engineering, University of Nebraska-Lincoln**
`vinod@cse.unl.edu`

Abstract

Investigating the complexity of randomized space-bounded machines that are allowed to make *multiple passes over the random tape* has been of recent interest. In particular, it has been shown that derandomizing such probabilistic machines yields a weak but new derandomization of probabilistic time-bounded classes.

In this paper we further explore the complexity of such machines. In particular, as our main result we show that for any $\epsilon < 1$, every language that is accepted by an $O(n^{\epsilon})$-pass, randomized logspace machine can be simulated in deterministic logspace with *linear amount of advice*. This result extends an earlier result of Fortnow and Klivans who showed that RL is in deterministic logspace with linear advice.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Space-bounded computations, randomized machines, advice

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.31

1 Introduction

In the standard definition of probabilistic space-bounded computations, a probabilistic machine accesses its random tape in a *one-way*, read-only manner. In particular, the machine cannot reread the random bits unless they are stored in its work tapes. This model captures machines that can toss coins and hence is the most natural and this leads to well-studied space-bounded probabilistic classes BPL and RL [7].

While one-way access to the random tape is the most natural notion for probabilistic space-bounded computations, researchers have explored space-bounded models where the base machines are allowed to read contents of the random tape multiple times. An interesting earlier result is due to Nisan who showed that two-sided error logspace machines with one-way access to the random tape can be simulated by zero-error logspace machines that have two-way access to the random tape (BPL $\subseteq$ 2-*way*ZPL) [12]. However, the progress in understanding such machines and corresponding complexity classes has been sporadic and

* Research Supported in part by NSF grants 1421163 and 1422668.

many relations are unknown. For example, while we know that BPL is in P, we do not know whether 2-*way*BPL is even in deterministic sub-exponential time (note that it is in BPP). A key issue is that allowing two-way access to the random tape for space-bounded machines brings the corresponding nonuniform classes closer to randomized circuit complexity classes, where progress is known to be difficult.

Allowing the probabilistic machine to have *multiple passes* over the random tape is an access mechanism that is in between one-way and two-way. In particular, a $k(n)$-pass probabilistic machine is allowed to make $k(n)$ passes over the random tape for deciding an input of length n. Such probabilistic space-bounded machines were first considered by David, Papakonstantinou, and Sidiropoulos [5]. They showed that any pseudorandom generator that fools traditional $k(n)s(n)$ space-bounded machines can also fool $k(n)$-pass $s(n)$ space-bounded machines. As a corollary, they obtain that polylog-pass, randomized logspace is contained in deterministic polylog-space. Very recently, Mandal, Pavan, and Vinodchandrian [10] showed that such multipass probabilistic machines are interesting from a derandomization point of view. In particular, they showed the following theorem.

▶ **Theorem 1** ([10]). *For some constant $k > 0$, if every language decided by a probabilistic logspace machine that uses $O(\log n \log^{(k+3)} n)$ random bits and makes $O(\log^{(k)} n)$ passes over its random tape is in* P*, then* $\mathrm{BPTIME}(n) \subseteq \mathrm{DTIME}(2^{o(n)})$.

Here $\log^{(k)} n$ denotes log function applied k times iteratively. Note that showing $\mathrm{BPTIME}(n)$ is a subset of $\mathrm{DTIME}(2^{o(n)})$ is a significant open problem. Thus derandomizing a slightly non-constant pass probabilistic space-bounded machine yields a non-trivial derandomization of $\mathrm{BPTIME}(n)$.

The Main Result

This note considers the *advice complexity* of multipass probabilistic machines. Using standard techniques, it can be shown that 2-*way*RL[1] is in L/poly. Can this simulation be improved for multipass machines? Indeed, for RL, Fortnow and Klivans established that RL is in $\mathrm{L}/O(n)$. Thus one-pass logspace probabilistic machines can be simulated deterministically in logspace using *linear* advice. Our main contribution is to show that even if the base probabilistic machine is allowed n^δ passes over the random tape, the corresponding complexity class can still be simulated in $\mathrm{L}/O(n)$. More formally,

▶ **Main Theorem.** *For any $\delta < 1$, n^δ-pass* RL *is in* $\mathrm{L}/O(n)$.

This result extends Fortnow and Klivans' result and improves a result in [10] where it is shown that n^δ-pass RL is in deterministic $\log^2(n)$ space with linear advice.

2 Preliminaries

We refer the reader to [3] for standard notions and definitions of complexity theory. We first define probabilistic space-bounded computations. A probabilistic $s(n)$ space-bounded Turing Machine M has a *random tape* in addition to its input and work tapes. The machine has read-only access to both input and random tapes and it is allowed to read the contents on the random tape in a one-way manner. The total space used by the work tapes is bounded

[1] In this paper we consider one-sided error classes. Similar results can be obtained for two-sided error classes also.

by $s(n)$ and the machine can read at most $2^{s(n)}$ cells of the random tape. Thus the total number of random bits used by such machines is bounded by $2^{s(n)}$. The complexity class RL is the class of languages accepted by $O(\log n)$ space-bounded machines with one-sided error. We can relax the restriction on the machine's access to the random tape so that the machine M is allowed to read the contents of the random tape in a two-way manner. We use 2-*way*RL to denote the class that is analogous to RL but the base machine has two-way access to the random tape. In this paper, we use two-way, probabilistic machines that use limited amount of randomness. Let 2-*way*RL$[r(n)]$ denote the class of languages that are in 2-*way*RL and the base machine uses only $r(n)$ random bits on inputs of length n.

Next we define multipass, probabilistic, space-bounded machines.

▶ **Definition 2.** A probabilistic Turing machine M is a *$k(n)$-pass, $s(n)$ space-bounded machine* if

- M has read-only, two-way access to the input tape,
- total space used by the work tapes is bounded by $s(n)$,
- M is allowed to make $k(n)$ passes (on inputs of length n) over the random tape and during each pass it accesses the tape in a one-way, read only manner, and
- the total number of random bits used by the machine is bounded by $2^{s(n)}$.

▶ **Definition 3.** We say that a language L belongs to *$k(n)$-pass* RL if there exists a $k(n)$-pass, $O(\log n)$ space-bounded probabilistic Turing machine M such that for every input x, if $x \in L$, M accepts x with probability at least $1/2$ and if $x \notin L$, then the probability that M accepts x is 0.

Next we define the notion of advice [9].

▶ **Definition 4.** Let f be a function from natural numbers to natural numbers. A language L is in $\mathrm{L}/f(n)$, if there is a logspace machine M and a sequence of strings $a_1, a_2, \cdots$ such that $|a_n| \leq f(n)$ and for every input x of length n, $M(x, a_n)$ accepts if and only if $x \in L$.

For a probabilistic machine M and an input x, we use $M(x; r)$ to denote the computation of M on x, where r is the contents of the random tape. We now define the notion of *pseudorandom generators* for space-bounded machines.

▶ **Definition 5.** A family of functions $G = \{G_n\}_{n\geq 0}$ is an $(m(n), r(n), \epsilon)$ pseudorandom generator for space $s(n)$ if for every probabilistic $s(n)$ space-bounded machine M that uses $r(n)$ random bits (on inputs of length n) and for every input x of length n,

$$\big|\Pr[M(x; r) = 1] - \Pr[M(x; G_n(y)) = 1]\big| \leq \epsilon,$$

where r is chosen uniformly at random from $\Sigma^{r(n)}$ and y is chosen uniformly at random from $\Sigma^{m(n)}$.

We can define a similar notion of *pseudorandom generators* for $k(n)$-pass, $s(n)$-space.

The following theorem of David *et al.* [5] connects pseudorandom generators for multipass machines to pseudorandom generators for single pass machines.

▶ **Theorem 6** ([5]). *Let $G = \{G_n\}_{n\geq 0}$ be an $(m(n), r(n), \epsilon)$ PRG for space $k(n)s(n)$. Then G is an $(m(n), r(n), \epsilon 2^{k(n)})$ PRG for $k(n)$-pass, $s(n)$-space.*

Our proofs use the pseudorandom generator of Babai, Nisan, and Szegedy [4] which is based on lowerbounds for multiparty communication complexity. We now define the necessary notions that are needed.

▶ **Definition 7.** Let f be a Boolean function which takes k r-bit strings $x_1, \dots, x_k$ as inputs. Suppose k people wish to collectively compute $f(x_1, \dots, x_k)$ with the constraint that the i^{th} person does not know x_i. The k-party, ϵ-distributional communication complexity of f, denoted by $C_\epsilon(f)$, is the minimum number of bits that must be communicated among the k people (by, say, writing on a public whiteboard) in order to compute the function f on at least $\frac{1+\epsilon}{2}$ fraction of inputs (from Σ^{kr}).

▶ **Definition 8.** The Generalized Inner Product of r k-bit strings, denoted by $\mathrm{GIP}_{r,k}(x_1, \dots, x_k)$, is 0 if there is an even number of indices i, $1 \le i \le r$, at which all of $x_1[i], \dots, x_k[i]$ are 1; otherwise $\mathrm{GIP}_{r,k}(x_1, \dots, x_k)$ is 1.

Babai, Nisan, and Szegedy obtained the following bound on the communication complexity of GIP.

▶ **Theorem 9** ([4]).

$$C_\epsilon(\mathrm{GIP}_{r,k}) = \Omega\left(\frac{r}{4^k} + \log \epsilon\right)$$

3 Two-way Simulation and Linear Advice

In this section, we prove the main result of the paper which is stated below.

▶ **Theorem 10.** *Let $0 \le \delta < 1$ be a constant. Every language L that is in n^δ-pass* RL *is in* $\mathrm{L}/O(n)$.

The proof proceeds in two steps. We first show that any n^δ-pass randomized logspace machine can be simulated by a two-way randomized logspace machine that uses only n^γ ($\delta < \gamma < 1$) random bits. We then prove that such two-way machines can be decided in deterministic logspace with a linear amount of advice. The first step is proved in Theorem 11 and the second step is proved in Theorem 13.

▶ **Theorem 11.** *For every $0 \le \delta < 1$, there exists a γ, where $\delta < \gamma < 1$, such that for every language L in n^δ-pass* RL, *L is in 2-way*RL *with n^γ random bits.*

Before we present a formal proof, we give an overview of the proof. Consider the class RL. The well-known result of Nisan [11] states that there exists a PRG for logspace that stretches an $O(\log^2 n)$-length seed to $p(n)$ bits, where $p(n)$ is a polynomial in n. Moreover, the generator can be computed in *logspace*. The logspace machine that computes the PRG accesses the seed in a two-way manner. Thus using this generator we obtain that RL is in 2-*way*RL$[O(\log^2 n)]$. In fact, this is the first step in the work of Fortnow and Klivans [6]. A natural approach to Theorem 11 is to use a PRG for multipass, space-bounded machines. Let M be an n^δ-pass, logspace machine. By Theorem 6, any PRG for $O(n^\delta \log n)$-space will also be a PRG for M. Since M uses at most polynomially many random bits, we only need a PRG for $O(n^\delta \log n)$-space machines that use a polynomial number of random bits (as opposed to potentially $2^{O(n^\delta \log n)}$ random bits). A natural candidate is a generalized version of Nisan's generator that uses $O(S \log R)$ seed length for space S machines that use R random bits. This leads to a PRG that stretches $O(n^{\delta'})$ bits to polynomially many bits (for some $\delta' > \delta$), and this PRG fools the multipass machine M. Thus M can be simulated by a two-way randomized machine that uses $O(n^{\delta'})$ random bits. However, there is a small caveat in this argument. The space needed to compute this PRG is $O(\log^2 n)$. Thus this simulation of M using a two-way probabilistic machine takes $O(\log^2 n)$ space, whereas our goal is to simulate M using a two-way probabilistic machine that only uses $O(\log n)$ space.

We get around this problem by using the generator due to Babai, Nisan, and Szegedy [4]. They exhibited a PRG for space S that stretches $2^{\sqrt{S}}$-bits to 2^S bits and this PRG can be computed in $O(S)$ space. More specifically, when applied to logspace, their generator uses $O(2^{\sqrt{\log n}})$ seed length and is inefficient compared to Nisan's generator (in seed length). However, we observe that their generator is much easier to compute than Nisan's generator. We show that the BNS generator for $O(n^\delta \log n)$-space machines that use only polynomially many random bits has a seed length of n^γ ($\delta < \gamma < 1$) and can be computed in $O(\log n)$ space.

We now provide a formal proof.

Proof. Let M be an n^δ-pass, RL machine that accepts a language L. Assume that M uses n^ℓ random bits on inputs of length n. We will use the BNS generator to reduce the number of random bits used by M.

▶ **Theorem 12** ([4]). *Let $f_{r,k} : \Sigma^{rk} \to \{0,1\}$ be a Boolean function, $t > k$, and $N \leq \binom{t}{k}$. There is an $(rt, N, N\epsilon)$-pseudorandom generator G for s space-bounded machines that use N random bits, where $s < C_\epsilon(f)/k$. Also, the space required to compute G is the space required to compute the bits of f in antilexicographic order.*

We invoke this theorem for our choice of parameters. By Theorem 6, any $(m(n), n^\ell, \frac{1}{4\times 2^{n^\delta \log n}})$ generator for space $n^\delta \log n$ is an $(m(n), n^\ell, 1/4)$ generator for machine M. Note that we need a generator for $n^\delta \log n$ space-bounded machines that uses only n^ℓ random bits. Let $1 > \delta'$ be a constant that is greater than δ. We choose $r = n^{\delta'}$, $k = 2\ell\sqrt{\log n}$, $t = 2^k$, $N = n^\ell$, and the function f to be $\mathrm{GIP}_{k,r}$. By Theorem 9, the ϵ-distributional communication complexity of f is at least $\frac{r}{4^k} + \log \epsilon$. We pick ϵ as $\frac{1}{n^\ell} \times \frac{1}{4\times 2^{n^\delta \log n}}$. Thus $C_\epsilon(f)$ is at least

$$c\left(\frac{n^{\delta'}}{4^{2\ell\sqrt{\log n}}} - \ell \log n - 2 - \delta \log^2 n\right)$$

for some constant $c > 0$. With this we have $C_\epsilon(f)/k > c\left(\frac{n^{\delta'}}{4^{2\sqrt{\log n}}} - \ell \log n - 2 - \delta \log^2 n\right)/k > n^\delta \log n$ and $n^\ell < \binom{t}{k}$. Thus, by Theorem 12, there is an $(n^{\delta'} \times 2^{2\ell\sqrt{\log n}}, n^\ell, \frac{1}{4\times 2^{n^\delta \log n}})$-generator G for $n^\delta \log n$-space that uses n^ℓ random bits. Let γ be a constant that is greater than δ' (and less than 1). By theorem 6, G is an $(n^\gamma, n^\ell, 1/4)$-generator for n^δ-pass RL machines and hence fools M.

Our 2-*way*RL machine that simulates M works as follows. On any input x of length n, it has n^γ-bits written on its random tape. Let r denote the random string. It keeps track of the index of the random bit that M attempts to read. When M asks for ith random bit, it invoke the BNS generator on r, compute the ith bit of the generator, and continue the simulation of M. Note that the space needed by this machine is bounded by space needed by M plus the space needed to compute a bit of the BNS generator. We now claim that each bit of the generator can be computed in $O(\log n)$ space.

▶ **Claim 12.1.** *Each bit of the BNS generator can be computed in $O(\log n)$ space.*

Proof. The input to the BNS generator is an rt bit string which is viewed as t strings each of length r. Let $x_1, x_2, \cdots, x_t$ be these strings. We will describe the i^{th} bit of BNS generator. Consider the first $N = n^\ell$ k-subsets of $\{1, 2, \cdots, t\}$ in antilexicographic order: for two sets A and B, $A < B$ iff the largest element in $A \triangle B$ is an element of B. Let $S_1, S_2, \cdots, S_N$ be these subsets. Let $S_i = \{i_1, i_2, \cdots, i_k\}$. Then the i^{th} bit of BNS generator is $\mathrm{GIP}_{k,r}(x_{i_1}, x_{i_2}, \cdots, x_{i_k})$. Note that each S_i can be stored in $O(k \times \log t) = O(\log n)$ bits.

Also, $\mathrm{GIP}_{k,r}$ is computable in $\log k + \log r = O(\log n)$ space. We will describe an $O(\log n)$ space algorithm that takes set A and outputs B which is the next set in the antilexicographic order. Given A, initialize B with the maximal k-set. It generates all N k-subsets one by one and replaces B with the current set C if $A < C < B$. Note that given A and B, checking whether $A < B$ can be done in logspace. This leads to a logspace algorithm for generating the next set in the antilexicographic order. ◀

Thus we can simulate the n^{δ}-pass machine M using n^{γ} random bits in $O(\log n)$ space. Note that the simulating machine needs to access the random tape in a two-way manner. This completes the proof of the theorem. ◀

▶ **Theorem 13.** $2\text{-}way\mathrm{RL}[O(n)] \subseteq \mathrm{L}/O(n)$.

Proof. Let L be a language in $2\text{-}way\mathrm{RL}[O(n)]$ and let M be a machine that witnesses this with error probability $\leq 1/2$. The idea is to reduce the error probability of M to $1/2^{2n}$ using additional $O(n)$ random bits. Then a standard counting argument implies that there exists an $O(n)$-length string y for which $M(x; y)$ is correct on all strings x of length n. Thus y can be used as an advice. For reducing the error probability, we will use a *space-efficient expander walk* given by Gutfreund and Viola [8].

The general technique of using constant degree expander graphs to reduce the error probability of probabilistic machines is due to Ajtai, Komlos, and Szemeredi [1]. Let $r(n)$ denotes the number of random bits used by M on inputs on length n. Let G be a constant degree expander over $2^{r(n)}$ vertices. Consider the following process of producing k vertices: randomly pick a node v_0. For $1 \leq i \leq k-1$, v_i is a random neighbor of v_{i-1}. Note that each v_i is described using an $r(n)$ bit string. Also, the total random bits used in this process is $r(n) + O(k)$.

Consider the following simulation of M by M': on input x of length n, M' simulates M k times where the i^{th} simulation uses the encoding of v_i as the random string. If one of the simulations accept, M' accepts. It is well known that for any constant degree expander, there is a k where $k = O(n)$ so that the error probability of M' is $1/2^{2n}$.

▶ **Theorem 14** ([1]). *Given a constant degree expander G, there is a k where $k = O(n)$ so that the error probability of the above simulation is $\leq 1/2^{2n}$.*

To make this work we need be able to perform the random walk in logspace. The following theorem due to Allender, Jiao, Mahajan, and Vinay shows that random walk on certain constant degree expanders can be done in logspace. In a latter work, Gutfreund and Viola show that random walks on certain constant degree expanders can be done in AC_0[2].

▶ **Theorem 15** ([2, 8]). *There exist an infinite family $\{G_n\}_{n\geq 0}$ of expander graphs where G_n has 2^n nodes and constant degree D, and an $O(\log n)$-space algorithm A such that A on input $v_0 \in \{0,1\}^n$ and indices $\ell_1, \cdots, \ell_k$ where $1 \leq \ell_i \leq D$, outputs $v_0, v_1, \cdots, v_k$ where v_i is the ℓ_i^{th} neighbor of v_{i-1}. The algorithm runs in space $O(\log n + \log k)$.*

Proof of Theorem 13 follows from Theorem 14 and Theorem 15. ◀

Acknowledgements. We thank anonymous reviewers for helpful comments which improved the presentation of the paper.

References

1 M. Ajtai, J. Komlos, and E. Szemeredi. Deterministic Simulation in LOGSPACE. In *Proc. 19th ACM Symposium on Theory of Computing (STOC)*, volume 2, pages 132–140, 1987. doi:10.1145/28395.28410.

2 Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: Depth reduction and size lower bounds. *Theor. Comput. Sci.*, 209(1-2):47–86, 1998.

3 S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009. doi:10.1088/1742-6596/1/1/035.

4 L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *Journal of Computer and System Sciences*, 45(2):204–232, 1992. doi:10.1016/0022-0000(92)90047-M.

5 M. David, P.A. Papakonstantinou, and A. Sidiropoulos. How strong is Nisan's pseudorandom generator. *Information Processing Letters*, 111(16):804–808, 2011. doi:10.1016/j.ipl.2011.04.013.

6 L. Fortnow and A.R. Klivans. Linear advice for randomized logarithmic space. In *Proc. STACS*, 2006. URL: http://link.springer.com/chapter/10.1007/11672142_38.

7 J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977. URL: http://epubs.siam.org/doi/abs/10.1137/0206049.

8 D. Gutfreund and E. Viola. Fooling parity tests with parity gates. In *Proc. APPROX and RANDOM*, pages 381–392. Springer-Verlag Berlin, Heidelberg, 2004.

9 R.M. Karp and R.J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proc. 12th ACM Symp. on Theory of Computing*, pages 302–309, 1980.

10 Debasis Mandal, A. Pavan, and N. V. Vinodchandran. On probabilistic space-bounded machines with multiple access to random tape. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, pages 459–471, 2015.

11 N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992. doi:10.1007/BF01305237.

12 N. Nisan. On read once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993. URL: http://www.sciencedirect.com/science/article/pii/030439759390258U.

Complexity of Constraint Satisfaction Problems over Finite Subsets of Natural Numbers

Titus Dose

Julius-Maximilians-Universität, Würzburg, Germany
dose@informatik.uni-wuerzburg.de

Abstract

We study the computational complexity of constraint satisfaction problems that are based on integer expressions and algebraic circuits. On input of a finite set of variables and a finite set of constraints the question is whether the variables can be mapped onto finite subsets of $\mathbb{N}$ (resp., finite intervals over $\mathbb{N}$) such that all constraints are satisfied. According to the operations allowed in the constraints, the complexity varies over a wide range of complexity classes such as L, P, NP, PSPACE, NEXP, and even Σ_1, the class of c.e. languages.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Computational complexity, Constraint satisfaction problems, Integer expressions and circuits

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.32

1 Introduction

The problems investigated in this paper are motivated by constraint satisfaction problems and integer expressions. We first introduce these notions and then explain their connection.

Constraint satisfaction problems. A constraint satisfaction problem (CSP) is a computational problem that on input of a finite set of variables and a finite set of constraints asks whether there is a mapping from the variables to some fixed domain such that all constraints are satisfied.

An example of a classical CSP is 3-Colorability, i.e., the question of whether there is a mapping α from a graph's vertices onto $\{0, 1, 2\}$ such that for adjacent nodes u and v it holds that $\alpha(u) \neq \alpha(v)$.

The set of relations permitted in the constraints is called *constraint language*. Obviously CSPs over finite domains permitting arbitrary constraints belong to NP. The question of which constraint languages lead to CSPs even decidable in polynomial time has been a topic of intensive research over the past decades.

Feder and Vardi [4] conjectured a dichotomy for CSPs over finite domains such that these CSPs are either in P or NP-complete. This conjecture is still open.

In the past years there has been an increasing interest in CSPs over *infinite* domains. Here much higher complexities are obtained, and some problems are even undecidable.

Integer expressions and algebraic circuits. In 1973, Meyer and Stockmeyer [14] asked for the complexity of decision problems regarding so-called integer expressions. An integer expression is a term built by singleton sets of natural numbers, the pairwise addition, and set operations like union, intersection, or complement. Meyer and Stockmeyer investigated the membership problem, i.e., the question of whether a given natural number is contained in the subset of ℕ described by an integer expression. Moreover, they studied the inequality problem, i.e., the question of whether two given integer expressions describe the same subset of ℕ.

For some constant $m \in \mathbb{N}$ the integer expression $(0 \cup 2^1) + (0 \cup 2^2) + \cdots + (0 \cup 2^{m-1})$ describes the set of all even natural numbers $< 2^m$ in a succinct way. Note that here the natural number n is an abbreviation for the set $\{n\}$.

McKenzie and Wagner [11] considered generalized integer expressions and the complexity of membership problems for circuits over finite subsets of ℕ.

Here a circuit is a directed, acyclic graph with two kinds of nodes: on the one hand, there are input nodes containing a single natural number. On the other hand all remaining nodes, so-called operation nodes, perform one of the following operations: union, intersection, complement, pairwise addition, and pairwise multiplication. Each operation node has an indegree equal to the number of operands required by the operation.

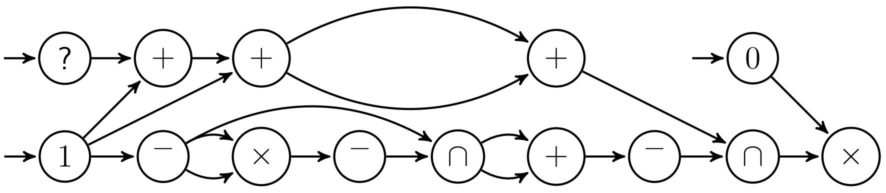
So each of the nodes computes a set of natural numbers: the input nodes compute singletons, and each operation node computes the corresponding set obtained from its predecessors. The set computed by the circuit overall is defined as the set computed by some fixed output node.

In contrast to integer expressions, these circuits are able to store intermediate results and reuse them several times. Thus, it is possible to describe large numbers and sets in a succinct way.

Similar to Meyer and Stockmeyer, who investigated the inequality problem for integer expressions, Glaßer et al. [5] considered the equivalence problem for circuits.

Moreover, Glaßer et al. [7] studied the satisfiability problem for circuits where the corresponding circuits are allowed to have unassigned input nodes. Now the problem is to decide for a given natural number b whether there exists an assignment for the input nodes such that b is contained in the set computed by the circuit. This modification makes the circuits even more similar to CSPs.

Consider the following circuit. Goldbach's conjecture fails if and only if there is an assignment for the ?-node in the circuit below such that the set computed by the rightmost node contains 0. Note that $\overline{\overline{1} \times \overline{1}} \cap \overline{1} = \mathbb{P}$ where $\mathbb{P}$ is the set of all primes.



Hence, if the satisfiability problem for circuits is decidable, then Goldbach's conjecture can be proven or refuted. Up to now it is not known whether this is possible.[1]

Connection of circuits and CSPs. Glaßer et al. [6] combined CSPs and integer circuits, and investigated CSPs over the domain of singletons of natural numbers and with operations from $\{+, \times, \cup, \cap, ^-\}$.

[1] Knuth [9] even assumes that Goldbach's conjecture is a problem that will never be solved. He mentions that "it might very well be that the conjecture happens to be true, but there is no rigorous way to prove it".

As we will see later, each CSP-instance can be represented by a primitive positive fo-sentence such as $\exists X\ (X+X+4)\cap\big(\mathbb{P}+\mathbb{P}\big)=0\cap 1$, where $\mathbb{P}$ can be expressed by $\overline{\overline{1}\times\overline{1}\cap\overline{1}}$. Here we use natural numbers as abbreviations for singletons. Since each variable stands for a singleton, this sentence is true if and only if Goldbach's conjecture fails.

However, the set of singletons of natural numbers is not closed under the mentioned operations. As it can be seen in the example above, variables and constants are singletons, whereas there are terms that describe bigger and even infinite sets. Therefore, we consider CSPs over the domain $\mathcal{P}_{\text{fin}}(\mathbb{N})=\{A\subseteq\mathbb{N}\mid A \text{ is finite}\}$, and replace the complement with the set difference. Thus, compared to the CSPs considered by Glaßer et al. [6] we consider problems that allow a straighter definition (i.e., variables and terms have the same domain), but that is more distant to the satisfiability problem for circuits.

Some results of the mentioned papers can be translated to our situation, but in general the questions for the enlarged domain turn out to be different ones.

Once an arithmetical and a set operation are permitted, we are not able to show the (un)decidability of the corresponding problems.

Therefore, we also consider a restricted version of the described CSPs, in which the domain is the set of all finite intervals over $\mathbb{N}$, denoted by $[\mathbb{N}]$. Note that this domain is not closed under all permitted operations anymore, but in several cases it is easier to obtain completeness results for this domain.

Our contribution. In the first part we consider all CSPs that only permit set operations. Here, for each problem we are able to show the $\leq_{\mathrm{m}}^{\log}$-completeness for one of the complexity classes L, P, and NP. We observe that there is a case in which the problem over $[\mathbb{N}]$ is more difficult than the corresponding problem over $\mathcal{P}_{\text{fin}}(\mathbb{N})$.

When also admitting arithmetical operations, the complexity is much higher. Each of the problems is $\leq_{\mathrm{m}}^{\log}$-hard for NP, and once both arithmetical operations are permitted, we obtain Σ_1-completeness.

The case with the addition as the only operation is particularly interesting. Here for CSPs over arbitrary finite sets we obtain only NP-hardness and membership in NEXP, and one of this paper's open questions is to close this gap. In contrast, the corresponding CSP over $[\mathbb{N}]$ turns out to be NP-complete. For the variant over $\mathcal{P}_{\text{fin}}(\mathbb{N})$ we even do not know whether the problem admitting addition and intersection is decidable. The corresponding problem over $[\mathbb{N}]$ is shown to be NP-complete again.

Furthermore, we consider the multiplication of intervals, and show that the CSP over $[\mathbb{N}]$ permitting only multiplication belongs to Σ_3^{p}. This result can be improved if the problem of testing whether two products of intervals are equal can be shown to belong to some class of the polynomial hierarchy lower than Π_2^{p}.

An overview over all results received in this paper can be found on pages 7 and 12. The technical report [3] provides a more comprehensive version of this paper.

2 Preliminaries

Let $\mathbb{N}$ (resp., $\mathbb{N}^+$) denote the set of non-negative (resp., positive) integers. $\mathcal{P}_{\text{fin}}(\mathbb{N})$ is the set of finite subsets of $\mathbb{N}$. For $A,B\in\mathcal{P}_{\text{fin}}(\mathbb{N})$ we define $A+B=_{\text{def}}\{a+b\mid a\in A,b\in B\}$ and $A\times B=_{\text{def}}\{ab\mid a\in A,b\in B\}$. By "$-$" we denote the set difference. Furthermore, $[\mathbb{N}]$ denotes the set of finite intervals over $\mathbb{N}$. An interval $\{x\mid a\leq x\leq b\}$ for non-negative integers a and b is represented by $[a,b]$. For $a>b$ it holds that $[a,b]=\emptyset$.

The set $f(A) = \{f(x) \mid x \in A\}$ for a function $f : A \to B$ and arbitrary sets A and B is also denoted by W_f.

We denote by L, P, NP, Σ_i^{p} and Π_i^{p} for $i \in \mathbb{N}$, PSPACE, and NEXP the standard complexity classes whose definitions can be found in textbooks on computational complexity [12]. The class of c.e. languages is denoted by Σ_1.

Note that commonly known $\leq_{\mathrm{m}}^{\mathrm{p}}$-complete problems for NP like SAT, 3SAT, or SOS are even $\leq_{\mathrm{m}}^{\log}$-complete for NP where $\leq_{\mathrm{m}}^{\log}$ (resp., $\leq_{\mathrm{m}}^{\mathrm{p}}$) denotes the many-one reduction computable in logarithmic space (resp., polynomial time).

We presume that finite subsets of natural numbers $\{a_1, \dots, a_k\}$ for $k \in \mathbb{N}^+$ are encoded such that the encoding's length is in $\Theta(\sum_{i=1}^{k} |a_k|)$, where $|a_k|$ denotes the length of the binary representation of a_k. An interval $[a, b]$ for $a < b$ is encoded such that the length of the encoding is in $\Theta(|a| + |b|)$. Note that all problems studied in this paper should be interpreted as subsets of $\mathbb{N}$ although for the sake of simplicity problems might be defined differently.

We successively define the CSPs this paper deals with. Let $\mathcal{O} \subseteq \{+, \times, \cup, \cap, -\}$ and X be a variable or a constant, where constants are finite subsets of $\mathbb{N}$.

Then X is a **term**. For terms β and γ as well as $\oplus \in \{+, \times, \cup, \cap, -\}$ the expression $(\beta \oplus \gamma)$ is a **term.** $X = Y$ for terms X and Y is an **atom**.

Let $a_1 = (t_{1,1} = t_{1,2}), \dots, a_m = (t_{m,1} = t_{m,2})$ for $m \in \mathbb{N}$ be atoms. Furthermore, let $X_1, \dots, X_n$ be the variables in the mentioned atoms. Then $\varphi = \exists X_1 \dots \exists X_n \; a_1 \wedge \dots \wedge a_m$ (or more shortly $\varphi = a_1 \wedge \dots \wedge a_m$) is an **$\mathcal{O}$-sentence**. If $\mathcal{O}$ is apparent from the context, we also write **sentence** instead of $\mathcal{O}$-sentence.

Let Var_φ denote the set of variables in φ, and let Const_φ denote the set of constants in φ. Moreover, we denote the set of all terms occurring in φ by T_φ.

We define the semantics of terms. A mapping $\alpha : T_\varphi \to \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ is an **assignment of terms** if the following conditions are satisfied.

- For constants C it holds that $\alpha(C) = C$.
- For variables X it holds $\alpha(X) \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$.
- For $\oplus \in \{+, \times, \cup, \cap, -\}$ and all terms $X \oplus Y$ it holds that $\alpha(X \oplus Y) = \alpha(X) \oplus \alpha(Y)$.

φ is **true** if and only if there is an assignment of terms α with $\alpha(t_{i,1}) = \alpha(t_{i,2})$ for all $i = 1, \dots, m$. We call such an α **satisfying**.

The restriction of an assignment of terms to Var_φ (resp., in some cases $\mathrm{Var}_\varphi \cup \mathrm{Const}_\varphi$) is called **assignment of variables** or **assignment**.

Moreover, an assignment of variables β is **satisfying** if the assignment of terms induced by β is satisfying. Note that for each assignment of variables β there is exactly one assignment of terms α such that $\beta(X) = \alpha(X)$ for all $X \in \mathrm{Var}_\varphi$.

▶ **Definition 1.** Let $\mathcal{O}$ denote an arbitrary subset of $\{+, \times, \cup, \cap, -\}$. Then we define $\mathrm{CSP}\Big(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \mathcal{O}\Big) =_{\mathrm{def}} \{\varphi \mid \varphi \text{ is a true } \mathcal{O}\text{-sentence}\}$.

Furthermore, we define

$$\mathrm{CSP}\Big([\mathrm{N}], \mathcal{O}\Big) =_{\mathrm{def}} \{\varphi \mid \varphi \text{ is an } \mathcal{O}\text{-sentence, all constants in } \varphi \text{ are intervals,}$$
$$\text{there is a satisfying assignment } \alpha \text{ with } \alpha(\mathrm{Var}_\varphi) \subseteq [\mathrm{N}]\}.$$

Here all constants are encoded as intervals. Hence, the length of the encoding of a constant $[a, b]$ for $a, b \in \mathbb{N}$ and $a \leq b$ is in $\Theta(|a| + |b|)$.

We also consider a slightly different problem. This enables us to simplify numerous proofs.

▶ **Definition 2.** Let $\mathcal{O} \subseteq \{+, \times, \cup, \cap, -\}$. We define

$$\mathrm{CSP}'\big(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \mathcal{O}\big) =_{\mathrm{def}} \{\varphi \,|\, \varphi \text{ is true, and each atom in } \varphi \text{ is of the form } X \oplus Y = Z \\ \text{for } X, Y, Z \in \mathrm{Const}_\varphi \cup \mathrm{Var}_\varphi \text{ and } \oplus \in \mathcal{O}\},$$

where we identify the atoms $X \oplus Y = Z$ and $Z = X \oplus Y$. $\mathrm{CSP}'\big([\mathbb{N}], \mathcal{O}\big)$ for $\mathcal{O} \subseteq \{\cap, +\}$ is defined analogously to $\mathrm{CSP}'\big(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \mathcal{O}\big)$.

The following lemma often allows us to presume that an input sentence is of the described simpler form. This is so because we can resolve a bigger term into several smaller terms by storing intermediate results in new variables. For CSPs over intervals, however, this works only when the set $[\mathbb{N}]$ is closed under the permitted operations.

▶ **Lemma 3.** $\mathrm{CSP}'\big(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \mathcal{O}\big) \equiv_m^{\log} \mathrm{CSP}\big(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \mathcal{O}\big)$ *for* $\mathcal{O} \subseteq \{+, \times, \cup, \cap, -\}$.
For $\mathcal{O} \subseteq \{+, \cap\}$ *it holds that* $\mathrm{CSP}'\big([\mathbb{N}], \mathcal{O}\big) \equiv_m^{\log} \mathrm{CSP}\big([\mathbb{N}], \mathcal{O}\big)$.

3 CSPs Permitting Set Operations Exclusively

We firstly focus on CSPs permitting only set operations.

Here all problems belong to NP, which is not the case when also arithmetic operations are allowed. In Section 4 we will see that some CSPs also permitting arithmetic operations are hard for PSPACE or even Σ_1.

Furthermore, when admitting only set operations it is much easier to prove CSPs to be complete for particular complexity classes. For each $\mathcal{O} \subseteq \{\cup, \cap, -\}$ and for $M \in \{\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), [\mathbb{N}]\}$ we show that $\mathrm{CSP}(\mathrm{M}, \mathcal{O})$ is $\leq_{\mathrm{m}}^{\log}$-complete for one of the classes L, P, and NP. Thus, all the problems studied in this section are considered exhaustively.

It can be proven: if an $\mathcal{O}$-sentence φ for $\mathcal{O} \subseteq \{\cup, \cap, -\}$ is true, then there is a satisfying assignment α with $W_\alpha \subseteq \bigcup_{C \in \mathrm{Const}_\varphi} C$. Hence, we obtain the following upper bound for all CSPs admitting only set operations.

▶ **Lemma 4.** *Let* $\mathcal{O} \subseteq \{-, \cup, \cap\}$ *and* $M \in \{[\mathbb{N}], \mathcal{P}_{\mathrm{fin}}(\mathbb{N})\}$. *Then* $\mathrm{CSP}\big(\mathrm{M}, \mathcal{O}\big) \in \mathrm{NP}$.

If we do not allow any operations at all, the corresponding CSP belongs to L. This can be proven by a Turing reduction to the problem USTCON, i.e., the question of whether two nodes in an undirected, finite graph are connected. This problem was shown to be in L by Reingold [13].

▶ **Lemma 5.** $\mathrm{CSP}\big(\mathrm{M}, \emptyset\big) \in \mathrm{L}$.

Thus, $\overline{\mathrm{CSP}(\mathrm{M}, \emptyset)}$ is trivially $\leq_{\mathrm{m}}^{\log}$-complete for L. Nevertheless, we remark that the reduction $\overline{\mathrm{USTCON}} \leq_{\mathrm{m}}^{\log} \mathrm{CSP}(\mathrm{M}, \emptyset)$ is simple. This shows that a direct proof of $\mathrm{CSP}(\mathrm{M}, \emptyset) \in \mathrm{L}$ is as difficult as a proof for $\mathrm{USTCON} \in \mathrm{L}$.

Problems admitting union only

▶ **Theorem 6.** $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\cup\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* P.

Proof. We first show $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\cup\}) \in \mathrm{P}$. According to lemma 3 it is sufficient to decide $\mathrm{CSP}'((\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\cup\})$ in polynomial time.

The following algorithm decides whether there is a satisfying assignment of variables for an input sentence φ of the form described in definition 2. For this purpose it successively computes an assignment α. Let $K = \bigcup_{C \in \mathrm{Const}_\varphi} C$.

1. For each constant C set $\alpha(C) = C$. For variables X set $\alpha(X) = K$.
2. Apply the following rules to every atom $X \cup Y = Z$.
   a. Set $\alpha(Z) = \alpha(Z) \cap (\alpha(X) \cup \alpha(Y))$.
   b. Set $\alpha(X) = \alpha(X) \cap \alpha(Z)$ and analogously $\alpha(Y) = \alpha(Y) \cap \alpha(Z)$.
   c. If $\alpha(C)$ for a constant C has been changed, return 0.
3. If $\alpha(X)$ for an $X \in \mathrm{Var}_\varphi$ has been changed in step 2, execute step 2 again.
4. Return 1.

Since the sets $\alpha(X)$ are decreased monotonically regarding the inclusion relation, there are at most $\mathrm{Var}_\varphi \cdot |\mathrm{K}|$ changes of values $\alpha(X)$ for variables X. Hence, the algorithm can be executed in polynomial time. Moreover, it can be proven that the algorithm returns 1 if and only if φ is true.

It remains to prove that $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\cup\})$ is $\leq_{\mathrm{m}}^{\log}$-hard for P: let MCVE be the problem of whether a monotone Boolean circuit, i.e., a Boolean circuit with solely OR- and AND-gates, outputs 1 on some fixed input. According to Greenlaw et al. [8] this problem is $\leq_{\mathrm{m}}^{\log}$-complete for P.

We simulate such a circuit by a $\{\cup\}$-sentence such that the set $\{1\}$ stands for the truth value 1, and $\emptyset$ stands for the truth value 0.

Each OR-gate can be simulated by an atom $X \cup Y = Z$ where X and Y stand for the gate's inputs, and Z stands for the gate's output.

Now we describe how AND-gates can be simulated. For an AND-gate consider the atoms $Z \cup H = X \wedge Z \cup H' = Y$ where X and Y stand for the gate's inputs, Z stands for the gate's output, and H and H' are auxiliary variables. These atoms are satisfied by an assignment α only if $\alpha(Z) = \{1\} \Rightarrow (\alpha(X) = \{1\} \wedge \alpha(Y) = \{1\})$. Note that there might indeed be an assignment of terms β with $\beta(Z) = \emptyset$ and $\beta(X) = \beta(Y) = \{1\}$. But as the circuit is monotone, this is no problem.

Finally, if G is the variable standing for the circuit's output value, we add the atom $G = \{1\}$. This completes the proof. ◀

Contrary to expectation the problem $\mathrm{CSP}([\mathbb{N}], \{\cup\})$ is – in case $\mathrm{P} \neq \mathrm{NP}$ – more difficult than the problem $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\cup\})$. Since the CSP over $[\mathbb{N}]$ might appear to be a restriction of the CSP over $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$, one might at first view rather expect the opposite. The $\{\cup\}$-sentences over $[\mathbb{N}]$ are indeed a restricted version of sentences over $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ because the constants belong to a strict subset of $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$.

However, since for the CSP over $[\mathbb{N}]$ also the variables are mapped onto intervals only, we obtain greater expressive power in this specific situation.

For instance, the atom $\{0\} \cup \{2\} = X \cup Y$ expresses that X is mapped onto $\{0\}$ (resp., $\{2\}$) by a satisfying assignment if and only if Y is mapped onto $\{2\}$ (resp., $\{0\}$). Furthermore, for $X, Y \in \{\{0\}, \{2\}\}$ the atoms $\{0\} \cup X \cup Y = \{0\} \cup Z \wedge Z \cup Z' = \{0\} \cup \{2\}$ express that Z is mapped onto $\{2\}$ by any satisfying assignment if and only if X or Y is mapped onto $\{2\}$. Otherwise Z is mapped onto $\{0\}$. Hence, interpreting $\{2\}$ (resp., $\{0\}$) as the truth value 1 (resp., 0), the logical disjunction and negation can be expressed. Thus, 3SAT is reducible to $\mathrm{CSP}([\mathbb{N}], \{\cup\})$ and we obtain the following theorem.

▶ **Theorem 7.** $\mathrm{CSP}([\mathbb{N}], \{\cup\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NP.

Further CSPs admitting set operations only. The methods and results for all other CSPs permitting only set operations are similar.

So we obtain the P-completeness for $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{\cap\})$. Yet, as $[\mathbb{N}]$ is not closed under union, but under intersection, $\mathrm{CSP}([\mathbb{N}], \{\cap\})$ is – in contrast to $\mathrm{CSP}([\mathbb{N}], \{\cup\})$ – not NP-complete, but P-complete.

▶ **Theorem 8.** $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{\cap\})$ *and* $\mathrm{CSP}([\mathbb{N}], \{\cap\})$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* P.

By use of sentences in $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{\cup, \cap\})$ and $\mathrm{CSP}([\mathbb{N}], \{\cup, \cap\})$ arbitrary propositional formulas can be described. Thus we obtain the following theorem.

▶ **Theorem 9.** $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{\cup, \cap\})$ *and* $\mathrm{CSP}([\mathbb{N}], \{\cup, \cap\})$ *are* $\leq_{\mathrm{m}}^{\log}$*-complete for* NP.

As $X = A \cup B$ can be expressed by $(X - A) - B = \emptyset \wedge A - X = \emptyset \wedge B - X = \emptyset$, and as $X = A \cap B$ holds if and only if $X = (A - (A - B))$, we obtain the first statement of the following theorem.

▶ **Theorem 10.** *Let* $\mathcal{O} \subseteq \{-, \cap, \cup\}$ *with* $- \in \mathcal{O}$.
1. $\mathrm{CSP}\big(\mathcal{P}_{\text{fin}}(\mathbb{N}), \mathcal{O}\big)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NP.
2. $\mathrm{CSP}\big([\mathbb{N}], \mathcal{O}\big)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NP.

Overview. The following table provides an overview over the results obtained in this section.

$\mathrm{CSP}\big(\mathrm{M}, \mathcal{O}\big)$ with	$M = \mathcal{P}_{\text{fin}}(\mathbb{N})$		$M = [\mathbb{N}]$	
	$\leq_{\mathrm{m}}^{\log}$-hard for	belongs to	$\leq_{\mathrm{m}}^{\log}$-hard for	belongs to
$\mathcal{O} = \emptyset$	L	L, 5	L	L, 5
$\mathcal{O} = \{\cup\}$	P, 6	P, 6	NP, 7	NP, 4
$\mathcal{O} = \{\cap\}$	P, 8	P, 8	P, 8	P, 8
$\mathcal{O} = \{\cup, \cap\}$	NP, 9	NP, 4	NP, 9	NP, 9
$\mathcal{O} \supseteq \{-\}$	NP, 10	NP, 4	NP, 10	NP, 4

For each of the problems the $\leq_{\mathrm{m}}^{\log}$-completeness for one of the complexity classes L, P, and NP is proven.

Additionally, it can be seen that in both situations we receive the same results for all $\mathcal{O} \subseteq \{\cup, \cap, -\}$ but $\mathcal{O} = \{\cup\}$. If it holds $\mathrm{P} \neq \mathrm{NP}$, then $\mathcal{O} = \{\cup\}$ is the only case throughout this paper where deciding a problem over $[\mathbb{N}]$ is more difficult than deciding the corresponding problem over $\mathcal{P}_{\text{fin}}(\mathbb{N})$. Contrary to that, in Section 4 there is an example for which – under the assumption $\mathrm{PSPACE} \neq \mathrm{NEXP}$ – the opposite is true.

4 CSPs Permitting Arithmetic Operations

Before we move to the consideration of some concrete problems, we prove an upper bound for all CSPs investigated in this paper:

▶ **Lemma 11.** *Let* $M \in \{\mathcal{P}_{\text{fin}}(\mathbb{N}), [\mathbb{N}]\}$ *and* $\mathcal{O} \subseteq \{+, \times, \cup, \cap, -\}$. *Then* $\mathrm{CSP}(\mathrm{M}, \mathcal{O}) \in \Sigma_1$.

Proof. The set $\{(\varphi, \alpha) \mid \varphi \in \mathrm{CSP}(\mathrm{M}, \mathcal{O}),\ \alpha$ is a satisfying assignment of variables for $\varphi\}$ is decidable. Hence, the problem $\mathrm{CSP}(\mathrm{M}, \mathcal{O})$ is a projection of a decidable set. ◀

4.1 CSPs over a Single Arithmetical Operation

In this section we consider the problems $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{+\})$ and $\mathrm{CSP}([\mathbb{N}], \{+\})$ as well as $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{\times\})$ and $\mathrm{CSP}([\mathbb{N}], \{\times\})$. All these problems turn out to be $\leq_{\mathrm{m}}^{\log}$-hard for NP. This shows that all CSPs permitting an arithmetical operation are $\leq_{\mathrm{m}}^{\log}$-hard for NP.

▶ **Definition 12.** Let $\mathrm{MSOS} =_{\mathrm{def}} \{x_1, \ldots, x_n, b \mid \exists a_1, \ldots, a_n \in \mathbb{N} : \sum_{i=1}^{n} a_i x_i = b\}$.

▶ **Lemma 13** ([1])**.** *MSOS is $\leq_{\mathrm{m}}^{\log}$-complete for NP.*

Let $M \in \{\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), [\mathbb{N}]\}$. In order to see that $\mathrm{MSOS} \leq_{\mathrm{m}}^{\log} \mathrm{CSP}(\mathrm{M}, \{+\})$, consider a $\{+\}$-sentence in which the factors a_i are guessed. Using the shift-and-add technique it can be made sure that some variable S_i in that sentence is mapped onto $\{a_i x_i\}$ by each satisfying assignment. Obviously $(x_1, \ldots, x_n, b) \in \mathrm{MSOS}$ if and only if the a_i can be chosen such that $\sum_{i=1}^{n} S_i = \{b\}$.

$\sum_{i \in I} a_i x_i = b$ is equivalent to $\prod_{i=1}^{n} 2^{a_i x_i} = 2^b$. Thus, with a similar approach using the square-and-multiply technique $\mathrm{MSOS} \leq_{\mathrm{m}}^{\log} \mathrm{CSP}(\mathrm{M}, \{\times\})$ can be shown.

▶ **Theorem 14.** *The following statements hold:*

1. $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{+\})$ *and* $\mathrm{CSP}([\mathbb{N}], \{+\})$ *are $\leq_{\mathrm{m}}^{\log}$-hard for* NP.
2. $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\times\})$ *and* $\mathrm{CSP}([\mathbb{N}], \{\times\})$ *are $\leq_{\mathrm{m}}^{\log}$-hard for* NP.

Yet we find different upper bounds. The problems over $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ belong to NEXP, whereas $\mathrm{CSP}([\mathbb{N}], \{\times\}) \in \Sigma_3^{\mathrm{p}}$ and even $\mathrm{CSP}([\mathbb{N}], \{+\}) \in \mathrm{NP}$ hold. We start with the consideration of CSPs that permit the addition only.

At first we show: if there is a satisfying assignment of variables for a given $\{+\}$-sentence φ, then there is also a satisfying assignment for φ that is "small" in some sense.

▶ **Lemma 15.** *Let $\varphi \in \mathrm{CSP}'\big(\mathrm{M}, \{+\}\big)$ for $M \in \{\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), [\mathbb{N}]\}$ and $n =_{def} |\varphi|$. Furthermore, let $x = \max(\bigcup_{C \in \mathrm{Const}_\varphi} C \cup \{0\})$. Then there is a satisfying assignment of variables α for φ with $\forall X \in \mathrm{Var}_\varphi : \ \max(\alpha(\mathrm{X}) \cup \{0\}) \leq \mathrm{x}2^{\mathrm{n}}$.*

Proof. The following non-deterministic algorithm successively constructs an assignment α.

1. Set $\alpha(C) = C$ for each constant C.
2. For each variable X with undefined $\alpha(X)$ occurring in an atom $Y + Z = X$ such that $\alpha(Y)$ and $\alpha(Z)$ are already defined, set $\alpha(X) = \alpha(Y) + \alpha(Z)$.
3. For each variable X with undefined $\alpha(X)$ occurring in an atom $X + Z_1 = Z_2$ such that $\alpha(Z_2)$ is defined and unequal to $\emptyset$, guess a set $S \in M$ with $\max\{S\} \leq \max(\alpha(Z_2))$ non-deterministically, and set $\alpha(X) = S$.
4. If there is a variable X such that $\alpha(X)$ has been defined in the last execution of the steps 2 and 3, then go to step 2.
5. For all variables X with undefined $\alpha(X)$ set $\alpha(X) = \emptyset$.
6. If α is a satisfying assignment, then return α.

It can be shown inductively that on at least one computation path the algorithm returns a satisfying assignment. Let $X_1, \ldots, X_{|\mathrm{Var}_\varphi|}$ denote the variables such that for $i < j$ the value $\alpha(X_i)$ is defined before $\alpha(X_j)$. Then it holds $\max(\alpha(X_i)) \leq x \cdot 2^i$ for $1 \leq i \leq |\mathrm{Var}_\varphi|$. ◀

Through this result we can proceed as follows: guess all assignments whose range is a subset of the power set of $\{0, 1, \ldots, x \cdot 2^n\}$. Test whether the respective assignment is satisfying, and return the corresponding return value.

Note that for CSPs over $[\mathbb{N}]$ only assignments with range $\subseteq \{[a, b] \mid a, b \leq x \cdot 2^n\}$ have to be considered. Hence, we obtain the following results.

▶ **Theorem 16.** *It holds that*

1. $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{+\}) \in \mathrm{NEXP}$
2. $\mathrm{CSP}([\mathbb{N}], \{+\}) \in \mathrm{NP}$.

▶ Remark. It is also possible to show $\mathrm{CSP}([\mathbb{N}], \{+\}) \in \mathrm{NP}$ by using ILP (the problem of whether an integer linear program has a solution) as an oracle. With that approach one can even show $\mathrm{CSP}([\mathbb{N}], \{+, \cap\}) \in \mathrm{NP}$ (theorem 24).

Now we consider $\mathrm{CSP}([\mathbb{N}], \{\times\})$. Contrary to $\mathrm{CSP}([\mathbb{N}], \{+\})$ we are only able to show $\mathrm{CSP}([\mathbb{N}], \{\times\})$ belonging to Σ_3^{p}. The reason for this difference is that $[\mathbb{N}]$ is not closed under multiplication. In particular, we may not assume that the input sentences are of the simplified form described in definition 2.

Nevertheless, by investigating the multiplication of intervals we find some properties that significantly simplify deciding $\mathrm{CSP}([\mathbb{N}], \{\times\})$.

▶ **Lemma 17.** *Let $A_1, \dots, A_m, B_1, \dots, B_n$ be finite intervals over $\mathbb{N}$ such that it holds $\emptyset \neq \prod_{i=1}^m A_i = \prod_{i=1}^n B_i \neq \{0\}$.*
Then $\prod_{1 \leq i \leq m, |A_i|=1} A_i = \prod_{1 \leq i \leq n, |B_i|=1} B_i$ and $\prod_{1 \leq i \leq m, |A_i| \neq 1} A_i = \prod_{1 \leq i \leq n, |B_i| \neq 1} B_i$.

▶ **Lemma 18.** *Let $A_1, \dots, A_m, B_1, \dots, B_n$ be intervals with at least two elements each such that $\max(A_1) \leq \max(A_2) \leq \dots \leq \max(A_m)$ and $\max(B_1) \leq \max(B_2) \leq \dots \leq \max(B_n)$. Let furthermore $\prod_{i=1}^m A_i = \prod_{i=1}^n B_i$. Then $\max(A_m) = \max(B_n)$.*

Proof. Let $L =_{\mathrm{def}} \prod_{i=1}^m A_i$ and $R =_{\mathrm{def}} \prod_{i=1}^n B_i$. In addition, let the greatest elements of A_i (resp., B_i) be denoted by a_i (resp., b_i). Because of $L = R$ the second greatest elements of L and R are equal. Thus $\max(L - \{\max(L)\}) = \max(R - \{\max(R)\})$.

We show $\max(L - \{\max(L)\}) = \big(\prod_{i=1}^{m-1} a_i\big) \cdot (a_m - 1) = \max(L) - \prod_{i=1}^{m-1} a_i$: the right equation is obvious. Furthermore, it apparently holds that $\big(\prod_{i=1}^{m-1} a_i\big) \cdot (a_m - 1) \in L - \{\max(L)\}$.

Let $x \in L - \{\max(L)\}$. There are $x_i \in A_i$ for $i = 1, \dots, m$ such that $x = \prod_{i=1}^m x_i$. Due to $x \neq \max(L)$ there is a j such that $x_j < a_j$. Then

$$x \leq \big(\prod_{1 \leq i \leq m, i \neq j} a_i\big) \cdot (a_j - 1) = \max(L) - \prod_{1 \leq i \leq m, i \neq j} a_i \leq \max(L) - \prod_{i=1}^{m-1} a_i.$$

Analogously it can be seen that $\max(R - \{\max(R)\}) = \max(R) - \prod_{i=1}^{n-1} b_i$. Hence $\max(L) - \prod_{i=1}^{m-1} a_i = \max(L - \{\max(L)\}) = \max(R - \{\max(R)\}) = \max(R) - \prod_{i=1}^{n-1} b_i$.

Because of $\max(L) = \max(R)$ we obtain $\prod_{i=1}^{m-1} a_i = \prod_{i=1}^{n-1} b_i$ and as a consequence $a_m = \frac{\max(L)}{\prod_{i=1}^{m-1} a_i} = \frac{\max(R)}{\prod_{i=1}^{n-1} b_i} = b_n$. ◀

We roughly describe a non-deterministic polynomial time algorithm satisfying the following properties: on input of a $\mathrm{CSP}([\mathbb{N}], \{\times\})$-instance the algorithm simplifies this sentence until no variables are left, and finally returns it. If and only if the input sentence is in $\mathrm{CSP}([\mathbb{N}], \{\times\})$, then there is a computation path on which a true sentence is returned.

After some preprocessing we obtain a possibly modified sentence and may neglect the sets $\emptyset$ and $\{0\}$ henceforth. Then we guess which variables stand for singletons. According to lemma 17 each atom can be split in two atoms: one for the "singleton variables and constants" and one for the other variables and constants. Hence, we obtain two sets of atoms, which can be considered independently.

For singletons we only store the exponents occurring in the prime decomposition. Two singletons are multiplied by adding the vectors of exponents componentwise. Hence, testing whether there is a satisfying assignment for the "singleton-atoms" can be done the same way as deciding $\mathrm{CSP}([\mathbb{N}], \{+\})$, which we have shown to be in NP. If there is a satisfying assignment, all "singleton-atoms" can be deleted. Otherwise return $[0, 1] = [1, 2]$ for instance.

Now consider the remaining atoms. Without loss of generality there is an atom $A_1 \times \cdots \times A_m = B_1 \times \cdots \times B_n$ such that all A_i are constants (if for each atom there are variables on both sides, then the remainig sentence is obviously true). For each variable B_j we guess an interval whose upper endpoint is $\leq \max(\bigcup_{i=1}^{m} A_i)$, and replace each occurrence of B_j with this interval. This approach is backed up by lemma 18 and can be repeated until no variable is left.

▶ **Lemma 19.** *There is a non-deterministic polynomial time algorithm $\mathcal{A}$ satisfying the following properties:*

- *As input $\mathcal{A}$ receives a $\{\times\}$-sentence φ whose constants are intervals.*
- *On each computation path $\mathcal{A}$ returns a $\{\times\}$-sentence ψ without any variables such that all constants are finite intervals with at least two elements each.*
- *If and only if $\varphi \in \mathrm{CSP}([\mathbb{N}], \{\times\})$, then on at least one computation path $\mathcal{A}$ returns a sentence $\psi \in \mathrm{CSP}([\mathbb{N}], \{\times\})$.*

Since $\mathcal{A}$ is a polynomial time algorithm, it holds that $|\psi| \in O(p(|\varphi|))$ for each ψ returned by the algorithm and some polynomial p.
It remains to test whether two products of intervals are equal.

▶ **Definition 20.** Let EPI be the set

$$\{(([a_1, a'_1], \ldots, [a_k, a'_k]), ([b_1, b'_1], \ldots, [b_n, b'_n])) \mid a_i < a'_i,\ b_i < b'_i,\ \prod_{i=1}^{k} [a_i, a'_i] = \prod_{i=1}^{n} [b_i, b'_i]\}\,.$$

▶ **Lemma 21.** $\mathrm{EPI} \in \Pi_2^{\mathrm{p}}$.

Proof. Let $A_1, \ldots, A_m, B_1, \ldots, B_n$ be non-empty intervals with $|A_i|, |B_j| \geq 2$.
It holds

$$\begin{aligned}\prod_{i=1}^{m} A_i = \prod_{i=1}^{n} B_i \Leftrightarrow & \forall x_1 \in A_1 \ldots \forall x_m \in A_m \forall y_1 \in B_1 \ldots \forall y_n \in B_n \\ & \exists x'_1 \in B_1 \ldots \exists x'_n \in B_n \exists y'_1 \in A_1 \ldots \exists y'_m \in A_m \\ & \prod_{i=1}^{m} x_i = \prod_{i=1}^{n} x'_i \wedge \prod_{i=1}^{n} y_i = \prod_{i=1}^{m} y'_i.\end{aligned}$$

Thus $\mathrm{EPI} \in \forall^{\mathrm{p}}\exists^{\mathrm{p}}\mathrm{P} = \Pi_2^{\mathrm{p}}$. ◀

▶ **Theorem 22.** $\mathrm{CSP}([\mathbb{N}], \{\times\}) \in \Sigma_3^{\mathrm{p}}$.

Proof. According to theorem 19 and lemma 21 the problem $\mathrm{CSP}([\mathbb{N}], \{\times\})$ can be decided by an NP-algorithm with Π_2^{p}-oracle. ◀

▶ Remark. Our decision algorithm for EPI tests whether two products of intervals are equal by considering all elements of the two products. Maybe this can be done more efficiently. In lemma 18 we have shown that a necessary condition for the equality of two products of intervals is that the greatest upper interval endpoint is the same in both products. If this condition can be extended to a sufficient condition that can still be tested efficiently, we would obtain a better upper bound for $\mathrm{CSP}([\mathbb{N}], \{\times\})$.

For the variant over $\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ we obtain membership in NEXP, which can be proven similarly to $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{+\}) \in \mathrm{NEXP}$.

▶ **Theorem 23.** $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\times\}) \in \mathrm{NEXP}$.

4.2 Addition and Intersection

Whereas we are not able to show $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{+, \cap\})$ to be decidable, the NP-hardness for the problem $\mathrm{CSP}([\mathbb{N}], \{+, \cap\})$ can be proven by use of integer linear programs.

Furthermore, due to the NP-hardness of $\mathrm{CSP}([\mathbb{N}], \{+\})$ also $\mathrm{CSP}([\mathbb{N}], \{+, \cap\})$ is NP-hard.

▶ **Theorem 24.** $\mathrm{CSP}([\mathbb{N}], \{+, \cap\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* NP.

Proof. According to the lemmata 3 and 14 it suffices to show $\mathrm{CSP}'([\mathbb{N}], \{+, \cap\}) \in \mathrm{NP}$. Hence, let φ be a $\{+, \cap\}$-sentence whose constants are solely intervals such that each atom is of the form $X \oplus Y = Z$ for $X, Y, Z \in \mathrm{Var}_\varphi \cup \mathrm{Const}_\varphi$ and $\oplus \in \{+, \cap\}$.

During a polynomial time computable preprocessing step φ is modified non-deterministically such that the following holds: $\varphi \in \mathrm{CSP}'([\mathbb{N}], \{+, \cap\})$ if and only if on at least one computation path a sentence φ' satisfying the following conditions has been computed: there is a satisfying assignment α for φ' with $\emptyset \notin W_\alpha$, and if there is an atom $X \oplus Y = Z$ in φ' containing $\emptyset$ as a constant, then $\oplus = \cap$, $Z = \emptyset$, and $X, Y \neq \emptyset$.

The problem of testing these conditions can be solved with the help of integer linear programs (ILP). For each $R \in (\mathrm{Var}_\varphi \cup \mathrm{Const}_\varphi) - \{\emptyset\}$ we introduce two ILP-variables r_0, r_1. If R is a constant and $R = [l, u]$, set $r_0 = l$ and $r_1 = u$.

1. For each atom $X + Y = Z$ we set up the equations $x_0 + y_0 = z_0$ and $x_1 + y_1 = z_1$.
2. For each atom $X \cap Y = Z$ with $Z \neq \emptyset$ use four further variables d, e, d', e'. We express $z_0 = \max(x_0, y_0)$ and $z_1 = \min(x_1, y_1)$:
   - On $z_0 = \max(x_0, y_0)$: Add $x_0 \leq z_0$, $y_0 \leq z_0$, $z_0 = dx_0 + ey_0$, and $d + e = 1$.
   - On $z_1 = \min(x_1, y_1)$: Add $x_1 \geq z_1$, $y_1 \geq z_1$, $z_1 = d'x_1 + e'y_1$, and $d' + e' = 1$.
3. For each atom $X \cap Y = Z$ with $Z = \emptyset$ we want to express $y_1 < x_0 \vee x_1 < y_0$. Hence, we guess a bit b. If $b = 0$, we add the inequation $y_1 < x_0$. Otherwise we add $x_1 < y_0$.
4. Furthermore, for every two ILP-variables x_0 and x_1 that describe the lower and upper endpoint of some interval we add the inequation $x_0 \leq x_1$.

If and only if one of the ILPs has a solution, it holds $\varphi \in \mathrm{CSP}([\mathbb{N}], \{+, \cap\})$. ◀

4.3 Lower Bounds for CSPs Permitting One Arithmetical and One Set Operation

We present two lower bounds obtained from literature. It should be possible to improve them in at least some cases.

By use of some results by Meyer and Stockmeyer [14] the following lower bound can be proven.

▶ **Theorem 25.**
1. $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{+, \cup\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* Π_2^{P}.
2. $\mathrm{CSP}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \{\times, \cup\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* Π_2^{P}.

These results can be shown by a reduction not making use of CSP-variables. This yields evidence that probably a better lower bound – such as the $\leq_{\mathrm{m}}^{\log}$-hardness for Σ_3^{P} – can be proven.

If beside one arithmetical operation there is also one of the two operations intersection and set difference available, we can revisit some results by McKenzie and Wagner [11] and show the corresponding problem to be PSPACE-hard.

▶ **Theorem 26.**
1. $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{+, \cap\})$ *is* $\leq_{\mathrm{m}}^{\mathrm{p}}$*-hard for* PSPACE*.*
2. $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{\times, \cap\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE*.*
3. $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{+, -\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE*.*
4. $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{\times, -\})$ *is* $\leq_{\mathrm{m}}^{\log}$*-hard for* PSPACE*.*

In Section 3 we observed that under the assumption $\mathrm{P} \neq \mathrm{NP}$ there are CSPs for which the variant over $[\mathbb{N}]$ is more difficult than the variant over $\mathcal{P}_{\text{fin}}(\mathbb{N})$. Here we have the opposite situation.

According to corollary 24 $\mathrm{CSP}([\mathbb{N}], \{+, \cap\})$ belongs to NP. According to theorem 26, however, $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{+, \cap\})$ is $\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for PSPACE. Hence, under the assumption $\mathrm{NP} \neq \mathrm{PSPACE}$ deciding $\mathrm{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{+, \cap\})$ is more difficult than deciding $\mathrm{CSP}([\mathbb{N}], \{+, \cap\})$.

4.4 Undecidability Results

As soon as both addition and multiplication are permitted, we receive undecidable, but computably enumerable problems.

More precisely, it can be shown that the problem of solving Diophantine equations, which was proven to be undecidable by Matiyasevich [2, 10], can be reduced to $\mathrm{CSP}(\mathrm{M}, \mathcal{O})$ for $M \in \{\mathcal{P}_{\text{fin}}(\mathbb{N}), [\mathbb{N}]\}$ and $\{+, \times\} \subseteq \mathcal{O}$.

▶ **Theorem 27.** *Let* $M \in \{\mathcal{P}_{\text{fin}}(\mathbb{N}), [\mathbb{N}]\}$ *and* $\mathcal{O} \subseteq \{\cup, \cap, -\}$*. Then* $\mathrm{CSP}\big(\mathrm{M}, \{+, \times\} \cup \mathcal{O}\big)$ *is* $\leq_{\mathrm{m}}^{\log}$*-complete for* Σ_1*.*

4.5 Overview

The following tables yield an overview over the results obtained in this section. For both tables there are sets of operations which do not occur in the list. For these problems we only know those bounds that follow directly from other entries in the corresponding tables (recall that whenever the set difference is permitted, then one may assume without loss of generality that also union and intersection are allowed).

The first table deals with the CSPs over $\mathcal{P}_{\text{fin}}(\mathbb{N})$.

$\mathrm{CSP}\big(\mathcal{P}_{\text{fin}}(\mathbb{N}), \mathcal{O}\big)$ with	hardness	member of
$\mathcal{O} = \{+\}$	$\leq_{\mathrm{m}}^{\log}$-hard for NP, 14	NEXP, 16
$\mathcal{O} = \{\times\}$	$\leq_{\mathrm{m}}^{\log}$-hard for NP, 14	NEXP, 23
$\mathcal{O} = \{+, \cup\}$	$\leq_{\mathrm{m}}^{\log}$-hard for Π_2^{p}, 25	Σ_1, 11
$\mathcal{O} = \{+, \cap\}$	$\leq_{\mathrm{m}}^{\mathrm{p}}$-hard for PSPACE, 26	Σ_1, 11
$\mathcal{O} = \{+, -\}$	$\leq_{\mathrm{m}}^{\log}$-hard for PSPACE, 26	Σ_1, 11
$\mathcal{O} \supseteq \{+, \times\}$	$\leq_{\mathrm{m}}^{\log}$-hard for Σ_1, 27	Σ_1, 11

The following tabular contains the results concerning CSPs over $[\mathbb{N}]$.

$\mathcal{O} = \mathrm{CSP}\big([\mathbb{N}], \mathcal{O}\big)$ with	$\leq_{\mathrm{m}}^{\log}$-hard for	member of
$\mathcal{O} = \{+\}$	NP, 14	NP, 16
$\mathcal{O} = \{\times\}$	NP, 14	Σ_3^{p}, 22
$\mathcal{O} = \{+, \cap\}$	NP, 14	NP, 24
$\mathcal{O} \supseteq \{+, \times\}$	Σ_1, 27	Σ_1, 11

The bounds for CSPs over $[\mathbb{N}]$ are in general lower than those for the corresponding CSPs over $\mathcal{P}_{\text{fin}}(\mathbb{N})$. If $[\mathbb{N}]$ is closed under all allowed operations, then we know the corresponding CSP to be complete for one of the classes L, NP, and Σ_1.

For the variant over $\mathcal{P}_{\text{fin}}(\mathbb{N})$ there are in almost all cases gaps between the lower and upper bound. It seems to be difficult to close these gaps.

In contrast to the section before, here remain several open questions. The following are particularly interesting:

Is $\text{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{+, \cap\})$ decidable? Is $\text{CSP}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \{+\})$ complete for some class between NP and NEXP? Does EPI belong to some class of the polynomial hierarchy lower than Π_2^{p}?

References

1 E. Böhler, C. Glaßer, B. Schwarz, and K. W. Wagner. Generation problems. *Theor. Comput. Sci.*, 345(2-3):260–295, 2005.

2 M. Davis, H. Putnam, and J. Robinson. The decision problem for exponential Diophantine equations. *Annals of Mathematics*, 74(2):425–436, 1961.

3 T. Dose. Complexity of constraint satisfaction problems over finite subsets of natural numbers. Technical Report 16-031, Electronic Colloquium on Computational Complexity (ECCC), 2016.

4 T. Feder and M. 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, February 1999.

5 Christian Glaßer, Katrin Herr, Christian Reitwießner, Stephen D. Travers, and Matthias Waldherr. Equivalence problems for circuits over sets of natural numbers. *Theory Comput. Syst.*, 46(1):80–103, 2010.

6 C. Glaßer, B. Martin, and P. Jonsson. Circuit satisfiability and constraint satisfaction problems around skolem arithmetic. In *Proceedings of the 12th International Conference on Computability in Europe (CiE-2016)*, Lecture Notes in Computer Science. Springer Verlag, 2016. To appear.

7 C. Glaßer, C. Reitwießner, S. Travers, and M. Waldherr. Satisfiability of algebraic circuits over sets of natural numbers. *Discrete Applied Mathematics*, 158(13):1394 – 1403, 2010.

8 R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. A Compendium of Problems Complete for P, 1991.

9 D. E. Knuth. All questions answered. *Notices of the AMS*, 49(3):318–324, 2002.

10 Y. V. Matiyasevich. Enumerable sets are Diophantine. *Doklady Akad. Nauk SSSR*, 191:279–282, 1970. Translation in Soviet Math. Doklady, 11:354–357, 1970.

11 Pierre McKenzie and Klaus W. Wagner. The complexity of membership problems for circuits over sets of natural numbers. *Computational Complexity*, 16(3):211–244, 2007.

12 C. M. Papadimitriou. *Computational complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

13 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, September 2008.

14 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM.

Faster Algorithms for the Maximum Common Subtree Isomorphism Problem[*]

Andre Droschinsky[1], Nils M. Kriege[2], and Petra Mutzel[3]

1 **Dept. of Computer Science, Technische Universität Dortmund, Germany**
`andre.droschinsky@tu-dortmund.de`
2 **Dept. of Computer Science, Technische Universität Dortmund, Germany**
`nils.kriege@tu-dortmund.de`
3 **Dept. of Computer Science, Technische Universität Dortmund, Germany**
`petra.mutzel@tu-dortmund.de`

Abstract

The maximum common subtree isomorphism problem asks for the largest possible isomorphism between subtrees of two given input trees. This problem is a natural restriction of the maximum common subgraph problem, which is NP-hard in general graphs. Confining to trees renders polynomial time algorithms possible and is of fundamental importance for approaches on more general graph classes. Various variants of this problem in trees have been intensively studied. We consider the general case, where trees are neither rooted nor ordered and the isomorphism is maximum w.r.t. a weight function on the mapped vertices and edges. For trees of order n and maximum degree Δ our algorithm achieves a running time of $\mathcal{O}(n^2\Delta)$ by exploiting the structure of the matching instances arising as subproblems. Thus our algorithm outperforms the best previously known approaches. No faster algorithm is possible for trees of bounded degree and for trees of unbounded degree we show that a further reduction of the running time would directly improve the best known approach to the assignment problem. Combining a polynomial-delay algorithm for the enumeration of all maximum common subtree isomorphisms with central ideas of our new algorithm leads to an improvement of its running time from $\mathcal{O}(n^6 + Tn^2)$ to $\mathcal{O}(n^3 + Tn\Delta)$, where n is the order of the larger tree, T is the number of different solutions, and Δ is the minimum of the maximum degrees of the input trees. Our theoretical results are supplemented by an experimental evaluation on synthetic and real-world instances.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases MCS, maximum common subtree, enumeration algorithms, maximum weight bipartite matchings

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.33

1 Introduction

The maximum common subgraph isomorphism problem (MCS) asks for an isomorphism between induced subgraphs of two given graphs that is of maximum weight w.r.t. a weight function on the mapped vertices and edges. The problem is of fundamental importance in applications like pattern recognition [5] or bio- and cheminformatics [9, 18]. MCS naturally generalizes the subgraph isomorphism problem (SI), where the task is to decide if one graph

[*] This work was supported by the German Research Foundation (DFG), priority programme "Algorithms for Big Data" (SPP 1736).

is isomorphic to a subgraph of another graph. Both problems are known to be NP-hard for general graphs.

It is not astonishing that these problems have been extensively studied for restricted graph classes. Polynomial time algorithms for SI and MCS in trees have been pioneered by Edmonds and Matula in the 1960s. They rely on solving a series of maximum bipartite matching instances, see [15]. These early results focused on the polynomial time complexity of the problem; since then considerable progress has been achieved in improving the running time of SI algorithms (also see [1] and references therein): Reyner [17, 23] and Matula [15] both showed a running time of $\mathcal{O}(n^{2.5})$ for rooted trees. Chung [4] later obtained the same bound for unrooted trees. Further improvements were made by Shamir and Tsur [19] who obtained time $\mathcal{O}(n^{2.5}/\log n)$ and $\mathcal{O}(n^{\omega})$, where ω is the exponent of matrix multiplication.

MCS on trees seems to be harder. For two rooted trees of size n, it is known that the problem can be solved, roughly speaking, in the same time as the associated maximum weight matching problem in a bipartite graph on n vertices: Gupta and Nishimura [11] presented an $\mathcal{O}(n^{2.5}\log n)$ algorithm for MCS in rooted trees by assuming weights to be in $\mathcal{O}(n)$, which allows to employ a scaling approach to the matching problem [10]. The running time can be improved to $\mathcal{O}(\sqrt{\Delta}n^2\log\frac{2n}{\Delta})$, where Δ denotes the maximum degree [12]. Allowing a real weight function to determine the similarity of mapped vertices gives rise to bipartite matching instances with unrestricted weights. Solving these with the Hungarian method leads to cubic running time, see e.g. [22]. Since the the size of the matching instances is bounded by the maximum degree Δ, the result can be improved to $\mathcal{O}(n^2\Delta)$ time [20]. Various related concepts for the comparison of rooted trees, either ordered or unordered, have been proposed and were studied in detail, see [22] and references therein, where the *tree edit distance* is a prominent example [3, 6].

In this article we consider the problem of finding a common subtree isomorphism in unrooted, unordered trees that is maximum w.r.t. a weight function on the mapped vertices and edges. This problem is directly relevant in various applications, where real-world objects like molecules or shapes are represented by (attributed) trees [16, 20]. Moreover, it forms the basis for several recent approaches to solve MCS in more general graph classes, see [2, 13, 14, 18]. Methods directly based on algorithms for rooted trees result in time $\mathcal{O}(n^4\Delta)$ by considering all pairs of possible roots. An improvement to $\mathcal{O}(n^3\Delta)$ has been reported in [20], which is limited to non-negative weight functions. Schietgat, Ramon and Bruynooghe [18] suggested an approach for MCS in outerplanar graphs, which solves the considered problem when applied to trees. The approach is stated to have a running time of $\mathcal{O}(n^{2.5})$, but in fact leads to a running time of $\Omega(n^4)$ in the worst case.[1]

Our contribution. We show that for arbitrary weights a maximum common subtree isomorphism between two trees G and H of order n with $\Delta(G) \leq \Delta(H)$ can be computed in time $\mathcal{O}(n^2(\Delta(G) + \log\Delta(H))$. We obtain the improvement by (i) considering only a specific subset of subproblems that we show to be sufficient to guarantee an optimal solution; (ii) exploiting the close relation between the emerging matching instances. We show that for general trees any further improvement of this time bound would allow to solve the assignment problem in $o(n^3)$, and hence improve over the best known approach to this famous problem for more than 30 years. For trees of bounded degree the running time bound of $\mathcal{O}(n^2)$ is

[1] The analysis of the algorithm appears to be flawed. An Erratum to [18] has been submitted to the *Annals of Mathematics and Artificial Intelligence*, see Appendix of `https://arxiv.org/abs/1602.07210`. Our experimental study of their implementation actually suggests a time bound of $\Omega(n^5)$.

tight. We apply our new techniques to the problem of enumerating all maximum common subtree isomorphisms, thus improving the state-of-the-art running times. Finally, we present an experimental evaluation on synthetic and real-world instances showing that our new algorithm is faster than existing approaches.

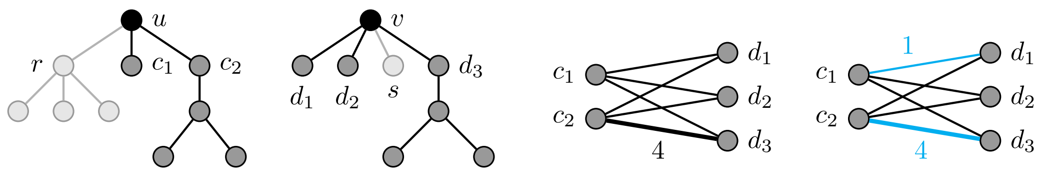
2 Preliminaries

In this paper, $G = (V, E)$ is a *simple undirected graph.* We call $v \in V$ a *vertex* and $uv = vu \in E$ an *edge* of G. For a graph $G = (V, E)$ we define $V(G) := V_G := V$, $E(G) := E_G := E$ and $|G| := |V(G)|$. For a subset of vertices $V' \subseteq V$ the graph $G[V'] := (V', E')$, $E' := \{uv \in E \mid u, v \in V'\}$, is called *induced* subgraph. A connected graph with a unique path between any two vertices is a *tree.* A tree G with an explicit root vertex $r \in V_G$ is called *rooted* tree, denoted by G^r. In a rooted tree G^r we denote the children of a vertex v by $C(v)$ and its parent by $p(v)$, where $p(r) = r$. The *depth* $\text{depth}(v)$ of a vertex v is the number of edges on the path from v to r. The *neighbors* of a vertex v are defined as $N(v) := \{u \in V_G \mid vu \in E_G\}$. The *degree* of a vertex $v \in V_G$ is $\delta(v) := |N(v)|$, the *degree* $\Delta(G)$ of a graph G is the maximum degree of its vertices.

For a graph $G = (V, E)$ a *matching* $M \subseteq E$ is a set of edges, such that no two edges share a vertex. A matching M of G is said to be *perfect*, if $2|M| = |V|$. A *weighted graph* is a graph endowed with a function $w : E \to \mathbb{R}$. The weight of a matching M in a weighted graph is $W(M) := \sum_{e \in M} w(e)$. We call a matching M of a weighted bipartite graph G a *maximum weight matching* (MWM) if there is no other matching M' of G with $W(M') > W(M)$. The *assignment problem* asks for a matching with maximum weight among all perfect matchings and we refer to a solution by MWPM.

An *isomorphism* between two graphs G and H is a bijective function $\phi : V_G \to V_H$ such that $uv \in E_G \Leftrightarrow \phi(u)\phi(v) \in E_H$; if such an isomorphism exists, G and H are said to be *isomorphic.* We call a graph G *subgraph isomorphic* to a graph H, if there is an induced subgraph $H' \subseteq H$ isomorphic to G. In this case, we write $G \preceq_\phi H$, where $\phi : V_G \to V_{H'}$ is an isomorphism between G and H'. A *common subgraph* of G and H is a graph I, such that $I \preceq_\phi G$ and $I \preceq_{\phi'} H$. The isomorphism $\varphi := \phi' \circ \phi^{-1}$ is called *common subgraph isomorphism* (CSI). For a function $f : X \to Y$ let $\text{dom}(f) := X$ be the domain of f. If there is no other CSI φ' with $|\text{dom}(\varphi')| > |\text{dom}(\varphi)|$, we call φ *maximum* common subgraph isomorphism (MCSI).

We generalize the above definitions to a pair of graphs G, H under a commutative weight-function $\omega : (V_G \times V_H) \cup (E_G \times E_H) \to \mathbb{R} \cup \{-\infty\}$. The weight $\mathcal{W}(\phi)$ of an isomorphism ϕ between G and H under ω is the sum of the weights $\omega(v, \phi(v))$ and $\omega(vu, \phi(v)\phi(u))$ of all vertices and edges mapped by ϕ. A maximum common subgraph isomorphism ϕ under a weight function is one of maximum weight $\mathcal{W}(\phi)$ instead of maximum size $|\text{dom}(\phi)|$. Note, this is less restrictive than a common approach for isomorphisms on labeled graphs, where the labels must match. By defining ω such that mapped vertices and mapped edges add 1 and 0, respectively, to the weight, we obtain an isomorphism of maximum size. Therefore, in the following we consider graphs under a weight function unless stated otherwise and refer to the corresponding solution as MCSI. For convenience we replace the word *graph* by *tree* in the above definitions when appropriate. The *maximum common subtree isomorphism problem* (MCST) is to determine the weight of an MCSI, where the input graphs and the common subgraph are trees. We further define $[1..k] := \{1, \ldots, k\}$ for $k \in \mathbb{N}$.



(a) Rooted subtree G^r_u. **(b)** Rooted subtree H^s_v. **(c)** Matching problem. **(d)** MWM.

Figure 1 Two rooted subtrees (a) and (b), the associated weighted matching instance (c), and an MWM on that instance (d). Light gray vertices and edges are not part of the rooted subtrees, root vertices are shown in solid black. The maximum weight matching is shown in blue. We assume a weight function ω with $\omega(u, v) = 1$ for all $(u, v) \in V_G \times V_H$ and $\omega(e, f) = 0$ for all $(e, f) \in E_G \times E_H$. The edges without label in (c) have weight 1.

3 Problem Decomposition and Fundamental Algorithms

We introduce the basic techniques for solving MCST following the ideas of Edmonds and Matula [15]. The approach requires to compute MWMs in bipartite graphs as a subroutine. We discuss the occurring matching instances in detail in Section 4.

By fixing the roots of both trees we can develop an algorithm solving MCST on this restricted setting. It is easy to generalize this solution by considering all possible pairs of roots. We then show that it is sufficient to fix the root of one tree while still obtaining a maximum solution.

Rooted trees. We first consider the problem restricted to rooted trees under the assumption that the roots of the two trees must be mapped to each other. For a rooted tree G^r we define the *rooted subtree* G^r_u as the subtree induced by u and all its descendants in G^r that is rooted at u, cf. Figures 1a and b. Note that $G^r_r = G^r$ and that G^r_u and G^s_u both refer to the same subtree unless s is contained in G^r_u. The key to solving MCST for two rooted trees G^r and H^s is the following recursive formulation:

$$\mathrm{MCS}_{\mathrm{root}}(G^r, H^s) = \omega(r, s) + W(M), \tag{1}$$

where M is an MWM of the complete bipartite graph on the vertex set $C(r) \uplus C(s)$ with weights $w(uv) = \omega(ru, sv) + \mathrm{MCS}_{\mathrm{root}}(G^r_u, H^s_v)$ for all $u \in C(r)$ and $v \in C(s)$. Hence, each edge weight corresponds to the solution of a problem of the same type for a pair of smaller rooted subtrees and the recursion naturally stops at the leaves. Each subproblem, the initial one as well as those arising in recursive calls, is uniquely defined by a pair of rooted subtrees and essentially consists of solving a matching instance.

Figure 1 illustrates the two rooted subtrees G^r_u and H^s_v and the corresponding matching problem under the weight function as given in the figure. For rooted trees G^r and H^s this problem arises on the second level in the recursion of Eq. (1). We obtain $\mathrm{MCS}_{\mathrm{root}}(G^r_u, H^s_v) = \omega(u, v) + W(M) = 1 + 5 = 6$, where M is an MWM of Figure 1c, depicted in Figure 1d.

In order to compute Eq. (1) the subproblems defined by the pairs of rooted subtrees $\mathcal{S}_{\mathrm{root}}(G^r, H^s) := \{(G^r_u, H^s_v) \mid \mathrm{depth}(u) = \mathrm{depth}(v)\}$ have to be solved.

▶ **Proposition 1.** *A maximum common subtree isomorphism for two rooted trees G^r and H^s can be computed in time $\mathcal{O}(n^3)$, where $n = |G| + |H|$.*

Proof. The bipartite graph for the subproblem (G^r_u, H^s_v) contains $k_u + l_v$ vertices, where $k_u := |C(u)|$ and $l_v := |C(v)|$. For the total running time we distinguish the cases $k_u \le l_v$

and $k_u > l_v$. For the first case we obtain a MWM in time $\mathcal{O}(k_u l_v(k_u + \log l_v))$ according to Lemma 5. The second case is analog. Since $\mathcal{S}_{\text{root}}(G^r, H^s) \subseteq \{(G^r_u, H^s_v) \mid u \in V_G, v \in V_H\}$ the total running time is bounded by $\mathcal{O}(n^3)$ as

$$\sum_{u \in V_G} \sum_{v \in V_H, k_u \leq l_v} k_u l_v(k_u + \log l_v) \leq \sum_{u \in V_G} k_u \sum_{v \in V_H} l_v(l_v + \log l_v) \leq n \cdot 2n^2 \in \mathcal{O}(n^3)$$

◀

Unrooted trees. We now consider the problem for unrooted trees. An immediate solution is to solve the rooted problem variant for all possible pairs of roots, i.e., by computing

$$\text{MCS}(G, H) := \max \{\text{MCS}_{\text{root}}(G^r, H^s) \mid r \in V(G), s \in V(H)\}. \tag{2}$$

Clearly, this yields the optimal solution in time $\mathcal{O}(n^5)$ with Proposition 1. Note that several recursive calls involve solving the same subproblem. Repeated computation can easily be avoided by means of a lookup table. Let $\text{Rt}(G^r) := \{G^r_u \mid u \in V(G)\}$ and $\text{Rt}(G) := \bigcup_{r \in V(G)} \text{Rt}(G^r)$. Note that we may uniquely associate the subtree G^r_u with G^p_u, where p is the parent of u in G^r. Hence, each rooted subtree $G^u_v \in \text{Rt}(G)$ either is the whole tree G with root $u = v$ or is the subtree rooted at v of some edge $uv \in E(G)$, where u is not contained in the subtree. Thus, $\text{Rt}(G) = \{G^u_v \mid v \in V(G) \wedge u \in N(v) \cup \{v\}\}$ is the set of all rooted subtrees of G. In total the subproblems defined by $\mathcal{S}(G, H) := \text{Rt}(G) \times \text{Rt}(H)$ have to be solved.

However, ensuring that each subproblem is solved only once does not allow to improve the bound on the running time, since $\mathcal{S}(G, H)$ still may contain a quadratic number of subproblems of linear size: Let G and H be two star graphs on n vertices, i.e., trees with all but one vertex of degree one. Each of the $(n-1)^2$ pairs of leaves can be selected as root pair and leads to a different subproblem of size $n - 1$.

We show that it is sufficient to consider only a subset of the subproblems to guarantee that an optimal solution is found. Let

$$\text{MCS}_{\text{fast}}(G^r, H) := \max \{\text{MCS}_{\text{root}}(G^r_u, H^s) \mid u \in V(G), s \in V(H)\}, \tag{3}$$

where $r \in V(G)$ is an arbitrary but fixed root of G. To compute Eq. (3), only the subproblems $\mathcal{S}_{\text{fast}}(G^r, H) := \text{Rt}(G^r) \times \text{Rt}(H) \subseteq \mathcal{S}(G, H)$ need to be solved.

▶ **Lemma 2.** *Let* MCS_{fast} *and* MCS *be defined as above and* $r \in V(G)$ *arbitrary but fixed, then* $\text{MCS}_{\text{fast}}(G^r, H) = \text{MCS}(G, H)$ *for all trees* G, H.

Proof. Let ϕ be an MCSI. If r is in the domain of ϕ, then $\omega(\phi) = \text{MCS}_{\text{root}}(G^r, H^{\phi(r)}) = \text{MCS}_{\text{fast}}(G^r, H)$. Otherwise the domain of ϕ is contained in the subtree rooted at one child of r. Let u be the unique vertex that is closest to r and mapped by ϕ. Then $\omega(\phi) = \text{MCS}_{\text{root}}(G^r_u, H^{\phi(u)}) = \text{MCS}_{\text{fast}}(G^r, H)$. ◀

Algorithm 1 implements this strategy, where the postorder traversal on G^r (line 2) ensures that the solutions to smaller subproblems are always available when required (line 9). The lookup table contains one entry for each subproblem in $\mathcal{S}_{\text{fast}}(G^r, H)$ and hence requires space $\mathcal{O}(n^2)$. Note that it is also possible to compute a concrete isomorphism from the MWMs associated with the computed optimal solution. The restriction of the considered subproblems allows to improve the bound on the running time.

▶ **Proposition 3.** *Algorithm 1 solves the maximum common subtree isomorphism problem for two trees* G *and* H *in time* $\mathcal{O}(n^4)$*, where* $n = |G| + |H|$.

```
Algorithm 1: Maximum Common Subtree Isomorphism
   Input  : Trees $G$ and $H$ under a weight function $\omega$
   Output : Weight of an MCSI between $G$ and $H$.
   Data   : Table $D(u, s, v)$ storing solutions $\mathrm{MCS}_{\mathrm{root}}(G^r_u, H^s_v)$ of subproblems.
 1 Select an arbitrary root vertex $r \in V_G$.
 2 foreach $u \in V_G$ in postorder traversal on $G^r$ do          ▷ All possible $G^r_u \in \mathrm{RT}(G^r)$
 3     $U \leftarrow C(u)$ in $G^r$
 4     foreach $v \in V_H$ do
 5         foreach $s \in N(v) \cup \{v\}$ do                     ▷ All possible $H^s_v \in \mathrm{RT}(H)$
 6             $V \leftarrow C(v)$ in $H^s$
 7             if $\omega(u, v) \neq -\infty$ then
 8                 foreach pair $(u', v') \in U \times V$ do
 9                     $w(u'v') \leftarrow \omega(uu', vv') + D(u', v, v')$
10                 $M \leftarrow$ MWM of the complete graph on $U \uplus V$ with weights $w$.
11                 $D(u, s, v) \leftarrow \omega(u, v) + W(M)$
12             else $D(u, s, v) \leftarrow -\infty$
13 return the maximum entry in $D$
```

Proof. According to Lemma 2 computing Eq. (3) yields the optimal solution and it suffices to solve the subproblems $\mathcal{S}_{\text{fast}}(G^r, H)$ as realized by Algorithm 1. Let k_u be the number of children of u in G^r_u, l^s_v the number of children of v in H^s, $s \in V(H)$, and $l_v = |N(v)|$. For all s we have $l^s_v \leq l_v$. Similar to Proposition 1 the subproblems $\mathcal{S}_{\text{fast}}(G, H)$ can be solved in a total time of

$$\mathcal{O}\left(\sum_{u \in V_G} \sum_{s \in V_H} \sum_{v \in V_H} (k_u l^s_v)(\min\{k_u, l^s_v\} + \log \max\{k_u, l^s_v\})\right) \subseteq \mathcal{O}\left(\sum_{s \in V_H} n^3\right) \subseteq \mathcal{O}\left(n^4\right).$$

◀

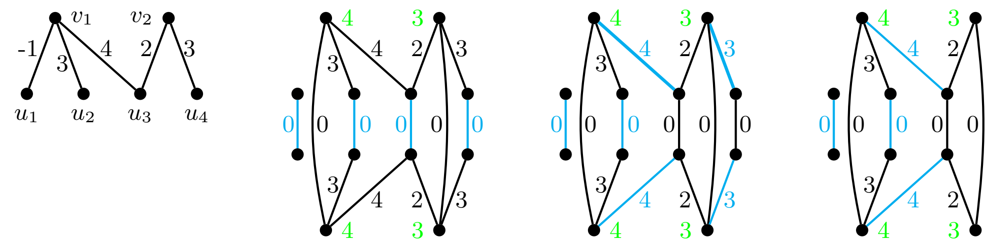
Further improvement of the running time is possible by no longer considering the MWM subroutine as a black box. We pursue this direction in the next section. Our findings there yield the following theorem.

▶ **Theorem 4.** *An MCSI between two unrooted trees G and H can be computed in time $\mathcal{O}(|G||H|(\min\{\Delta(G), \Delta(H)\} + \log \max\{\Delta(G), \Delta(H)\}))$.*

Proof. The MWM computations in Algorithm 1 are dominating, thus we obtain the above running time directly from Theorem 7 of the following section. ◀

4 Computing All Maximum Weight Matchings

In this section we improve the total time bound for solving all the matching instances arising in Algorithm 1. First, we provide a time bound to compute an MWM in a single bipartite graph $(V \uplus U, E)$, where possibly $|V| \neq |U|$. In the following, we exploit the fact that during the run of our algorithm, we get sets of "similar" bipartite graph instances. After computing an MWM on one graph in one of the sets, we can derive MWMs for all the other bipartite graphs in that set very efficiently. Finally, we provide an upper bound to compute an MWM in all the occurring bipartite graphs.



(a) Input graph B. **(b)** Reduced graph B'. **(c)** MWMs M', M. **(d)** B'_4 with M'_4.

Figure 2 Weighted bipartite graph B (a); reduced graph B' with initial duals in green (vertices without label have dual value 0) and initial matching M'' in blue (b); MWM M' of B' in blue, M of B in thick blue c; B'_4 with matching M'_4 in blue (d). - cf. proofs of Lemma 5 and Lemma 6.

Computing an MWM is closely related to finding an MWPM and there is extensive literature on both problems [8]. Gabow and Tarjan [10] describe a reduction to solve the MWM problem using any algorithm for MWPM, without altering the algorithm's asymptotic time bound, which we will make use of. For computing an MWPM, we use the well known Hungarian method, which has at most n iterations in its outer loop and a total running time of $\mathcal{O}(n^3)$ or $\mathcal{O}(n(m + n \log n))$ using Fibonacci heaps, where n and m denote the number of vertices and edges of the bipartite graph. We denote this algorithm by $\mathcal{A}_{\text{PM}}$.

The Hungarian method is a primal-dual algorithm. It starts with an empty matching and computes a new matching with one more edge in each iteration, maintaining a feasible dual solution of a primal linear program. The complementary slackness theorem ensures, that the obtained perfect matching after n iterations is a MWPM. Note, by using the reduction in [10], we always have at least one perfect matching.

▶ **Lemma 5.** *Let $B = (V \uplus U, E)$ be a bipartite graph with edge weights $w : E \to \mathbb{R}$. Let $k := |V|$, $l := |U|$, and $k \leq l$. An MWM M on B can be computed in time $\mathcal{O}(kl(k + \log l))$.*

Proof. Let $\{v_1, \ldots, v_k\} = V, \{u_1, \ldots, u_l\} = U$ be the two vertex sets of B. First, we remove all edges from B with negative edge weight, because they never contribute to an MWM. Then, we add a copy B^{C} of B to the graph. For each vertex $v \in V \uplus U$ we denote its copy v^{C} and for each edge $e \in E$ we denote its copy e^{C}. We then copy the edge weights, i.e., $w(e^{\text{C}}) := w(e)$ for each edge $e \in E$. Next we insert a new edge of weight 0 between each vertex $v \in V \uplus U$ and its copy v^{C}. This graph is called *reduced graph* B'. Figures 2a and b show an example of B and B'. An MWPM M' of B' yields an MWM M of B: $vu \in M \Leftrightarrow vu \in M'$ and $v \in V, u \in U$. This follows from the construction of B'.

In the following, we prove an upper time bound to compute M'. An initial feasible dual solution $d : V_{B'} \to \mathbb{R}$, i.e., $d(v) + d(u) \geq w(vu)$ for all edges $vu \in E_{B'}$, including l matching edges vu with $d(v) + d(u) = w(vu)$, is computed as follows: We set $d(u) = 0$ for all $u \in U$ and $d(v) := \max\{w(vu) \mid u \in U\}$ for all $v \in V$. Next, for each $v \in V \uplus U$ the vertex v^{C} obtains the dual value $d(v^{\text{C}}) := d(v)$. We define an initial matching $M'' := \{uu^{\text{C}} \mid u \in U\}$. Note, $d(u) + d(u^{\text{C}}) = 0 = w(uu^{\text{C}})$.

The dual solution d is feasible and can be computed in time $\mathcal{O}(kl)$. Let $n := |V_{B'}| = 2(k+l) \in \Theta(l)$ and $m := |E_{B'}| \leq 2kl+k+l \in \mathcal{O}(kl)$. Increasing the number of matching edges by one using a single iteration of $\mathcal{A}_{\text{PM}}$ is possible in time $\mathcal{O}(m + n \log n) = \mathcal{O}(l(k + \log l))$. To obtain an MWPM M' form M'' in B' we need to increase the number of matching edges by k, therefore the time to compute M' and thus M is $\mathcal{O}(kl(k + \log l))$. ◀

Algorithm 2: Computing MWMs on B and B_j, cf. Lemmas 5, 6

```
Input  : Bipartite graph $B = (V \uplus U, E), |U| \geq 2, U = \{u_1, u_2, \ldots\}$ with edge weights
         $w : E \to \mathbb{R}$
Output : MWMs $M, M_j$ on $B, B_j := G[V \uplus U \setminus \{u_j\}]$ for each $j \in [1..|U|]$.
if $|V| \leq |U|$ then                                          ▷ Compute MWM $M$ of $B$
    Let $B' := (V', E')$, where $V' := V \cup U \cup \{v^{\mathrm{C}} \mid v \in V \cup U\}$ and
    $E' := E \cup \{e^{\mathrm{C}} \mid e \in E\} \cup \{vv^{\mathrm{C}} \mid v \in V \cup U\}$.
    $w(e^{\mathrm{C}}) \leftarrow w(e)$ for all $e \in E$              ▷ Weights of additional edges
    $w(vv^{\mathrm{C}}) \leftarrow 0$ for all $v \in V \cup U$
    $d(u^{\mathrm{C}}) \leftarrow d(u) \leftarrow 0$ for all $u \in U$                  ▷ Dual values
    $d(v^{\mathrm{C}}) \leftarrow d(v) \leftarrow \max\{w(vu) \mid u \in U\}$ for all $v \in V$
    $M'' \leftarrow \{uu^{\mathrm{C}} \mid u \in U\}$                       ▷ Initial matching edges
    Starting with $M''$ and $d$, compute an MWPM $M'$ on $B'$ using $|V|$ iterations of $\mathcal{A}_{\mathrm{PM}}$
    $M \leftarrow \{vu \mid vu \in M', v \in V, u \in U\}$
else
    Exchange the vertices of $V$ and $U$.
    Compute $M$ as in lines 2 to 9 and exchange $V$ and $U$ back.
$d \leftarrow$ The dual values obtained while computing $M'$.
foreach $j \in [1..|U|]$ do                                    ▷ MWMs $M_j$ on $B_j$
    if $u_j$ is not matched by $M$ then
        $M_j \leftarrow M$
    else
        $B'_j \leftarrow B' \setminus \{u_j, u_j^{\mathrm{C}}\}$
        $M'_j \leftarrow M'$ without the matching edges incident to $u_j, u_j^{\mathrm{C}}$   ▷ Initial matching
        Compute an MWPM $M'_j$ on $B'_j$ using $d$ and a single iteration of $\mathcal{A}_{\mathrm{PM}}$.
        $M_j \leftarrow \{vu \mid vu \in M'_j, v \in V, u \in U\}$
```

▶ **Lemma 6.** *Let $B = (V \uplus U, E)$ be a weighted bipartite graph with $k := |V|, U = \{u_1, \ldots, u_l\}, l \geq 2$. Let $B_j := G[V \uplus U \setminus \{u_j\}]$ for each $j \in [1..l]$. Computing MWMs for all graphs $B, B_1, \ldots, B_l$ is possible in total time $\mathcal{O}(kl(\min\{k,l\} + \log \max\{k,l\}))$.*

Proof. According to Lemma 5 we obtain an MWM M of B in time $\mathcal{O}(kl(\min\{k,l\} + \log \max\{k,l\}))$. We compute an MWM on each B_j as follows: Let d be an optimal dual solution obtained while computing M' (on B', see proof of Lemma 5). If u_j is not matched by M, i.e., $u_j \notin e$ for all $e \in M$, then $M_j := M$ is an MWM of B_j. Otherwise let B'_j be the reduced graph as explained in the proof of Lemma 5. We obtain a feasible dual solution d_j on the bipartite graph B'_j by taking the dual values from d, i.e., $d_j(v) := d(v)$ for all $v \in V(B'_j)$. Note, we have $2(k + l)$ vertices in B', and exactly two less in B'_j, i.e., a perfect matching in B'_j consists of $k + l - 1$ matching edges.

We can derive an initial matching M'_j on B'_j with $k+l-2$ edges from M'; M'_j contains the matching edges that are not incident to the two removed vertices from B' to B'_j. Therefore only one more iteration of $\mathcal{A}_{\mathrm{PM}}$ is needed, which is possible in time $\mathcal{O}(\max\{k,l\}(\min\{k,l\} + \log \max\{k,l\}))$, cf. proof of Lemma 5. We then obtain M_j from M'_j as previously described. The complementary slackness conditions ensure M'_j and therefore M_j is of maximum weight. An example of M' and B'_j ($j = 4$) is shown in Figures 2c and d.

We need to compute an MWM different from M for at most $\min\{k,l\}$ of the l graphs $B_1, \ldots, B_l$, because at most k vertices of U are matched by M, cf. Figure 2c: only u_3

and u_4 of U are matched by M. Therefore the time bound to compute MWMs for all the graphs $B_1, \ldots, B_l$ is $\mathcal{O}(\min\{k,l\}\max\{k,l\}(\min\{k,l\} + \log\max\{k,l\})) = \mathcal{O}(kl(\min\{k,l\} + \log\max\{k,l\}))$. ◀

We call B and the graphs $B_j, j \in [1..l]$, a set of "similar" bipartite graph instances. Algorithm 2 shows how we compute an MWM for each graph in this set. Next, we apply Lemma 6 to Algorithm 1. For each pair $u \in V_G, v \in V_H$ of vertices, selected in line 2 and 4, respectively, the algorithm computes up to $|N(v)| + 1$ MWMs, cf. lines 5, 10. A close look at Algorithm 1 reveals this as a set of "similar" bipartite graph instances. The first graph B is obtained by selecting $s = v$ in line 5. The other graphs B_j are obtained by selecting all the vertices $s \in N(v)$. This observation allows to prove the following theorem.

▶ **Theorem 7.** *All the MWMs in Algorithm 1 can be computed in total time* $\mathcal{O}(kl(\min\{\Delta(G), \Delta(H)\} + \log\max\{\Delta(G), \Delta(H)\}))$*, where* $k = |G|$ *and* $l = |H|$*.*

Proof. For each pair $(v, u) \in V_G \times V_H$ we compute an MWM on each of the "similar" graphs, where $B = (C(v) \uplus N(u), E)$ and edge weights as determined by Eq. (1). Let $d_{\min} := \min\{\Delta(G), \Delta(H)\}$ and $d_{\max} := \max\{\Delta(G), \Delta(H)\}$. For all the pairs (v, u) we obtain a time complexity of

$$\begin{aligned}
&\mathcal{O}\left(\sum_v \sum_u \delta(v)\delta(u)(\min\{\delta(v), \delta(u)\} + \log\max\{\delta(v), \delta(u)\})\right)\\
\subseteq\ &\mathcal{O}\left(\sum_v \delta(v) \sum_u \delta(u)(d_{\min} + \log d_{\max})\right)\\
=\ &\mathcal{O}\left((d_{\min} + \log d_{\max}) \sum_v \delta(v) l\right)\\
=\ &\mathcal{O}((d_{\min} + \log d_{\max})kl).
\end{aligned}$$

◀

5 Lower Bounds on the Time Complexity and Optimality

Providing a tight lower bound on the time complexity of a problem is generally a non-trivial task. We obtain this for trees of bounded degree and reason why the existence of an algorithm with subcubic running time for unrestricted trees is unlikely. In order to solve MCST with an arbitrary weight function ω for two trees G and H, all values $\omega(u, v)$ for $u \in V(G)$ and $v \in V(H)$ must be considered. This directly leads the lower bound of $\Omega(|G||H|)$ for the time complexity of MCST. For trees of bounded degree our approach achieves running time $\mathcal{O}(|G||H|)$ according to Theorem 4 and, thus, has an optimal worst-case running time in the considered setting.

For unrestricted trees of order n our approach has a running time of $\mathcal{O}(n^3)$ according to Theorem 4. In the next paragraph we present a linear time reduction from the assignment problem to MCST, which preserves the time complexity. Therefore solving MCST in time $o(n^3)$ yields an algorithm to solve the assignment problem in time $o(n^3)$. The Hungarian method solves the assignment problem in $\mathcal{O}(n^3)$, which is the best known time bound for bipartite graphs with $\Theta(n^2)$ edges of unrestricted weight for more than 30 years.

Let $B = (U \uplus V, E, w)$ be a weighted bipartite graph on which we want to solve the assignment problem, i.e., to find an MWPM. We assume weights to be non-negative, which can be achieved by adding a sufficiently large constant to every edge weight to obtain an assignment problem that is equivalent w.r.t. the MWPMs. We construct a star graph G with center c and leaves U and another star graph H with center c' and leaves V. Let $n = |U| = |V|$ and $N = \max_{e\in E} w(e)$. We define ω such that $\omega(u, v) = w(uv) + nN$ for all $uv \in E$, $\omega(c, c') = nN$ and $\omega(u, v) = -\infty$ for all other pairs of vertices. For all pairs of edges we define $\omega(e, e') = 0$. Let ϕ be an MCSI between G and H w.r.t. w and $p := |\operatorname{dom}(\phi)|$. It directly follows from the construction that $M := \{uv \in E \mid \phi(u) = v\}$ is an MWM in B with $W(M) = \mathcal{W}(\phi) - pnN$. Furthermore, the incremented weights ensure that M is perfect, i.e., $p = n + 1$, whenever B admits a perfect matching. Therefore we obtain:

▶ **Proposition 8.** *Only if we can solve the assignment problem on a graph with n vertices and $\Theta(n^2)$ edges of unrestricted weight in time $o(n^3)$, we can solve MCST on two unrooted trees of order $\Theta(n)$ in time $o(n^3)$.*

6 Output-Sensitive Algorithms for Listing All Solutions

Algorithm 1 can easily be modified to not only output the weight of an MCSI, but also an associated isomorphism. Let $D(u, s, v)$ be a maximum entry in D. Then $\phi(u) = v$. Further mappings are defined by the matching edges occurring in Eq. (1). In the example of Figure 1d we obtain $\phi(c_1) = d_1$ and $\phi(c_2) = d_3$. Since in general there is no single unique MCSI, it is of interest to find and list all of them. In this section we show how our techniques can be combined with the enumeration algorithm from [7], which lists all the different MCSIs of two trees exactly once. We obtain the best known time bound for listing all solutions by an improved analysis.

Since the number of MCSIs is not polynomially bounded in the size of the input trees, we cannot expect polynomial running time. An algorithm is said to be *output-sensitive* if its running time depends on the size of the output in addition to the size of the input.

The basic idea to enumerate all MCSIs is to first compute the weight of an MCSI. Then for each maximum table entry $D(u, v, v), u \in V_G, v \in V_H$, all the different rooted MCSIs on the rooted subtrees G^r_u, H^v_v are listed. Note, we omit maximum table entries $D(u, s, v)$, where $s \neq v$. We do this, because every MCSI of G^r_u, H^s_v is also an MCSI of G^r_u, H^v_v. As an example let u be the root of G in Figure 1. Then $D(u, v, v) = D(u, s, v) = 7$. For both table entries we obtain the same MCSI ϕ with $\phi(u) = v, \phi(r) = d_1, \phi(c_1) = d_2, \phi(c_2) = d_3, \ldots$.

We enumerate the MCSIs on a pair of rooted subtrees by enumerating all MWMs of the associated bipartite graphs of Eq. (1) and then expanding ϕ recursively along all the different MWMs of the mapped children. For the problem depicted in Figure 1c there are two different MWMs: $M_1 = \{c_1d_1, c_2d_3\}$ and $M_2 = \{c_1d_2, c_2d_3\}$. Therefore we first expand along M_1 as explained in the first paragraph of this section and then along M_2. We do this recursively for each occurring matching instance. The enumerated isomorphisms of each maximum entry are pairwise different, based on the different MWMs. They are also pairwise different between two different maximum entries. The proof of the latter claim is similar to the proof of Lemma 2. Thus we do not enumerate an MCSI twice. Further we do not omit an MCSI, because we consider all necessary maximum table entries and their rooted subtrees, as well as all possible expansions along the MWMs.

Note, the enumeration algorithm of [7] uses a somewhat different table to store maximum solutions. The basic idea to list all solutions is the same. For trees of sizes $k := |G|$ and $l := |H|$, $k \le l$, their enumeration algorithm requires total time $\mathcal{O}(k^2l^4 + Tl^2)$, where T is the number of different MCSIs. The $\mathcal{O}(k^2l^4)$ term of the running time is caused by computing the weight of an MCSI in time $\mathcal{O}(kl^4)$ and repeated deletions of single edges in one tree and recalculations of the weight of an MCSI to avoid outputting an MCSI twice. We have improved the time bound to compute the weight of an MCSI, cf. Theorem 4. Therefore we can improve the $\mathcal{O}(k^2l^4)$ term to $\mathcal{O}(kl(\min\{\Delta(G), \Delta(H)\} + \log \max\{\Delta(G), \Delta(H)\}))$.

The $\mathcal{O}(Tl^2)$ term in the original running time is caused by the enumeration of MWMs. For each MCSI ϕ several MWMs have to be enumerated, let this number be m_ϕ. The time to do this can be bounded by $\mathcal{O}(l^2)$, when using a variant of the enumeration algorithm for perfect matchings presented in [21]. The running time follows from the fact, that for each MWM two depth first searches (DFS) in a directed subgraph of B', cf. Figure 2, are computed. The running time of DFS is linear in the number of edges and vertices. Let k_i, l_i be the sizes of the disjoint vertex sets of the i-th bipartite graph, on which we enumerate the MWMs, $i \in [1..m_\phi]$. Then $\sum_i k_i \leq k$ and $\sum_i l_i \leq l$, because all the vertices in all the m_ϕ bipartite graphs are pairwise disjoint. The running time of DFS in the directed subgraphs of the i-th bipartite graph is $\mathcal{O}(k_il_i)$, cf. Figure 2b or d. For all m_ϕ DFS runs we have $\sum_i k_il_i \leq \sum_i k_i\Delta(H) \leq k\Delta(H)$ as well as $\sum_i k_il_i \leq \sum_i \Delta(G)l_i \leq \Delta(G)l$. Hence, the time to enumerate ϕ is bounded by $\mathcal{O}(\min\{k\Delta(H), \Delta(G)l\})$.

Both improvements combined together, the initial computation of the weight of an MCSI and the MWM enumeration, improve the enumeration time from $\mathcal{O}(n^6 + Tn^2)$ to $\mathcal{O}(n^3 + Tn \min\{\Delta(G), \Delta(H)\})$. More precisely we obtain the following theorem.

▶ **Theorem 9.** *Enumerating all MCSIs of two unrooted trees G and H is possible in time $\mathcal{O}(\,|G|\,|H|\,(\,\min\{\Delta(G), \Delta(H)\} + \log \max\{\Delta(G), \Delta(H)\}\,) + T(\,\min\{|G|\Delta(H), \Delta(G)|H|\}\,)\,)$, where T is the number of different MCSIs.*

7 Experimental Comparison

In this section we experimentally evaluate the running time of our approach (DKM) on synthetic and real-world instances. We compare our algorithm to the approach of [18] which also solves MCST when the input graphs are trees. The corrected analysis of the approach yields a running time of $\mathcal{O}(n^4)$, which aligns better with our experimental findings of $\Omega(n^5)$. The implementation was provided by the authors as part of the FOG package.[2] Both algorithms were implemented in C++ and compiled with GCC v.4.8.4. Running times were measured on an Intel Core i7-3770 CPU with 16 GB of RAM using a single core only. We generated random trees by iteratively adding edges to a randomly chosen vertex and averaged over 40 to 100 pairs of instances depending on their size. The weight function ω was set to 1 for each pair of vertices and edges, i.e., we compute isomorphisms of maximum size. This matches the setting in FOG.

Table 1 summarizes our results and we observe that the running time of our approach aligns with our theoretical analysis. In comparison, FOG's running time is much higher and increases to a larger extent with the input size. The running times of both algorithms show a low standard deviation for random trees, cf. Tables 1a, b. Table 1c shows the running time in star graphs, which are worst-case examples for some approaches, see Sec. 3. Our theoretical proven cubic running time matches the experimental results, while FOG's running time increases drastically. Table 1d summarizes the computation time under different weight functions. We defined ω such that different labels are simulated, i.e., vertices and edges with different labels have weight $-\infty$, which again matches FOG's setting. Both algorithms clearly benefit from the fact that less MWMs have to be computed. The results on random trees are also shown in Figure 3.

[2] https://dtai.cs.kuleuven.be/software/PMCSFG

Table 1 Average running time in ms ± RSD in % and speedup factor $q := \text{FOG}/\text{DKM}$.

(a) Random trees of the same order.

Order	DKM	FOG	q
20	$0.9 \pm 8\%$	$40 \pm 7\%$	44.1
40	$3.5 \pm 6\%$	$221 \pm 5\%$	62.7
80	$15.2 \pm 4\%$	$1\,286 \pm 5\%$	84.8
160	$58.9 \pm 3\%$	$8\,342 \pm 5\%$	141.7
320	$237.4 \pm 2\%$	$63\,327 \pm 8\%$	266.9

(b) Random trees with $|G| = 80$ fixed.

$\|H\|$	DKM	FOG	q
20	$3.6 \pm 8\%$	$192 \pm 4\%$	53.6
40	$7.3 \pm 7\%$	$504 \pm 4\%$	68.7
80	$15.2 \pm 4\%$	$1\,286 \pm 5\%$	84.8
160	$30 \pm 9\%$	$3080 \pm 4\%$	103.3
320	$59.5 \pm 3\%$	$6842 \pm 4\%$	114.9

(c) Star graphs.

Order	DKM	FOG	q
10	0.1	18	117.6
20	1	489	458.5
40	8.9	18 722	2109.9
80	77.5	929 784	11 992.1

(d) Different ω-functions, order 80

#labels	DKM	FOG	q
1	$15.2 \pm 4\%$	$1\,286 \pm 5\%$	84.8
2	$5.4 \pm 8\%$	$217 \pm 8\%$	40
3	$3.3 \pm 7\%$	$118 \pm 12\%$	36.1
4	$2.6 \pm 8\%$	$83 \pm 9\%$	31.9

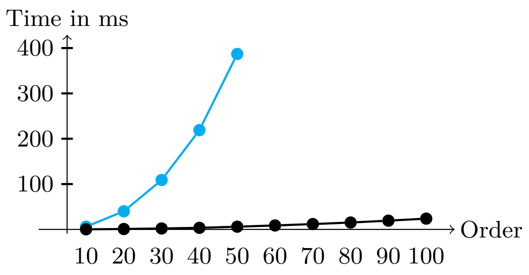


Figure 3 Average running time in ms (y-axis) for MCSI computation on random trees of order n (x-axis). Black = Our implementation (DKM). Blue = FOG implementation.

From a chemical database of thousands of molecules[3] we extracted 100 pairs of graphs with block-cut trees (BC-trees) consisting of more than 40 vertices. BC-trees are a representation of graphs, where each maximal biconnected component is represented by a B-vertex. If two such components share a vertex, the corresponding B-vertices are connected through a C-vertex representing this shared vertex. The running time for MCST on BC-trees is an important factor for the total running time of MCS algorithms for outerplanar and series-parallel graphs like [2, 13, 18]. The average running time of our algorithm was 11.2 ms, compared to FOG's 481.3 ms. The speedup factor ranges from 24 to 59, with an average of 43. This indicates that the above mentioned approaches could greatly benefit from the techniques presented in this paper.

8 Conclusions

We have presented a novel algorithm for MCST which (i) considers only the subproblems required to guarantee that an optimal solution is found and (ii) solves groups of related matching instances efficiently in one pass. Rigorous analysis shows that the approach achieves cubic time in general trees and quadratic time in trees of bounded degree. Our analysis of the

[3] NCI Open Database, GI50, `http://cactus.nci.nih.gov`

problem complexity reveals that there is only little room for possible further improvements. The practical efficiency is documented by an experimental comparison.

If the weight function is restricted to integers of a bounded value, scaling approaches [8] to the corresponding matching problems become applicable. It remains future work to improve the running time for this case.

References

1 Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1256–1271. SIAM, 2016.

2 Tatsuya Akutsu and Takeyuki Tamura. A polynomial-time algorithm for computing the maximum common connected edge subgraph of outerplanar graphs of bounded degree. *Algorithms*, 6(1):119–135, 2013. doi:10.3390/a6010119.

3 Tatsuya Akutsu, Takeyuki Tamura, Avraham A. Melkman, and Atsuhiro Takasu. On the complexity of finding a largest common subtree of bounded degree. In Leszek Gasieniec and Frank Wolter, editors, *Fundamentals of Computation Theory*, volume 8070 of *LNCS*, pages 4–15. Springer, 2013. doi:10.1007/978-3-642-40164-0_4.

4 Moon Jung Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, 8(1):106–112, 1987. doi:10.1016/0196-6774(87)90030-7.

5 Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 2004. doi:10.1142/S0218001404003228.

6 Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1):2:1–2:19, December 2009. doi:10.1145/1644015.1644017.

7 Andre Droschinsky, Bernhard Heinemann, Nils Kriege, and Petra Mutzel. Enumeration of maximum common subtree isomorphisms with polynomial-delay. In Hee-Kap Ahn and Chan-Su Shin, editors, *Algorithms and Computation (ISAAC)*, LNCS, pages 81–93. Springer, 2014. doi:10.1007/978-3-319-13075-0_7.

8 Ran Duan and Hsin-Hao Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1413–1424. SIAM, 2012.

9 Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011. doi:10.1002/wcms.5.

10 Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1987.

11 Arvind Gupta and Naomi Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21:183–210, 1998. doi:10.1007/PL00009212.

12 Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40(2):212–233, 2001. doi:10.1006/jagm.2001.1163.

13 Nils Kriege, Florian Kurpicz, and Petra Mutzel. On maximum common subgraph problems in series-parallel graphs. In Kratochvíl Jan, Mirka Miller, and Dalibor Froncek, editors, *IWOCA 2014*, volume 8986 of *LNCS*, pages 200–212. Springer, 2014. doi:10.1007/978-3-319-19315-1_18.

14 Nils Kriege and Petra Mutzel. Finding maximum common biconnected subgraphs in series-parallel graphs. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, edit-

ors, *MFCS 2014*, volume 8635 of *LNCS*, pages 505–516. Springer, 2014. doi:10.1007/978-3-662-44465-8_43.

15 David W. Matula. Subtree isomorphism in $O(n^{5/2})$. In P. Hell B. Alspach and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91–106. Elsevier, 1978. doi:10.1016/S0167-5060(08)70324-8.

16 Matthias Rarey and J.Scott Dixon. Feature trees: A new molecular similarity measure based on tree matching. *Journal of Computer-Aided Molecular Design*, 12(5):471–490, 1998. doi:10.1023/A:1008068904628.

17 Steven W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM J. Comput.*, 6(4):730–732, 1977.

18 Leander Schietgat, Jan Ramon, and Maurice Bruynooghe. A polynomial-time maximum common subgraph algorithm for outerplanar graphs and its application to chemoinformatics. *Annals of Mathematics and Artificial Intelligence*, 69(4):343–376, 2013. doi:10.1007/s10472-013-9335-0.

19 Ron Shamir and Dekel Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999. doi:10.1006/jagm.1999.1044.

20 A. Torsello, D. Hidovic-Rowe, and M. Pelillo. Polynomial-time metrics for attributed trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1087–1099, July 2005. doi:10.1109/TPAMI.2005.146.

21 Takeaki Uno. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In *Algorithms and Computation (ISAAC)*, volume 1350 of *LNCS*, pages 92–101. Springer, 1997.

22 Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.

23 Rakesh M. Verma and Steven W. Reyner. An analysis of a good algorithm for the subtree problem, corrected. *SIAM J. Comput.*, 18(5):906–908, 1989.

A Single-Exponential Fixed-Parameter Algorithm for Distance-Hereditary Vertex Deletion*

Eduard Eiben[1], Robert Ganian[2], and O-joung Kwon[3]

1 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
2 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
3 **Institute for Computer Science and Control, Hungarian Academy of Sciences, Budapest, Hungary.**

Abstract

Vertex deletion problems ask whether it is possible to delete at most k vertices from a graph so that the resulting graph belongs to a specified graph class. Over the past years, the parameterized complexity of vertex deletion to a plethora of graph classes has been systematically researched. Here we present the first single-exponential fixed-parameter algorithm for vertex deletion to distance-hereditary graphs, a well-studied graph class which is particularly important in the context of vertex deletion due to its connection to the graph parameter rank-width. We complement our result with matching asymptotic lower bounds based on the exponential time hypothesis.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms – G.2.2 Graph Algorithms

Keywords and phrases distance-hereditary graphs, fixed-parameter algorithms, rank-width

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.34

1 Introduction

Vertex deletion problems include some of the best studied NP-hard problems in theoretical computer science, including Vertex Cover or Feedback Vertex Set. In general, the problem asks whether it is possible to delete at most k vertices from a graph so that the resulting graph belongs to a specified graph class. While these problems are studied in a variety of contexts, they are of special importance for the parameterized complexity paradigm [11, 9], which measures the performance of algorithms not only with respect to the input size but also with respect to an additional numerical parameter. Vertex deletion problems allow a highly natural choice of the parameter (specifically, k), and many vertex deletion problems are known to admit so-called *single-exponential fixed-parameter algorithms*, which are algorithms running in time $\mathcal{O}(c^k \cdot n^{\mathcal{O}(1)})$ for input size n and some constant c.

Over the past years, the parameterized complexity of vertex deletion to a plethora of graph classes has been systematically researched. However, there still remain a few important classes where the existence of a single-exponential fixed-parameter algorithm remains open. One such class has, until now, been the class of *distance-hereditary graphs* [17] (also called *completely separable graphs* [15]). Distance-hereditary graphs have several equivalent

* The authors acknowledge support by ERC Starting Grant PARAMTIGHT (No. 280152) and the Austrian Science Fund (FWF, projects P26696 and W1255-N23). Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.

characterizations; for instance, they are the graphs where every induced path is a shortest path. But perhaps the main reason why distance-hereditary graphs are particularly important in the context of vertex deletion problems is their connection to the structural parameter *rank-width* [24, 23]. While TREEWIDTH-t VERTEX DELETION[1] is known to admit a single-exponential fixed-parameter algorithm for every fixed t [12, 22], the existence of such algorithms for the analogous RANK-WIDTH-t VERTEX DELETION is a challenging open problem. Since distance-hereditary graphs are exactly the graphs of rank-width 1 [23], a single-exponential fixed-parameter algorithm for DISTANCE-HEREDITARY VERTEX DELETION represents the first step towards handling RANK-WIDTH-t VERTEX DELETION.

DISTANCE-HEREDITARY VERTEX DELETION
Instance : A graph G and an integer k.
Parameter : k.
Task : Is there a vertex set $Q \subseteq V(G)$ with $|Q| \leq k$ such that $G-Q$ is distance-hereditary?

The main result of this paper is an $\mathcal{O}(37^k \cdot |V(G)|^7(|V(G)| + |E(G)|))$-time algorithm for DISTANCE-HEREDITARY VERTEX DELETION, solving an open problem of Kanté, Kim, Kwon, and Paul [20]. The core of our approach exploits two distinct characterizations of distance-hereditary graphs: one by forbidden induced subgraphs (obstructions), and the other by admitting a special kind of split decomposition [7].

The algorithm can be conceptually divided into three parts. First, we use the well-known iterative compression technique [25] to reduce the problem to the easier DISJOINT DISTANCE-HEREDITARY VERTEX DELETION, where we assume that the instance additionally contains a certain form of advice to aid us in our computation. Specifically, this advice is a vertex deletion set S to distance-hereditary graphs which is disjoint from and slightly larger than the desired solution. Then we exhaustively apply two branching rules to simplify the given instance of DISJOINT DISTANCE-HEREDITARY VERTEX DELETION. At a high level, these branching rules allow us to assume that the resulting instance contains no small obstructions and furthermore that certain connectivity conditions hold on $G[S]$. Lastly, we compute the split decomposition of $G - S$ and exploit the properties of our instance G guaranteed by branching to prune the decomposition. In particular, we show that the connectivity conditions and non-existence of small obstructions mean that S must interact with the split decomposition of $G - S$ in a special way, and this allows us to identify irrelevant vertices in $G - S$. This is by far the most technically challenging part of the algorithm.

A more detailed explanation of our algorithm is provided in Section 3, after the definition of required notions. We complement this result with an algorithmic lower bound which rules out a subexponential fixed-parameter algorithm for DISTANCE-HEREDITARY VERTEX DELETION under well-established complexity assumptions.

The set of induced subgraph obstructions for distance-hereditary graphs consists of three small graphs, and induced cycles of length at least 5. We remark that Heggernes et al. [16] showed that the problem asking whether it is possible to delete k vertices so that the resulting graph has no induced cycles of length at least 5 is W[2]-hard. Therefore, one cannot simply obtain a single-exponential fixed-parameter algorithm for DISTANCE-HEREDITARY VERTEX DELETION using the problem of hitting induced cycles of length at least 5.

The paper is organized as follows. Section 2 contains the necessary preliminaries and notions required for our results. In Section 3, we set the stage for the process of simplifying

[1] TREEWIDTH-t VERTEX DELETION asks whether it is possible to delete k vertices so that the resulting graph has treewidth at most t.

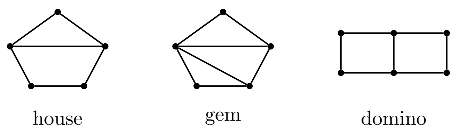


Figure 1 Small DH obstructions which are not cycles.

the split decomposition, which entails the definition of Disjoint Distance-Hereditary Vertex Deletion, introduction of our branching rules, and a few technical lemmas which will be useful throughout the later sections. Section 4 then introduces and proves the safeness of 8 polynomial-time reduction rules; crucially, the exhaustive application of these rules guarantees that the resulting instance will have a certain "inseparability" property. In Section 5, we introduce and prove the safeness of our final reduction rule using this inseparability property. Finally, the proof of our main result as well as the corresponding lower bound are presented in Section 6.

2 Preliminaries

For a graph G, let $V(G)$ and $E(G)$ denote the vertex set and the edge set of G, respectively. For $S \subseteq V(G)$, let $G[S]$ denote the subgraph of G induced on S. For $v \in V(G)$ and $S \subseteq V(G)$, let $G - v$ be the graph obtained from G by removing v, and let $G - S$ be the graph obtained by removing all vertices in S. For $v \in V(G)$, the set of neighbors of v in G is denoted by $N_G(v)$. For $A \subseteq V(G)$, let $N_G(A)$ denote the set of all vertices in $G - A$ that have a neighbor in A. The *length* of a path is the number of edges on the path. For $v \in V(G)$ and a subgraph H of $G - v$, we say v is adjacent to H if it has a neighbor in H.

Two vertices v, w in a graph G are called *twins* if they have the same set of neighbors on $V(G) \setminus \{v, w\}$. For two vertex sets A and B, we say that

- A is *complete* to B if for every $a \in A$, $b \in B$, a is adjacent to b,
- A is *anti-complete* to B if for every $a \in A$, $b \in B$, a is not adjacent to b.

2.1 Distance-Hereditary Graphs

A graph G is called *distance-hereditary* if for every connected induced subgraph H of G and every $v, w \in V(H)$, the distance between v and w in H is the same as the distance between v and w in G. This graph class was first introduced by Howorka [17], and deeply studied by Bandelt and Mulder [3].

The house, the gem, the domino graphs are depicted in Figure 1. A graph isomorphic to one of the house, the gem, the domino, and induced cycles of length at least 5 will be called a *distance-hereditary obstruction* or shortly a *DH obstruction.* A DH obstruction with at most 6 vertices will be called a *small DH obstruction.* Note that every DH obstruction does not contain any twins.

It is known that distance-hereditary graphs are precisely the graphs not containing any DH obstruction as an induced subgraph [3]. The following lemma will be used to find DH obstructions later on.

▶ **Lemma 1** (Kantè, Kim, Kwon, and Paul [20]). *Let G be a graph obtained from an induced path of length at least 3 by adding a vertex v adjacent to its end vertices where v may be adjacent to some internal vertices of the path. Then G has a DH obstruction containing v.*

In particular, if the given path has length at most 4*, then* G *has a small DH obstruction containing* v*.*

2.2 Split decompositions

We follow the notations in [4]. A *split* of a connected graph G is a vertex partition (X, Y) of G such that $|X| \geq 2, |Y| \geq 2$, and $N_G(Y)$ is complete to $N_G(X)$. Splits are also called *1-joins*, or simply *joins* [13]. A connected graph G is called a *prime graph* if $|V(G)| \geq 5$ and it has no split.

A connected graph D with a distinguished set of edges $M(D)$ is called a *marked graph* if the edges in $M(D)$ form a matching and each edge in $M(D)$ is a cut edge. An edge in $M(D)$ is called a *marked edge*, and every other edge is called an *unmarked edge*. A vertex incident with a marked edge is called a *marked vertex*, and every other vertex is called an *unmarked vertex*. Each connected component of $D - M(D)$ is called a *bag* of D.

When G admits a split (X, Y), we construct a marked graph D on the vertex set $V(G) \cup \{x', y'\}$ such that

- for vertices x, y with $\{x, y\} \subseteq X$ or $\{x, y\} \subseteq Y$, $xy \in E(G)$ if and only if $xy \in E(D)$,
- $x'y'$ is a new marked edge,
- X is anti-complete to Y,
- $\{x'\}$ is complete to $N_G(Y) \cap X$ and $\{y'\}$ is complete to $N_G(X) \cap Y$ (with unmarked edges).

The marked graph D is called a *simple decomposition of* G. A *split decomposition* of a connected graph G is a marked graph D defined inductively to be either G or a marked graph defined from a split decomposition D' of G by replacing a connected component H of $D' - M(D')$ with a simple decomposition of H. See Figure 2 for an example of a split decomposition. We note that when D is a split decomposition of a graph G and u, v are two vertices in G, $uv \in E(G)$ if and only if there is a path from u to v in D where its first and last edges are unmarked, and an unmarked edge and a marked edge alternatively appear in the path [1, Lemma 2].

Naturally, we can define a reverse operation of decomposing into a simple decomposition; for a marked edge xy of a split decomposition D, *recomposing* xy is the operation of removing two vertices x and y and making $N_D(x) \setminus \{y\}$ complete to $N_D(y) \setminus \{x\}$ with unmarked edges. It is not hard to observe that if D is a split decomposition of G, then G can be obtained from D by recomposing all marked edges.

Note that there are many ways of decomposing a complete graph or a star, because every its non-trivial vertex partition is a split. Cunningham and Edmonds [8] developed a canonical way to decompose a graph into a split decomposition by not allowing to decompose a bag which is a star or a complete graph. A split decomposition D of G is called a *canonical split decomposition* if each bag of D is either a prime graph, a star, or a complete graph, and D cannot be obtained from a split decomposition with the same property by recomposing a marked edge. It is not hard to observe that every canonical split decomposition has no marked edge linking two complete bags, and no marked edge linking a leaf of a star bag and the center of another star bag [4]. Furthermore, for each pair of twins a, b in G, it holds that a, b must both be located in the same bag of the canonical split decomposition.

▶ **Theorem 2** (Cunningham and Edmonds [8])**.** *Every connected graph has a unique canonical split decomposition, up to isomorphism.*

▶ **Theorem 3** (Dahlhaus [10])**.** *The canonical split decomposition of a graph* G *can be computed in time* $\mathcal{O}(|V(G)| + |E(G)|)$*.*

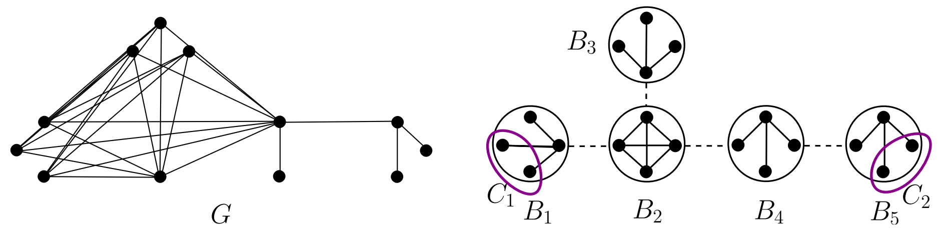


Figure 2 A graph G and its canonical split decomposition. Marked edges are represented by dashed edges, and bags are indicated by circles. Note that path$(B_1, B_5) = \{B_1, B_2, B_4, B_5\}$, bags B_4, B_5 are (C_1, C_2)-separator bags, and B_4 is a (B_1, B_5)-separator bag.

We can now give the second characterization of distance-hereditary graphs that is crucial for our results. For convenience, we call a bag a *star bag* or a *complete bag* if it is a star or a complete graph, respectively.

▶ **Theorem 4** (Bouchet [4]). *A graph is a distance-hereditary graph if and only if every bag in its canonical split decomposition is either a star bag or a complete bag.*

We will later on also need a little bit of additional notation related to split decompositions. Let D be a canonical split decomposition. For two distinct bags B_1 and B_2, we denote by comp(B_1, B_2) the connected component of $D - V(B_1)$ containing B_2. Technically, when $B_1 = B_2$, we define comp(B_1, B_2) to be the empty set. For two bags B_1 and B_2, we denote by path(B_1, B_2) the set of all bags containing a vertex in a shortest path from B_1 to B_2 in D. Note that path(B_1, B_2) contains B_1 and B_2. See Figure 2 for an example.

Let C_1, C_2 be two disjoint vertex subsets of D such that each C_1, C_2 is a set of unmarked vertices contained in (not necessarily distinct) bags B_1, B_2, respectively. A bag B is called *a* (C_1, C_2)*-separator bag* if B is a star bag contained in path(B_1, B_2) whose center is adjacent to neither comp(B, B_1) nor comp(B, B_2). We remark that B can be B_i for some $i \in \{1, 2\}$, and especially when $B_1 = B_2$ and it is a star bag and each C_i consists of leaves of B, B_1 is a (C_1, C_2)-separator bag. For convenience, we also say that a bag B is *a* (B_1, B_2)*-separator bag* if B is a star bag contained in path$(B_1, B_2) \setminus \{B_1, B_2\}$ whose center is adjacent to neither comp(B, B_1) nor comp(B, B_2). For this notation, B cannot be B_1 nor B_2. It is not hard to check that the length of the shortest path from C_1 to C_2 in the original graph is exactly the same as one plus the number of (C_1, C_2)-separator bags.

3 Setting the Stage

We begin by applying the *iterative compression technique* [25]. This technique allows us to transform our problem to a simpler problem called DISJOINT DISTANCE-HEREDITARY VERTEX DELETION. Our goal for the majority of the paper will be to obtain a single-exponential fixed-parameter algorithm for DISJOINT DISTANCE-HEREDITARY VERTEX DELETION; this is then used to obtain the sought after algorithm for DISTANCE-HEREDITARY VERTEX DELETION in Section 6.

DISJOINT DISTANCE-HEREDITARY VERTEX DELETION
Instance : A graph G, an integer k, and $S \subseteq V(G)$ with $|S| \leq k+1$ such that $G - S$ is distance-hereditary.
Parameter : k.
Task : Is there $Q \subseteq V(G) \setminus S$ with $|Q| \leq k$ such that $G - Q$ is distance-hereditary?

We will denote instances of Disjoint Distance-Hereditary Vertex Deletion as a tuple (G, S, k). By Theorem 4, every connected component of $G - S$ admits a canonical split decomposition whose bags are either a star or a complete graph.

Before explaining the general approach for solving Disjoint Distance-Hereditary Vertex Deletion, it will be useful to introduce a few definitions. Since the canonical split decomposition guaranteed by Theorem 4 only helps us classify twins in $G - S$ and not in G, we explicitly define an equivalence $\sim$ on the vertices of $G - S$ which allows us to classify twins in G: for two vertices $u, v \in V(G - S)$, $u \sim v$ iff they are twins in G.

We denote by $\mathbf{tc}(G - S)$ the set of equivalence classes of $\sim$ on $V(G - S)$, and each individual equivalence class will be called a *twin class* in $G - S$. We can observe that if $U \in \mathbf{tc}(G - S)$ lies in a single connected component of $G - S$, then U must be contained in precisely one bag of the split decomposition of this connected component of $G - S$, as U is a set of twins in $G - S$ as well. A twin class is *S-attached* if it has a neighbor in S, and *non-S-attached* if it has no neighbors in S. Similarly, we say that a bag in the canonical split decomposition of $G - S$ is *S-attached* if it has a neighbor in S, and *non-S-attached* otherwise.

3.1 Overview of the Approach

Now that we have introduced the required terminology, we can provide a high-level overview of our approach for solving Disjoint Distance-Hereditary Vertex Deletion.

1. We exhaustively apply the branching rules described in Section 3.2. Branching rules will be applied when G has a small subset $X \subseteq V(G - S)$ such that $S \cup X$ induces a DH obstruction, or there is a small connected subset $X \subseteq V(G - S)$ such that adding X to S decreases the number of connected components in $G[S]$.
2. We exhaustively apply the initial reduction rules described in Section 4. Each of these rules runs in polynomial time, finds a part in the canonical split decomposition of a connected component of $G - S$ that can be simplified, and modifies the decomposition. Each application of a reduction rule from Section 4 either reduces the number of vertices in $G - S$ or reduces the total number of bags in the canonical split decomposition (of a connected component of $G - S$). It is well known that the total number of bags in the canonical split decomposition of a graph is linear in the number of vertices. Therefore, the total number of applications of these initial reduction rules will also be at most linear in the number of vertices.
3. We say that G and the canonical split decompositions of $G - S$ are *reduced* if the branching rules in Section 3.2 and reduction rules in Section 4 cannot be applied anymore. We will obtain the following simple structure of the decompositions in the reduced instance:
   - Each canonical split decomposition D of a connected component of $G - S$ contains at least two distinct S-attached twin classes (Reduction Rule 1).
   - Each bag contains at most one S-attached twin class (Reduction Rule 3).
   - When B is a bag and D' is a connected component of $D - V(B)$ containing no bags having a neighbor in S, D' consists of one bag and B is a star bag whose center is adjacent to D' (Lemma 8).
   - When B is a bag and D' is a connected component of $D - V(B)$ such that D' contains exactly one S-attached bag B', there is no (B', B)-separator bag (Lemma 10).
4. We choose a canonical split decomposition D of a connected component of $G - S$ and assign any bag as a root bag of D. We choose a bag farthest from the root bag such that there are two descendant bags having S-attached twin classes C_1 and C_2, respectively. Then the length of every shortest path from C_1 to C_2 in $G - S$ is at most 2, and we

introduce a special polynomial-time reduction rule in Section 5 which simplifies this configuration.

Whenever we introduce a new rule, we need to show that it is *safe*; for branching rules this means that there exists at least one subinstance resulting from the rule which is a YES-instance iff the original graph was a YES-instance, while for reduction rules this means that the application of the rule preserves the property of being a YES-instance.

A vertex v in $G - S$ is called *irrelevant* if (G, S, k) is a YES-instance if and only if $(G - v, S, k)$ is a YES-instance. We will be identifying and removing irrelevant vertices in several of our reduction rules. When removing a vertex v from $G - S$, it is easy to modify the canonical split decomposition containing v, and thus it is not necessary to recompute the canonical split decomposition of the resulting graph from scratch [14].

3.2 Branching Rules

We state our two branching rules below.

▶ Branching Rule 1. For every vertex subset X of $G - S$ with $|X| \leq 5$, if $G[S \cup X]$ is not distance-hereditary, then we remove one of the vertices in X, and reduce k by 1.

▶ Branching Rule 2. For every vertex subset X of $G - S$ with $|X| \leq 5$ such that $G[X]$ is connected and the set $N_G(X) \cap S$ is not contained in a connected component of $G[S]$, then we either remove one of the vertices in X and reduce k by 1, or put all of them into S (which reduces the number of connected components of $G[S]$).

The safeness of Branching Rules 1 and 2 are clear, and these rules can be performed in polynomial time. The exhaustive application of these branching rules guarantees the following structure of the instance.

▶ **Lemma 5.** *Let (G, S, k) be an instance reduced under Branching Rules 1 and 2.*

1. *G has no small DH obstructions.*
2. *Let $v \in V(G - S)$. For every two vertices $x, y \in N_G(v) \cap S$, they are contained in the same connected component of $G[S]$ and there is no induced path of length at least 3 from x to y in $G[S]$. Specifically, if $xy \notin E(G)$, then there is an induced path xpy for some $p \in S$.*
3. *There is no induced path $v_1 \cdots v_5$ of length 4 in $G - S$ where v_1 and v_5 have neighbors in S but v_2 and v_4 have no neighbors in S.*
4. *There is no induced path $v_1 \cdots v_4$ of length 3 in $G - S$ where v_1 and v_4 have neighbors on S but v_2 has no neighbors on S.*

Lemma 5, and especially point (2) in the lemma, is used in many parts of our proofs. Since we will apply the branching rules exhaustively at the beginning and also after each new application of a reduction rule, these properties will be implicitly assumed to hold in subsequent sections.

4 Reduction Rules in Split Decompositions

In this section, we assume that the given instance (G, S, k) is reduced under Branching Rules 1 and 2. The reduction rules introduced here either remove some irrelevant vertex, or move some vertex into S, or reduce the number of bags in the decomposition by modifying the instance into an equivalent instance. After we apply any of these reduction rules, we will run the two branching rules from Section 3 again.

Before we move on to the reduction rules themselves, we introduce a generic way of finding an irrelevant vertex which will be used in many reduction rules. For a vertex v in $G - S$ and an induced cycle H of length at least 5 in G containing a vertex v and two neighbors w, z of v in H, a vertex v' in S is called a *bad vertex* for H and v if v' is adjacent to w and z. If such a vertex v' exists, it is clear that v' is not contained in H because $vwv'zv$ is a cycle of length 4. More importantly, since $H - v$ is an induced path of length at least 3 from w to z and v' is adjacent to both of its endpoints, by Lemma 1, $G[(V(H) \setminus \{v\}) \cup \{v'\}]$ contains a DH obstruction. This implies that one of the vertices in $V(H) \setminus \{v\}$ must be contained in every solution (note that $v' \in S$ and so v' itself cannot be part of a solution). This property results in the following two lemmas.

▶ **Lemma 6.** *Let (G, S, k) be an instance reduced under Branching Rule 1. Let v be a vertex in $G - S$ such that for every induced cycle H of length at least 7 containing v, there is a bad vertex for H and v. Then v is irrelevant.*

▶ **Lemma 7.** *Let (G, S, k) be an instance reduced under Branching Rules 1 and 2. Let v be a vertex in $G - S$ and H be an induced cycle of length at least 7 containing v, and let w, z be the two neighbors of v in H. If $w, z \in S$, then there is a bad vertex for H and v, and thus $G[(V(H) \setminus \{v\}) \cup S]$ contains a DH obstruction.*

We are now ready to start with our reduction rules. For the remainder of this section, let us fix a canonical split decomposition D of a connected component of $G - S$.

▶ Reduction Rule 1. If D has at most one S-attached twin class, then we remove all unmarked vertices of D from G.

▶ Reduction Rule 2. Let B be a star bag whose center is unmarked, and let v be a leaf unmarked vertex in B. If v has no neighbor in S, then we remove v. If v has a neighbor in S, then we move v into S.

We remark that when we move v into S in Reduction Rule 2, $k + cc(G[S])$ does not increase. Next, we introduce an important rule which reduces the number of S-attached twin classes in each bag.

▶ Reduction Rule 3. Let B be either a complete bag or a star bag whose center is marked. Let C_1, C_2 be two distinct S-attached twin classes in B such that $(N_G(C_1) \setminus N_G(C_2)) \cap S$ is non-empty. Then we remove C_1.

We proceed by introducing a reduction rule which sequentially arranges non-S-attached bags in a canonical split decomposition. The number of bags in D is strictly reduced when applying Reduction Rule 4.

▶ Reduction Rule 4. Let B be a leaf bag and B' be the neighbor bag of B.
1. If B is a complete bag having exactly one twin class and B' is a star bag whose leaf is adjacent to B, then we transform B into a star whose center is adjacent to B', and recompose the marked edge connecting B and B'.
2. If B is a star bag having exactly one twin class, the center of B is adjacent to B', and B' is a complete bag, then we transform B into a complete graph, and recompose the marked edge connecting B and B'.

The next reduction rule allows us to remove a non-S-attached twin class under certain conditions (see Figure 3).

▶ Reduction Rule 5. Let B_1 be a leaf bag containing at most one S-attached twin class and B_2 be a bag distinct from B_1 such that

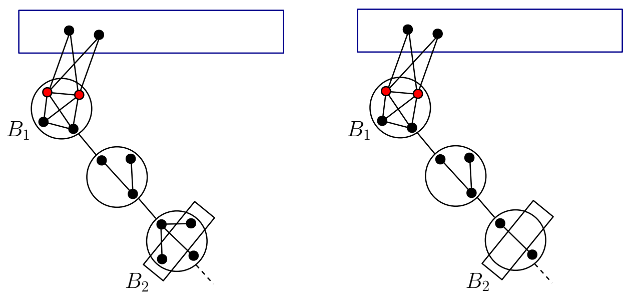


Figure 3 Reduction Rule 5.

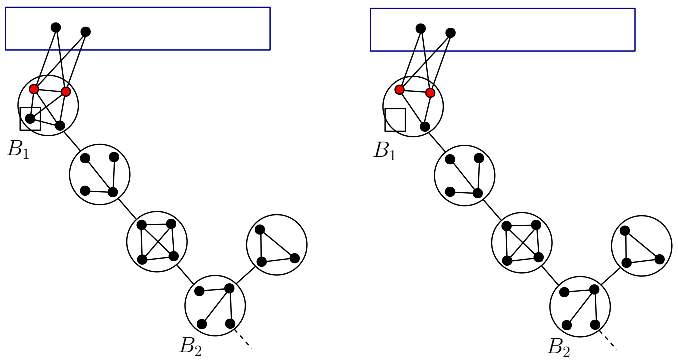


Figure 4 Reduction Rule 6.

- every bag in path$(B_1, B_2) \setminus \{B_1, B_2\}$ is non-S-attached, not a (B_1, B_2)-separator bag, and has exactly two neighbor bags, and
- B_2 is a star bag whose center is adjacent to comp(B_2, B_1).

If B_2 contains a non-S-attached twin class C, then we remove C.

We can now show that after the exhaustive application of the reduction rules introduced up to this point, every connected component of $D - V(B)$ containing no S-attached bags is "simple", as formalized in the next lemma.

▶ **Lemma 8.** *Let D be the canonical split decomposition of a connected component of $G - S$ reduced under Reduction Rules 1–5. Let B be a bag and D' be a connected component of $D - V(B)$ containing no S-attached bags. Then D' consists of one bag and B is a star bag whose center is adjacent to D'.*

Next, we introduce some rules simplifying connected components of $D - V(B)$ for some bag B containing one S-attached twin class. The following rule is depicted in Figure 4.

▶ Reduction Rule 6. Let B_1 be a leaf bag having exactly one S-attached twin class and B_2 be a bag distinct from B_1 such that

- B_1 is not a star whose leaf is adjacent to a neighboring bag,
- every bag in path$(B_1, B_2) \setminus \{B_1, B_2\}$ is non-S-attached, not a (B_1, B_2)-separator bag and has exactly two neighbor bags, and
- B_2 is a star whose center is either an unmarked vertex, or adjacent to a connected component of $D - V(B_2)$ consisting of one non-S-attached bag.

If B_1 contains a non-S-attached twin class C, then we can safely remove C.

By applying Reduction Rules 4, 5, and 6, we can simplify the decomposition near an S-attached leaf containing one S-attached twin class; for instance, in Figure 4, B_1 will be

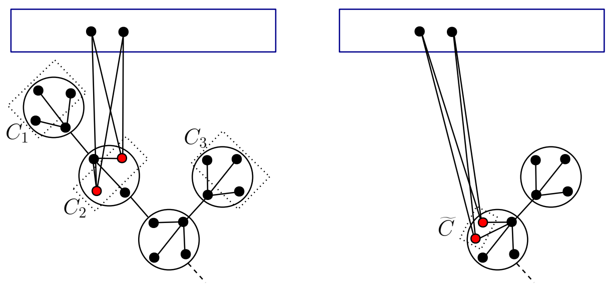


Figure 5 Reduction Rule 8.

eventually merged with B_2. We state the properties that are guaranteed by the reduction rules introduced up to this point in the following lemma.

▶ **Lemma 9.** *Let D be the canonical split decomposition of a connected component of $G - S$ reduced under Reduction Rules 1–6. Let B be a star bag whose center is unmarked or adjacent to a connected component of $D - V(B)$ consisting of one non-S-attached bag. Let D' be a connected component of $D - V(B)$ such that*

- *D' contains exactly one S-attached bag B', and*
- *there is no (B', B)-separator bag.*

Then B' is a star whose leaf is adjacent to $\text{comp}(B', B)$ *and there is a leaf bag B'' where the center of B' is adjacent to B''.*

The final two rules in this section help us simplify the configuration specified in Lemma 9; using Reduction Rule 7 we can remove all unmarked vertices in $\text{path}(B, B') \setminus \{B, B'\}$, and then Reduction Rule 8 allows us to merge B' with B.

▶ Reduction Rule 7. Let B_1 and B_2 be two star bags in D such that

- for each i, either the center of B_i is an unmarked vertex, or the center of B_i is adjacent to a connected component of $D - V(B_i)$ consisting of one non-S-attached bag,
- every bag in $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$ is a non-S-attached bag, has two neighbor bags, and is not a (B_1, B_2)-separator bag.

Then we remove every unmarked vertex in every bag in $\text{path}(B_1, B_2) \setminus \{B_1, B_2\}$.

▶ Reduction Rule 8. Let B_1, B_2, B_3 be distinct bags in D such that

- B_1 is a non-S-attached leaf bag whose neighbor bag is B_2, and it is not a star whose leaf is adjacent to B_2,
- B_2 has exactly two neighbor bags B_1 and B_3, it is a star whose center is adjacent to B_1, and the set of unmarked vertices in B_2 is the unique S-attached twin class C_2 in B_2, and
- B_3 is a star whose center is either an unmarked vertex, or adjacent to a connected component of $D - V(B_3)$ consisting of one non-S-attached bag.

Let C_1 be the set of unmarked vertices in B_1. Then we remove B_1 and B_2, and add a leaf set of unmarked vertices $\widetilde{C}$ with $\min(|C_1|, |C_2|)$ vertices to B_3, that is complete to $N_G(C_2) \cap S$ and has no other neighbors in S.

We provide an illustration of Reduction Rule 8 in Figure 5.

Finally, after applying all the reduction rules in this section, our instance has the desired inseparability property. We formalize and prove this property below.

▶ **Lemma 10.** *Let D be the canonical split decomposition of a connected component of $G - S$ reduced under Reduction Rules 1–8. Let B be a bag and let D' be a connected component of $D - V(B)$ such that D' contains exactly one S-attached bag B'. Then there is no (B', B)-separator bag.*

▶ **Proposition 11.** Let (G, S, k) be an instance reduced under Branching Rules 1 and 2. Given a canonical split decomposition D of a connected component of $G - S$, we can in time $\mathcal{O}(|V(G)|^2)$ either apply one of Reduction Rules 1–8, or correctly answer that Reduction Rules 1–8 cannot be applied anymore.

5 Twin Class Reduction Rule

In this section, we introduce our last, but perhaps most important, reduction rule. Later on in the proof of Theorem 13, we will show that whenever the other rules cannot be applied, we can either apply Reduction Rule 9 or our instance is trivial.

▶ Reduction Rule 9. Suppose that (G, S, k) and all canonical split decompositions of connected components of $G-S$ are reduced under Branching Rules 1–2 and Reduction Rules 1–8. Let D be the canonical split decomposition of a connected component of $G - S$, and let B be a bag, and B_1, B_2 be two distinct S-attached bags (possibly $B_i = B$ for some $i \in \{1, 2\}$). Furthermore, let C_1, C_2 be two distinct S-attached twin classes in B_1, B_2, respectively, such that for each $i \in \{1, 2\}$, either $B_i = B$ or C_i is the unique S-attached twin class in $\text{comp}(B, B_i)$. Then we apply one of the following:

1. If the distance from C_1 to C_2 in $G - S$ is 2 and the unique (C_1, C_2)-separator bag is contained in $\text{comp}(B, B_2)$, then we remove C_2. (We show that B cannot be the (C_1, C_2)-separator bag.)
2. If C_1 is complete to C_2, $B \neq B_2$, and B is a star bag whose center is adjacent to $\text{comp}(B, B_2)$, then we remove C_1.
3. If C_1 is complete to C_2, $B \neq B_1$, and B is a complete bag, then B_1 contains a non-S-attached twin class C_1' and we remove C_1'.

▶ **Proposition 12.** Reduction Rule 9 is safe.

Sketch of Proof. Here we prove the proposition for one important special case. Suppose that C_1 is anti-complete to C_2 and the (C_1, C_2)-separator bag is contained in $\text{comp}(B, B_2)$. We claim that every vertex in C_2 is irrelevant. For each $i \in \{1, 2\}$, let $c_i \in C_i$ and let $T_i = N_G(C_i)$. Let B' be the (C_1, C_2)-separator bag. We first confirm that $B_2 = B'$. If not, then B' is a (B_2, B)-separator bag. However, since $\text{comp}(B, B_2)$ has exactly one S-attached bag B_1, by Lemma 10, there is no (B_2, B)-separator bag, a contradiction. We conclude that $B' = B_2$. There is a leaf bag B_2' where the center of B_2 is adjacent to B_2', otherwise, we can apply Reduction Rule 2.

Let $v \in C_2$. We claim that for every induced cycle H of length at least 7 containing v, there is a bad vertex for H and v. If this is true, then the result follows from Lemma 6. Let w and z be the two neighbors of v in H. If w and z are contained in S, then by Lemma 7, there is a bad vertex. On the other hand, w and z cannot be contained in $V(G-S)$ together, because the vertices in B_2' form a twin class. We may assume that $w \in (T_1 \cap T_2) \cap V(G - S)$ and $z \in S$. We actually show that this is not possible. Note that since $w \in V(B_2')$, w has no neighbors in S.

We divide cases depending on the location of z: specifically, to conclude the proof, we separately consider the case of $z \in (T_2 \setminus T_1) \cap S$ and $z \in (T_1 \cap T_2) \cap S$. We show that the former case always leads to a contradiction with w having no neighbors in S. On the other hand, it can be shown that the latter case necessarily implies the existence of a small DH obstruction, contradicting the exhaustive application of Branching Rules 1–2. ◀

6 The Algorithm and Lower Bounds

Our goal in this section is to give a proof of our main result, Theorem 14, and prove corresponding lower bounds.

▶ **Theorem 13.** Disjoint Distance-Hereditary Vertex Deletion *can be solved in time* $\mathcal{O}(36^k \cdot |V(G)|^6(|V(G)| + E(G)))$.

Sketch of Proof. The main argument in the proof is that whenever we cannot apply one of Branching Rules 1–2 and Reduction Rules 1–8, either we have a trivial instance, or we run into a situation where we can apply Reduction Rule 9. Suppose that D is the canonical split decomposition of a connected component of $G - S$ such that G and D are reduced under those rules. If D contains at most one S-attached twin class, then we can apply Reduction Rule 1. Thus, we know that D contains at least two distinct S-attached twin classes.

We choose a root bag of D, and choose a bag B that is farthest from the root bag such that there are two descendant bags B_1, B_2 of B having distinct S-attached twin classes C_1, C_2, respectively. By Reduction Rule 3, we have $B_1 \neq B_2$. Using the structure that if $B_i \neq B$, then there is no (B_i, B)-separator bag by Lemma 10, we can observe that the distance between C_1 and C_2 in $G - S$ is at most 2, and then C_1 and C_2 satisfy one of the conditions in Reduction Rule 9.

We can notice that each branching rule reduces either k or the number of connected components in S and branch into at most 6 subinstances. Since none of the reduction rules change k or the number of components in S, branching rules are applied at most $2k$ times. Due to the application of reduction rules (which we also consider as nodes of the branching tree and which may be applied independently in different branches), the branching tree will have at most $\mathcal{O}(36^k \cdot |V(G)|)$ nodes, and the runtime in every node will not exceed $\mathcal{O}(|V(G)|^5(|V(G)| + |E(G)|))$. Hence, the whole algorithm for Disjoint Distance-Hereditary Vertex Deletion can be implemented in time $\mathcal{O}(36^k \cdot |V(G)|^6(|V(G)| + |E(G)|))$. ◀

▶ **Theorem 14.** Distance-Hereditary Vertex Deletion *can be solved in time* $\mathcal{O}(37^k \cdot |V(G)|^7(|V(G)| + |E(G)|))$.

Sketch of Proof. Let $n := |V(G)|$ and $m := |E(G)|$. Fix an arbitrary labeling $v_1, \dots, v_n$ of $V(G)$ and let $G_i := G[\{v_1, \dots, v_i\}]$ for $1 \leq i \leq n$. From $i = 1$ up to n, given a graph G_i and $S_i \subseteq V(G_i)$ with $|S_i| \leq k + 1$ such that $G_i - S_i$ is distance-hereditary, we aim to find a set $S_i' \subseteq V(G_i)$ with $|S_i'| \leq k$ such that $G_i - S_i'$ is distance-hereditary if one exists. We further guess all possible $S_i' \cap S_i$ as I, and we aim to find a deletion set S_i'' of size at most $k - |I|$ in $G_i - I$ where $S_i'' \cap (S_i \setminus I) = \emptyset$ if one exists. We can recursively resolve this problem using Disjoint Distance-Hereditary Vertex Deletion. As we iterate the subproblem n times, we obtain the runtime $n \cdot \sum_{i=0}^{k} \binom{k+1}{i} \cdot \mathcal{O}(36^{k-i} \cdot n^6(n+m)) = \mathcal{O}(37^k \cdot n^7(n+m))$. ◀

Our lower bound result is based on the well-established exponential time hypothesis [19], and specifically uses the fact that there is no $2^{o(k)} \cdot |V(G)|^{\mathcal{O}(1)}$ algorithm for Vertex Cover, unless ETH fails [5]. The proof relies on a reduction which is similar to the one used for vertex deletion to graphs of linear rank-width 1 [20].

▶ **Theorem 15.** *There is no* $2^{o(k)} \cdot |V(G)|^{\mathcal{O}(1)}$ *algorithm for* Distance-Hereditary Vertex Deletion *unless ETH fails.*

7 Concluding Notes

We conclude with a few remarks on why we believe that the presented algorithm is of high interest. First, it intrinsically exploits the properties guaranteed by distinct, seemingly unrelated characterization of distance-hereditary graphs; this approach can likely be used to design or improve algorithms for other vertex deletion problems. Second, it uses highly nontrivial reduction rules which simplify canonical split decompositions, and an adaptation or extension of the presented rules could be highly relevant for other graph classes characterized by special canonical split decompositions, such as parity graphs [6] or circle graphs [13]. Third, it is the first of its kind which targets a "full" class of graphs of bounded rank-width (contrasting previous results for specific subclasses of graphs of rank-width 1 [18, 2, 21, 20]).

It is worth noting that there remains a number of interesting open problems in this general area. Perhaps the most prominent one is the question of whether vertex deletion to graphs of rank-width c, for any constant c, admits a single-exponential fixed-parameter algorithm. Our algorithm represents the first steps in this general direction. The existence of a polynomial kernel or an approximation algorithm for such vertex deletion problems also remains open, even for the case of distance-hereditary graphs.

References

1 Isolde Adler, Mamadou Moustapha Kanté, and O-joung Kwon. Linear rank-width of distance-hereditary graphs. In *Graph-Theoretic Concepts in Computer Science - 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, pages 42–55, 2014. doi:10.1007/978-3-319-12340-0_4.

2 Akanksha Agrawal, Sudeshna Kolay, Daniel Lokshtanov, and Saket Saurabh. A faster FPT algorithm and a smaller kernel for block graph vertex deletion. In *LATIN 2016: Theoretical Informatics - 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 1–13, 2016.

3 Hans-Jürgen Bandelt and Henry M. Mulder. Distance-hereditary graphs. *J. Combin. Theory Ser. B*, 41(2):182–208, 1986. doi:10.1016/0095-8956(86)90043-2.

4 André Bouchet. Transforming trees by successive local complementations. *J. Graph Theory*, 12(2):195–207, 1988.

5 Liming Cai and David Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67(4):789 – 807, 2003. doi:10.1016/S0022-0000(03)00074-6.

6 Serafino Cicerone and Gabriele Di Stefano. On the extension of bipartite to parity graphs. *Discrete Applied Mathematics*, 95(1–3):181–195, 1999. doi:10.1016/S0166-218X(99)00074-8.

7 William H. Cunningham. Decomposition of directed graphs. *SIAM J. Algebraic Discrete Methods*, 3(2):214–228, 1982.

8 William H. Cunningham and Jack Edmonds. A combinatorial decomposition theory. *Canad. J. Math.*, 32(3):734–765, 1980. doi:10.4153/CJM-1980-057-7.

9 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

10 Elias Dahlhaus. Parallel algorithms for hierarchical clustering, and applications to split decomposition and parity graph recognition. *Journal of Algorithms*, 36(2):205–240, 2000. doi:10.1006/jagm.2000.1090.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

12 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Planar f-deletion: Approximation, kernelization and optimal FPT algorithms. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 470–479, 2012.

13 Csaba P. Gabor, Kenneth J. Supowit, and Wen Lian Hsu. Recognizing circle graphs in polynomial time. *J. Assoc. Comput. Mach.*, 36(3):435–473, 1989.

14 Emeric Gioan and Christophe Paul. Split decomposition and graph-labelled trees: characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Appl. Math.*, 160(6):708–733, 2012. doi:10.1016/j.dam.2011.05.007.

15 Peter L. Hammer and Frédéric Maffray. Completely separable graphs. *Discrete Applied Mathematics*, 27(1-2):85–99, 1990. doi:10.1016/0166-218X(90)90131-U.

16 Pinar Heggernes, Pim van 't Hof, Bart M.P. Jansen, Stefan Kratsch, and Yngve Villanger. Parameterized complexity of vertex deletion into perfect graph classes. *Theoretical Computer Science*, 511:172 – 180, 2013. doi:10.1016/j.tcs.2012.03.013.

17 E. Howorka. A characterization of distance-hereditary graphs. In *The Quarterly Journal of Mathematics, Oxford, Second Series*, 28 (112):417–420, 1977.

18 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory Comput. Syst.*, 47(1):196–217, 2010. doi:10.1007/s00224-008-9150-x.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512 – 530, 2001. doi:10.1006/jcss.2001.1774.

20 Mamadou Moustapha Kanté, Eun Jung Kim, O-joung Kwon, and Christophe Paul. An FPT Algorithm and a Polynomial Kernel for Linear Rankwidth-1 Vertex Deletion. In Thore Husfeldt and Iyad Kanj, editors, *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 138–150, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2015.138.

21 Eun Jung Kim and O-joung Kwon. A Polynomial Kernel for Block Graph Deletion. In Thore Husfeldt and Iyad Kanj, editors, *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 270–281, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2015.270.

22 Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 613–624, 2013. doi:10.1007/978-3-642-39206-1_52.

23 Sang-il Oum. Rank-width and vertex-minors. *J. Comb. Theory, Ser. B*, 95(1):79–100, 2005. URL: http://dx.doi.org/10.1016/j.jctb.2005.03.003, doi:10.1016/j.jctb.2005.03.003.

24 Sang-il Oum. Rank-width and well-quasi-ordering. *SIAM J. Discrete Math.*, 22(2):666–682, 2008. doi:10.1137/050629616.

25 Bruce Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299 – 301, 2004. doi:10.1016/j.orl.2003.10.009.

Preprocessing Under Uncertainty: Matroid Intersection

Stefan Fafianie[1], Eva-Maria C. Hols[2], Stefan Kratsch[3], and Vuong Anh Quyen[4]

1 **Department of Computer Science, University of Bonn, Germany**
`fafianie@cs.uni-bonn.de`

2 **Department of Computer Science, University of Bonn, Germany**
`hols@cs.uni-bonn.de`

3 **Department of Computer Science, University of Bonn, Germany**
`kratsch@cs.uni-bonn.de`

4 **Department of Computer Science, University of Bonn, Germany**
`vuong@cs.uni-bonn.de`

Abstract

We continue the study of preprocessing under uncertainty that was initiated independently by Assadi et al. (FSTTCS 2015) and Fafianie et al. (STACS 2016). Here, we are given an instance of a tractable problem with a large static/known part and a small part that is dynamic/uncertain, and ask if there is an efficient algorithm that computes an instance of size polynomial in the uncertain part of the input, from which we can extract an optimal solution to the original instance for all (usually exponentially many) instantiations of the uncertain part.

In the present work, we focus on the MATROID INTERSECTION problem. Amongst others we present a positive preprocessing result for the important case of finding a largest common independent set in two linear matroids. Motivated by an application for intersecting two gammoids we also revisit MAXIMUM FLOW. There we tighten a lower bound of Assadi et al. and give an alternative positive result for the case of low uncertain capacity that yields a MAXIMUM FLOW instance as output rather than a matrix.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases preprocessing, uncertainty, maximum flow, matroid intersection

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.35

1 Introduction

Recently, Assadi et al. [2] and, independently, Fafianie et al. [12] initiated a study of problems where part of the input is *dynamic* or *uncertain*. While the introduced concepts are differently named, i.e., *dynamic sketching* [2] and *preprocessing under uncertainty* [12], and are rooted in different areas, i.e., *streaming algorithms* and *parameterized complexity* respectively, the fundamental goal is the same: Given an instance x that is largely static/known and a small specified part, say of k bits, that is dynamic/uncertain. Can we extract from x in polynomial time an instance (or just any string) x' of size polynomial in k such that optimal solutions for x for any instantiation of the k bits of dynamic/uncertain part can also be computed just from x' and the k bits? Since there are 2^k instantiations of k bits this is clearly nontrivial, as we can afford neither the time nor the space to simply precompute and store 2^k solutions, even for polynomial-time solvable problems (which are the focus of this work). Arguably, this parallel development of essentially the same goal in different contexts speaks to the generality and importance of the question. Let us anyway recall some of the motivation.[1]

We are often faced with inputs that appear over time or that are subject to changes. The areas of *online algorithms* and *streaming algorithms* deal with the extreme case that the entire input is only revealed right before or during the computation, or that the data is continuously changing. What if we already hold "most" of the input and there is only a small amount of data that is uncertain or subject to changes? Can we do better than under the strict settings of online or streaming algorithms? Canonical examples are machine scheduling in a factory with only few irregularly occurring jobs, or routing in a network when there is a small set of links that are frequently congested. Unlike online algorithms, the goal is to obtain optimal solutions rather than approximate ones. To make this possible, we do not insist on committing to (parts of) a solution without knowing the uncertain part and only require to preprocess and shrink the given part to size polynomial in the amount of uncertain information. In other words, we do not desire a solution that is (approximately) robust to uncertainty, but to do as much work as possible before receiving the uncertain part (and then compute a solution). In the introduction of [12] this is sketched for the simple case of computing a shortest s,t-path in a road network when the transit times of some k roads is not known in advance; let us discuss a different straightforward example for illustration.

Consider the Closest Pair problem where we are given a set P of points in the Euclidean plane such that for k points $P' \subseteq P$ it is not known beforehand whether they appear in the final input. For any instantiation of availability of points in P' we know that the closest pair of points is either contained entirely in $P \setminus P'$ or P', or one point is in $P \setminus P'$ while the other is in P'. Accordingly, it suffices to store the closest pair in $P \setminus P'$, the set P', and for each point $p' \in P'$ its closest point in P. Thus, we obtain an equivalent instance with at most $2|P'| + 2$ points such that, after removing any subset of P', the closest pair of points has the same distance as it would have in the input instance.

The example shows that such a preprocessing can be quite simple and fast, and be feasible even for larger amounts of uncertain data. On the other end of the spectrum, there are problem settings where even an arbitrary amount of preprocessing time does not suffice to obtain a polynomially large sketch of the instance. Conveniently, the lower bound proofs do not require any complexity assumptions, but rely on fundamental information-theoretic arguments that are implicit in the well-known lower bound for the membership game: Therein, Alice holds any subset $S \subseteq [n]$ unknown to Bob, whereas Bob holds an integer $i \in [n]$ unknown to Alice, and communication is only allowed from Alice to Bob. How many bits of information does Alice need to send to Bob in order for Bob to be able to answer whether $i \in S$? The answer is that n bits are necessary and (obviously) sufficient. Note that the sent information necessarily works for all $i \in [n]$ that Bob could hold, and hence must represent the entire set S. In other words, the lower bound also gives us an incompressibility argument for n bits of information. Fafianie et al. [12] give lower bounds for small connected vertex cover and linear programming under various forms of uncertainty by showing that the existence of an efficient preprocessing algorithm would lead to a violation of the aforementioned bounds.

Assadi et al. [2] explore graph problems where uncertainty is restricted to a set T of k *terminal* vertices, e.g., their adjacency is uncertain. Their main positive result is on the maximum matching problem. Using the *Tutte matrix* of the input graph with n vertices, they show that storing only a $2k \times 2k$ matrix whose entries are in $\mathbb{Z}_p$ (p is any prime of

[1] In the following we will stick to the terms as used in [12].

magnitude $\Theta(\frac{n}{\delta})$ with $0 < \delta < 1$) and an integer suffices to extract the rank of the Tutte matrix for each instantiation/change to the adjacency of the terminals; this yields the size of a maximum matching of G. This result then gives rise to a *cut-preserving sketch* result where the value of all $(S, T \setminus S)$-cuts in G is preserved for $S \subseteq T$ in a sketch of size $\mathcal{O}(kC^2)$ where C is the total capacity of edges incident on T. They obtain this result by constructing a bipartite graph and creating a dynamic sketch for the maximum matching problem. In addition, they prove a lower bound of $\Omega(C/\log C)$ bits which implies a lower bound of $2^{\Omega(k)}$ bits. Furthermore, they show how to obtain a sketch of size $\mathcal{O}(k^4)$ for s,t-CONNECTIVITY, by again using the dynamic sketch for the maximum matching problem. These results extend to MAXIMUM FLOW as well: it follows that the maximum flow problem has a sketch of size $\mathcal{O}((k+C')^4)$ (in the form of a $8(k+C')^2 \times 8(k+C')^2$) matrix); here C' is the total capacity of edges between terminals, which is arguably an advantage over the dependency on C. They also point out that the maximum flow problem has a $2^{\Omega(k)}$ lower bound on size of dynamic sketches which follows from the lower bound for the cut-preserving sketch. Finally, they give an $\mathcal{O}(k)$ size dynamic sketch for the MINIMUM SPANNING TREE problem.

The result for MINIMUM SPANNING TREE was obtained independently by Fafianie et al. [12], where it is generalized to the problem of finding a minimum weight basis of a matroid when the presence of k ground set elements is uncertain. (The MST problem is the same as finding a minimum weight basis of a *graphic matroid*.) If the matroid is given by a matrix respectively by oracle access then the output is a smaller matroid given by a (smaller) matrix respectively by restricted access to the original oracle (e.g., smaller ground set). These results work also in the more general setting where the *weights* of k edges/elements are not known beforehand. Furthermore, for the BIPARTITE MATCHING problem with k uncertain vertices or edges, Fafianie et al. [12] show how to efficiently reduce to a new graph G' whose maximum matchings, relative to availability of the uncertain vertices, differ from those of the input graph G by a fixed (and known) amount. In other words, the output in all three cases is an equivalent instance of the same problem.

The known results leave different directions for further study. The main direction would be to study other polynomial-time solvable regarding preprocessing under uncertainty respectively dynamic sketching. There were several positive results, but the lower bounds for MAXIMUM FLOW and LINEAR PROGRAMMING show limitations for more general problems. Among other fundamental polynomial-time problems there are certainly MATROID INTERSECTION, LINEAR MATROID PARITY, STABLE MARRIAGE, and problem families such as string matching or scheduling. We note that the same problem may have several interesting variants for making parts of its input uncertain. Due the importance of MAXIMUM FLOW and LINEAR PROGRAMMING also restrictions of the problems, e.g., restricted input graphs or special types of LPs, seem reasonable in order to obtain positive results for them. Another question would be how important it is to have the output be an instance of the same problem. This is arguably beneficial for applications, especially if existing algorithms for the underlying problem can be applied as-is. Likely, lower bounds will be unaffected by this decision since we only know how to get lower bounds for the bit size of the encoding; this is similar to the situation of lower bounds for kernelization in parameterized complexity (cf. [6, 10]).

Our work. We focus mainly on preprocessing under uncertainty for the MATROID INTERSECTION problem (Section 4) and present three positive results, including the important case of intersecting two linear matroids. We also revisit MAXIMUM FLOW (Section 3) where we tighten a lower bound of Assadi et al. [2] and give a positive result that is used as a subroutine for GAMMOID INTERSECTION. We conclude with a brief discussion and point out some open problems (Section 5). Let us discuss the results in some more detail.

Matroid Intersection. In the Matroid Intersection problem we are given as input two matroids over the same ground set E, and the goal is to find a set of maximum cardinality which is independent in both matroids. This is a classical optimization problem studied since the early 1970s, e.g., in [1, 11, 18], and generalizes a wide range of concrete problems such as the bipartite matching problem and the colorful spanning tree problem. There are also applications of matroid intersection outside of combinatorial optimization [9, 20]. For many algorithms the independent sets of a matroid are given by an independence oracle, i.e., a blackbox algorithm which answers whether a given subset of the ground set is independent or not. Another common way is to represent a matroid by a matrix over some field: Matroids which can be represented in this way are called *linear matroids.* One can obtain more efficient algorithms for linear matroid intersection that work directly on a matrix rather than via an oracle; e.g., Gabow and Xu [13] make use of fast matrix multiplication. It is also possible to provide matroids implicitly, e.g., by the underlying graphs (*gammoids, graphic matroids*). Finding a maximum common independent set of two matroids is solvable in polynomial time [11] but finding a maximum common independent set of three matroids is NP-hard; e.g. the DIRECTED HAMILTON PATH problem can be formulated as the intersection of three matroids [26]. We study the LINEAR MATROID INTERSECTION problem in the setting that the presence of k elements in the ground set is uncertain. Solving all possible 2^k instantiations in polynomial time is impossible; but we will show how to construct a small encoding from which we can compute the size of a maximum common independent set for all instantiations.

To get the result for LINEAR MATROID INTERSECTION we use a result of Harvey [15], which determines the size of a maximum common independent set by computing the rank of a matrix Z that contains the matrix representations of the two linear matroids as sub-matrices. We use this matrix Z to compute a $2k \times 2k$ matrix and an integer from which we can compute the size of a maximum common independent set for all 2^k instantiations. The construction of the $2k \times 2k$ matrix uses similar ideas as the construction of the $2k \times 2k$ matrix for the dynamic sketching scheme for the maximum matching problem of Assadi et al. [2] and the compression for the K-set-cycle problem of Wahlström [25]. However, we have to be much more careful during row and column operations because our initial matrix has entire rows and columns whose presence is uncertain in the final instance; the uncertain part in the paper of Assadi et al. [2] is contained in a $k \times k$ sub-matrix of the initial matrix.

Since the output of our preprocessing is not an instance of LINEAR MATROID INTERSECTION, this poses the question of whether special cases of the problem permit a preprocessing whose output is an equivalent instance of the same problem. We prove this for the fairly general case of the intersection of two *gammoids*, which contains several classes of well-studied matroids (e.g., *transversal matroids*) and for the ROOTED ARBORESCENCE problem, where we want to determine the existence of a rooted arborescence in a directed graph with some uncertain arcs; note that the ROOTED ARBORESCENCE problem can be described as the intersection of a *partition matroid* and a *graphic matroid.* For the gammoid intersection problem, we show how to compute two new gammoids over the same ground set of size $\mathcal{O}(|T|^3)$ and an offset value from which we can compute the size of a maximum common independent set for all 2^k instantiations. For the ROOTED ARBORESCENCE problem we compute a graph with $k+1$ vertices from which we can decide for all 2^k instantiations whether the input instance has a rooted arborescence. We complement this by a lower bound of $\binom{k}{\lceil k/2 \rceil}$ bits for the case of k uncertain vertices.

Maximum Flow. The problem of finding a maximum flow is one of the most important problems besides the more general problem of solving linear programs and has been explicitly studied in graph theory and combinatorial optimization. We show that if there is an arc set

F of unit-capacity arcs whose presence is uncertain (equivalently: there is a total of $|F|$ units of uncertain capacity), while capacity of other arcs is arbitrary, then we can efficiently reduce to an equivalent unit-capacity flow network with $\mathcal{O}(|F|^3)$ vertices. If we are only interested in any encoding of small size, then instead of the obvious $\mathcal{O}(|F|^6)$ bits encoding size this can also be represented as a gammoid, using $\mathcal{O}(|F|^3)$ bits, which follows from results from Kratsch and Wahlström [17]. This improves the upper bound of Assadi et al. [2] who give a dynamic sketching scheme with sketch size $\mathcal{O}((k+C')^4)$ for the maximum flow problem (represented as a $\mathcal{O}((k+C')^2) \times \mathcal{O}((k+C')^2)$ matrix); since the size of a sketch is measured by the number of machine words of length $\mathcal{O}(\log(n))$, their sketch needs $\mathcal{O}((k+C')^4 \log(n))$ bits. Recall that C' is the total capacity of all edges between the k terminals; by small modifications to the graph $|F|$ and C' become comparable. We complement this by a lower bound of 2^k bits for the case of k uncertain arcs with large capacity which slightly improves the lower bound of $2^{\Omega(k)}$ of Assadi et al. [2]. Furthermore we show that our lower bound is tight, even when the encoding only preserves the parity of the maximum flows.

Further related work. Generally, apart from the areas of online and streaming algorithms, there are several models of optimization problems on uncertainty, such as *stochastic optimization* or *robust optimization*. We refer interested readers to some papers [3, 4, 5, 7] for more information. Some ideas of our work come from the area of kernelization from parameterized complexity, which is about preprocessing algorithms for NP-hard problems. Some particular results from this field inspired our work, namely a result of Pilipczuk et al. [22] on Steiner trees connecting terminals on the outer face of a plane graph, and a result of Kratsch and Wahlström [16] on cut-covering sets (which is also used in Section 3).

2 Preliminaries

Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. If U is a set, then $\binom{U}{k}$ are all its subsets of size k.

We mostly use graph notation as given by Diestel [8]. For a graph $G = (V, E)$ and set of edges $F \subseteq E$, let $V(F)$ denote the vertices incident with F. For a vertex $v \in V$ we denote by $\delta(v)$ the set of edges that are incident to v; thus $\delta(v) = \{e \in E \mid v \in e\}$. Let $D = (V, A)$ be a directed graph and $v \in V$ be a vertex of D. For a vertex $v \in V$ we denote by $\delta^-(v)$ (resp. $\delta^+(v)$) the set of arcs $(u, v) \in A$ (resp. $(v, u) \in A$) with $u \in V$. Let $N^-(v)$ (resp. $N^+(v)$) denote the in-neighbors (resp. out-neighbors) of v. If F is a subset of V then we use $D - F$ to denote the graph obtained from D by deleting all vertices in F and $D[F]$ to denote the graph induced in D by F. If F is a subset of A then we use $D - F$ to refer the graph obtained from D by removing all edges in F. If f is a flow in D then we use $|f|$ to denote the value of f.

We use standard matroid notation as given by Oxley [21]. A *matroid* is a pair $(E, \mathcal{I})$, where E is a finite set of elements, called *ground set*, and $\mathcal{I}$ is a family of subsets of E which are called *independent sets* such that:

1. $\emptyset \in \mathcal{I}$.
2. If $A \in \mathcal{I}$, then for every subset $B \subseteq A$ we have $B \in \mathcal{I}$.
3. If A and B are two independent sets in $\mathcal{I}$ and $|A| > |B|$, then there is an element $e \in A \setminus B$ such that $B \cup \{e\} \in \mathcal{I}$.

Given a matroid $\mathcal{M} = (E, \mathcal{I})$ and $F \subseteq E$, we denote by $\mathcal{M}/F$ the matroid obtained from $\mathcal{M}$ by contracting F. The *rank function* corresponding to $\mathcal{M}$ is a function $r\colon 2^E \to \mathbb{N}$ which is defined by $r(S) := \max\{|I| : I \subseteq S, I \in \mathcal{I}\}$.

Let us recall some well-known types of matroids. For any matrix A over some field F there is an associated matroid $\mathcal{M}$ on the set of columns with independence defined by linear independence of the column vectors. We then say that A *represents* $\mathcal{M}$, and representable matroids are also called *linear matroids*. Let $U_1, \dots, U_m$ be a collection of pairwise disjoint sets and $d_1, \dots, d_m$ be integers with $0 \leq d_i \leq |U_i|$ for each $i = 1, \dots, m$. If we set $E = \cup_{i=1}^m U_i$ and $\mathcal{I} = \{I \subseteq E \colon |I \cap U_i| \leq d_i \text{ for all } i = 1, \dots, m\}$ then $(E, \mathcal{I})$ becomes a matroid, and matroids of this form are called *partition matroids*. The family of forests in a graph $G = (V, E)$ forms a matroid on E. Matroids that can be represented in this way are called *graphic matroids*. Let $G = (V, E)$ be a graph and S and T be two subsets of V. In the set T, we define a subset $U \subseteq T$ to be independent if there are $|U|$ vertex-disjoint paths from S onto U in G. Then this constructs a matroid on T, and matroids of this type are called *gammoids*.

3 Maximum flow

In the MAXIMUM FLOW problem we are given a directed graph $G = (V, A)$, capacities $c \colon A \to \mathbb{N}$, and two vertices $s, t \in V$; the task is to find a flow $f \colon A \to \mathbb{N}$ of maximum value. We consider preprocessing for the MAXIMUM FLOW problem for the case that capacity respectively presence of arcs in a set $F \subseteq A$ is not yet known. Results of this type were previously obtained by Assadi et al. [2]. We tighten one of their lower bounds and give a variant for the case of preserving the parity of the maximum flow. Moreover, we obtain a positive result for the case of $|F|$ uncertain arcs of unit capacity, which is a subroutine for our result for GAMMOID INTERSECTION. Crucially, the output of the latter is again an instance of MAXIMUM FLOW; it also implies a slightly improved encoding size in bits when represented by a matrix. In this section all capacities are integers, implying that there always exists an integral maximum flow. We tacitly assume that all considered maximum flows are integral.

▶ **Theorem 1.** *There is no algorithm that, given an instance $G = (V, A)$, $c \colon A \to \mathbb{N}^+$, and vertices $s, t \in V$ of* MAXIMUM FLOW *together with a set $F \subseteq A$, returns an encoding that requires fewer than $2^{|F|}$ bits, from which we can correctly extract the value of a maximum s,t-flow in $G - (F \setminus F')$ for all $F' \subseteq F$.*

It can be checked that the theorem is tight for the family of graphs used for the lower bound construction. The point is that the relevant information about each graph consists only of $2^{|F|}$ bits, and all flow values can be computed once the graph is known. In general, the lower bound should not be seen as the question of outright storing the $2^{|F|}$ results but regarding any way of storing enough information to compute requested values.

For an arbitrary graph with uncertain arcs F it is not clear whether $2^{|F|}$ bits are sufficient information to compute all flow values. Nevertheless, it is clearly enough space to store the *parities* of the maximum flows, and we can show that this is tight: By reinspecting our proof we can see that it can be adapted to the parity question. The key point is that the matrix used in the proof of Theorem 1 also has full rank over $GF(2)$, which can be easily verified.

▶ **Corollary 2.** *The lower bound of Theorem 1 is tight for* MAXIMUM FLOW PARITY, *i.e., given a graph $G = (V, A)$, $c \colon A \to \mathbb{N}^+$, and vertices $s, t \in V$, there is an encoding of $2^{|F|}$ bits, from which we can correctly extract the parity of the value of a maximum s,t-flow in $G - (F \setminus F')$ for all $F' \subseteq F$. There is no algorithm that returns a smaller encoding.*

The construction in the proof of Theorem 1 relies on uncertain edges with a high capacity. The following positive result shows that this is necessary since we can achieve an encoding to size polynomial in $|F| + l$ where l is the maximum capacity of any uncertain edge. Moreover, this also works if the capacity of edges in F can be instantiated to any value in $\{0, 1, \dots, l\}$. The preprocessing can be performed by a randomized polynomial-time algorithm.

Let us first observe that we may easily reduce this question to the case that $l = 1$, which is equivalent to having a set $\hat{F}$ of edges of capacity 1 each that may or may not be present in the final instance: It suffices to replace each edge of F by l parallel edges of capacity 1 in $\hat{F}$. Setting the capacity of some $e \in F$ to a value $c(e) \in \{0, 1, \ldots, l\}$ is equivalent to making exactly $c(e)$ copies of e in $\hat{F}$ available (We can avoid getting a multigraph by subdividing the edges and putting one of the newly obtained edges in $\hat{F}$). Note that we make no assumption about the capacity of other edges, but the returned graph will have unit capacities.

▶ **Theorem 3.** *Let $G = (V, A)$ be a directed graph, $s, t \in V$, $F \subseteq A$, and $c\colon A \to \mathbb{N}$ be a capacity function such that $c(F) \equiv 1$. There exists a randomized polynomial-time algorithm that, given a network (G, s, t, c) and a set $F \subseteq A$ as above, returns a network $(G' = (V', A'), s, t, c')$ with $F \subseteq A'$, $c' \equiv 1$, $|V'| \in \mathcal{O}(|F|^3)$, and an integer $\alpha \in \mathbb{N}$ such that for any $F' \subseteq F$, the network $(G - F', s, t, c|_{A \setminus F'})$ has a maximum s,t-flow of value β if and only if the network $(G' - F', s, t, c'|_{A' \setminus F'})$ has maximum s,t-flow of value $\beta' = \beta - \alpha$. Here, α is the value of a maximum s,t-flow in $G - F$.*

As mentioned before, differing from the work of Assadi et al. [2] we are interested in finding a small instance of the same problem. In the full version we discuss how our resulting network (G', s, t, c') can be compressed into size $\mathcal{O}(|F|^3)$ bits which improves upon the upper bound of Assadi et al. [2].

The result is obtained by analyzing the *residual graph* with respect to any maximum flow f of $G - F$, i.e., not using any arc of F. We transform this graph in such a way that the cut-covering results of Kratsch and Wahlström [16] can be applied. Crucially, the residual graph has a small minimum s,t-cut size and its maximum flow with respect to deletion of $F' \subseteq F$ is equal to the possible additional flow in $G - F'$ as compared to f. Using the cut-covering set, a small equivalent instance can be obtained. A special case of Theorem 3, which is used for the gammoid intersection result, is the following.

▶ **Corollary 4.** *There exists a randomized polynomial-time algorithm that, given a directed graph $D = (V, A)$, two vertices $s, t \in V$ and a set $F \subseteq V \setminus \{s, t\}$, returns a directed graph $D' = (V', A)$ with $F \subseteq V'$, $|V'| \in \mathcal{O}(|F|^3)$ and an integer $\alpha \in \mathbb{N}$ such that $F \cup \{s, t\} \subseteq V'$ and for any $F' \subseteq F$, the maximum number of internally vertex-disjoint s,t-paths in $D - F'$ is β if and only if the maximum number of internally vertex-disjoint s,t-paths in $D' - F'$ is $\beta - \alpha$. Here, α is the maximum number of vertex-disjoint paths in $D - F$.*

4 Matroid intersection

In this section, we consider the well-known Matroid Intersection problem. In this problem, we are given two matroids $\mathcal{M}_1 = (E, \mathcal{I}_1)$ and $\mathcal{M}_2 = (E, \mathcal{I}_2)$ over the same ground set E. Our task is to determine the largest size of a set $I \subseteq E$ which is independent in both $\mathcal{M}_1$ and $\mathcal{M}_2$. We are interested in the case where the availability of a set of elements $F \subseteq E$ is uncertain and we want to know how much we can compress the input without knowing F. We obtain a positive result for the linear matroid intersection problem, where we construct a matrix from which we can compute the size of a common independent set. Furthermore, for two special cases of matroid intersection, gammoid intersection and rooted arborescence, we show how to obtain small instances of the same problem.

4.1 Linear matroid intersection

In this section we discuss preprocessing for the intersection problem of linear matroids over the same ground set $E = \{e_1, e_2, \ldots, e_m\}$ with respect to a set $F = \{e_1, e_2, \ldots, e_k\} \subseteq E$ of elements whose presence in the final instance is uncertain. Throughout, we will refer to the two matroids in question as M_1 and M_2, and let A and B be $r \times m$ matrices over some field $\mathbb{F}$ that represent M_1 and M_2; here $r \leq m$ is an upper bound on the ranks of M_1 and M_2.

In previous work, e.g., for MINIMUM SPANNING TREE, it was possible to compute for each $F' \subseteq F$ an optimal solution that avoids F': The preprocessing would identify a set $X \subseteq E$ such that there exist solutions $X \cup Y_{F'}$ for each $F' \subseteq F$ where $Y_{F'}$ can be obtained from the outcome of the preprocessing. Unfortunately, this can be easily ruled out for LINEAR MATROID INTERSECTION: Let $G = (A \dot{\cup} B, E)$ be a bipartite graph and $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ be two linear matroids over E with $\mathcal{I}_1 = \{E' \subseteq E \colon |\delta(v) \cap E'| \leq 1 \, \forall v \in A\}$ and $\mathcal{I}_2 = \{E' \subseteq E \colon |\delta(v) \cap E'| \leq 1 \, \forall v \in B\}$. It can be checked that a set E' is independent in M_1 and M_2 if and only if E' is a matching in G. Consider the bipartite graph that is a cycle of length $2n$. This graph has two disjoint maximum matchings E_1 and E_2. Let $F = \{e_1, e_2\}$ with $e_2 \in E_1$ and $e_1 \in E_2$. Now, the unique maximum common independent set in $M_1 - \{e_i\}$ and $M_2 - \{e_i\}$ is E_i for $i = 1, 2$. Thus, we cannot hope to identify $X \subseteq E$ that is shared by optimal solutions. Generally, the size of E cannot be bounded in terms of $|F|$ and as just seen the union of two maximum independent sets in two different instantiations can be the set E; hence we cannot hope to report for each $F' \subseteq F$ a largest common independent set $I \subseteq E \setminus F'$ from any preprocessed instance of size bounded in terms of $|F|$.

Instead, we will show that the *size* of a maximum common independent set in $M_1 - F'$ and $M_2 - F'$, for all $F' \subseteq F$, can be computed from (the rank of) an appropriate $2|F| \times 2|F|$ matrix M that is derived from A and B, which represent M_1 and M_2. To construct M we use a theorem due to Harvey [15]. Before stating the theorem, we need to introduce some notation. For each $J \subseteq E$ we define an $|E| \times |E|$ matrix $T(J)$ by

$$T(J)_{ij} := \begin{cases} 0 & \text{if } i \neq j \text{ or } i = j \in J, \\ t_i & \text{if } i = j \notin J, \end{cases}$$

where each t_i is an indeterminate. Next we define the matrix $Z(J)$ as

$$Z(J) := \begin{pmatrix} 0 & A \\ B^T & T(J) \end{pmatrix}.$$

By $\lambda(J)$ we denote the maximum cardinality of a set that is independent in the contracted matroids M_1/J and M_2/J. Later we consider these two matrices for the case that $J = \emptyset$; we define the shorthands $T = T(\emptyset)$, $Z = Z(\emptyset)$, and $\lambda = \lambda(\emptyset)$.

▶ **Theorem 5** (Harvey [15]). *Let M_1 and M_2 two linear matroids of rank r over the same ground set E. Let A (resp. B) be the $r \times m$ matrix that represents M_1 (resp. M_2). Let $r_1 \colon E \to \mathbb{N}$ and $r_2 \colon E \to \mathbb{N}$ the rank functions of M_1 (resp. M_2). For any $J \subseteq E$, we have* $\operatorname{rank}(Z(J)) = m + r_1(J) + r_2(J) - |J| + \lambda(J)$.

To determine the maximum cardinality of a set that is independent in M_1 and M_2, we use Theorem 5 for the case where $J = \emptyset$. For this case it implies $\operatorname{rank}(Z) = m + \lambda$. For $J = \emptyset$, this result was also obtained by Geelen [14] and it follows from the connection between matroid intersection and the Cauchy-Binet formula [24] (see also Murota [20, Remark 2.3.37]).

During the construction of the desired $2|F| \times 2|F|$ matrix M, we will perform many elementary row and column operations. This can lead to entries which are polynomials of large degree, because matrix T contains m indeterminates. To avoid this we replace some indeterminates by random elements from a field $\mathbb{F}$; this was also used in previous work [2, 25]. Performing row and column operations on the resulting matrix can cause elements to vanish over $\mathbb{F}$, which can reduce the rank and thus lead to a wrong result. We bound the resulting error probability by using the Zippel-Schwartz lemma [23, 27].

Essentially, the idea of the proof is to derive an equivalent matrix whose rank can easily be computed from the rank of a couple of submatrices, where one of the submatrices is small and captures the uncertainty. Thus, we only need to keep the latter and store the rank of the other submatrices. A similar idea is used by Wahlström [25] and by Assadi et al. [2]. In the dynamic sketching scheme of Assadi et al. they have as initial matrix the Tutte matrix, where the uncertainty is contained in a $k \times k$ sub-matrix that contains indeterminates which can be set to zero in the final instance. In our case the initial matrix the is matrix Z which contains the sub-matrices A and B^T. The crucial difficulty is that we need the matrix Z from Harvey's theorem for each choice of $F' \subseteq F$ (or at least a matrix of same rank relative to an offset). Since each F' corresponds to a deletion of pairs of rows and columns, this is not simply handled by a small number of indeterminates that can be set to zero. Also, we cannot avoid using elements of these rows and columns for cancellation. We have to prove that our construction is *independent of the choice of* $F' \subseteq F$. This means, that taking the matrix Z for the matroids without elements of F' (which is same as deleting those rows and columns from Z) and applying our transformation yields the same result as first applying the transformation and then deleting rows and columns corresponding to F'.

To formally state our theorem and to describe the transformation steps let us denote by $W[F'^C, F'^C]$ the sub-matrix of W that contains all rows and columns that do not correspond to elements in $F' \subseteq F$, where W is the matrix in the current step; this means we delete the rows and columns that contain an indeterminate t_i with $e_i \in F'$.

▶ **Theorem 6.** *Let M_1 and M_2 be two linear matroids of rank r over the same ground set $E = \{e_1, e_2, \ldots, e_m\}$. Let A and B be $r \times m$ matrices over the same field $\mathbb{F}$ with $|\mathbb{F}| \geq N = 2^p(r+k)2^k$ that represent M_1 and M_2. Let $F = \{e_1, e_2, \ldots, e_k\} \subseteq E$ be the set of uncertain elements. There exists a randomized polynomial-time algorithm that, given the representations A and B of matroids M_1 and M_2 and $F \subseteq E$, returns a $2k \times 2k$ matrix M, and $\alpha \in \mathbb{N}$ such that with probability at least $1 - 2^{-p}$ for all $F' \subseteq F$, the maximum cardinality of a set that is independent in $M_1 - F'$ and $M_2 - F'$ is equal to* $\operatorname{rank}(M[F'^C, F'^C]) + \alpha - |F \setminus F'|$.

Proof sketch. In our case columns of A (resp. B) correspond to elements in E. For a set $X \subseteq E$ we denote by $A[X, \cdot]$ (resp. $B^T[\cdot, X]$) the matrix that contains the columns (resp. rows) that correspond to set X. Note that both row i and column i of matrix T correspond to the element $e_i \in E$, since $T[i,i] = t_i$. Therefore, by $T[X,X]$ we denote the submatrix of T that is induced by the rows an columns that correspond to the set X. To make sure that our preprocessing works for all choices of $F' \subseteq F$ we have to treat columns from A and B that correspond to elements in F differently from the remaining ones. To this end let $A_F = A[\cdot, F]$, $A_{E\setminus F} = A[\cdot, F^C]$, $B_F = B[\cdot, F]$, $B_{E\setminus F} = B[\cdot, F^C]$, $T_F = T[F,F]$ and $T_{E\setminus F} = T[F^C, F^C]$, i.e., $A = (A_F\, A_{E\setminus F})$, $B = (B_F\, B_{E\setminus F})$ and $T = \operatorname{diag}(T_F, T_{E\setminus F})$. We construct the matrix M in five steps, which we outline below. Due to space constrains, we only show how the construction of matrix M looks like; the crucial point of the proof, which is to show that the construction is independent on the choice of $F' \subseteq F$, is deferred to full version.

$$Z = \begin{pmatrix} 0 & A_F & A_{E\setminus F} \\ B_F^T & T_F & 0 \\ B_{E\setminus F}^T & 0 & T_{E\setminus F} \end{pmatrix} \qquad \xrightarrow{\text{step 1}} Z_1 = \begin{pmatrix} T_F & B_F^T \\ A_F & -A_{E\setminus F} T_{E\setminus F}^{-1} B_{E\setminus F}^T \end{pmatrix}$$

$$\xrightarrow{\text{step 2}} Z_2 = \begin{pmatrix} T_F & B_F^T \\ A_F & \widetilde{X} \end{pmatrix} \qquad \xrightarrow{\text{step 3}} Z_3 = \begin{pmatrix} T_F & {B_F}^T C \\ RA_F & D \end{pmatrix}$$

$$\xrightarrow{\text{step 4}} Z_4 = \begin{pmatrix} T_F - {B'}^T A' & {B''}^T \\ A'' & 0 \end{pmatrix} \qquad \xrightarrow{\text{step 5}} Z_5 = \begin{pmatrix} T_F - {B'}^T A' & \widetilde{B}^T \\ \widetilde{A} & 0 \end{pmatrix}$$

Step 1 In the first step, we reduce Z to an $(r+k)\times(r+k)$ matrix Z_1 such that the uncertain rows and columns remain unaffected; furthermore it holds that the rank of matrix Z equals the rank of matrix Z_1 plus $m-k$. We obtain matrix Z_1 by zeroing out the matrices $A_{E\setminus F}$ and $B_{E\setminus F}^T$ using the invertible matrix $T_{E\setminus F}$. Afterwards, we delete the rows and columns that contain matrix $T_{E\setminus F}$; this reduces the rank by $m-k$.

Step 2: We replace the indeterminate t_i for $k < i \leq m$ by random elements to avoid polynomials of large degree; the entries of matrix Z_1 are polynomials of degree at most one and this leads to an error probability of at most 2^{-p}. Denote the resulting sub-matrix by $\widetilde{X}$ and the complete matrix by Z_2. Note that we could replace the indeterminates before Step 1; but then we have to choose elements from a set of size at least $2^p(r+m)2^k$ to obtain the same error probability (instead of a set of size $N = 2^p(r+k)2^k$).

Step 3: Let $h = \operatorname{rank}(\widetilde{X})$. We apply elementary row and column operations to turn $\widetilde{X}$ into a diagonal matrix $D = \operatorname{diag}(1,\ldots,1,0,\ldots,0)$ with only h non-zero entries. It is well known that there exist two matrices R and C such that $D = R\widetilde{X}C$, where R is the product of all row operations and C the product of all column operations. Let Z_3 be the matrix after applying these row and column operations to Z_2. Matrix Z_2 and Z_3 have the same rank, because we only apply elementary row and column operations. Note that neither matrix R nor matrix C depend on the choice of $F' \subseteq F$, because $\widetilde{X}$ does not depend on the choice of $F' \subseteq F$. One can show that a column j of $R \cdot A_F$ (resp. $C^T \cdot B_F$) only depends on entries of column j of A_F (resp. B_F) and matrix entries that do not depend on the choice of $F' \subseteq F$; thus this transformation is independent on the choice of $F' \subseteq F$.

Step 4: Let $A' = (R\cdot A_F)[[h],\cdot]$ and let $A'' = (R\cdot A_F)[\{h+1,h+2,\ldots,r\},\cdot]$, i.e. $R\cdot A_F = \begin{pmatrix} A' & A'' \end{pmatrix}^T$. Analog we define the $h\times k$ sub-matrix B' and the $(r-h)\times k$ sub-matrix B'' of $B_F^T \cdot C$, i.e. $B_F^T \cdot C = \begin{pmatrix} B'^T & B''^T \end{pmatrix}$. Note that A' (resp. ${B'}^T$) correspond to the rows (resp. columns) where matrix D has a non-zero entry. We zero-out matrices A' and B'^T using the identity matrix I_h. Afterwards, we delete the rows and columns that contain matrix I_h. We denote the resulting matrix by Z_4. By a well-known fact from linear algebra we have that matrix Z_4 has rank l if and only if matrix Z_3 has rank $l+h$.

Step 5: Since A'' (resp. B'') is an $(r-h)\times k$ matrix, at most k rows can be linear independent. We pick a maximum set of linear independent rows from A'' (resp. B''); if less than k rows are independent, then we arbitrarily pick from the remaining rows or add zero rows until we have k rows. Denote this matrix by $\widetilde{A}$ (resp. $\widetilde{B}$). This results in the $2k \times 2k$ matrix Z_5. Note that matrix Z_4 and matrix Z_5 have the same rank, because deleting rows that are dependent on the rows in $\widetilde{A}$ (resp. columns in $\widetilde{B}^T$) corresponds to row (resp. column) operations that zero-out these rows (resp. columns) and this does not change the rank of a matrix.

Altogether, we have constructed a $2|F| \times 2|F|$ matrix $M = Z_5$ and, as we show in the full version, for all $F' \subseteq F$ the equation $\operatorname{rank}(M[{F'}^C, {F'}^C]) + \beta = \operatorname{rank}(Z[{F'}^C, {F'}^C])$ holds with an error probability of at most 2^{-p} where $\beta = h + m - k$. Since the matroid $M_1 - F'$ (resp.

the matroid $M_2 - F'$) is represented by the matrix $A[\cdot, {F'}^C]$ (resp. the matrix $B[\cdot, {F'}^C]$), and the matroids M_1 and M_2 are defined over the same ground set of size $m - |F'|$ it follows from Theorem 5 that $\mathrm{rank}(Z[{F'}^C, {F'}^C]) = (m - |F'|) + \lambda_{F'}$, where $\lambda_{F'}$ denotes the maximum cardinality of a set that is independent in $M_1 - F'$ and $M_2 - F'$. Combining these two equations result in equation $\lambda_{F'} = \mathrm{rank}(M[{F'}^C, {F'}^C]) + h - |F \setminus F'|$. All operations required to obtain the matrix M can be performed in polynomial time and the matrix M together with integer $\alpha = h$ satisfy the required properties in Theorem 6. This completes the proof. ◀

4.2 Gammoid intersection

In this section, we consider the Matroid Intersection problem for the case that both matroids are gammoids. Since gammoids are also linear matroids (cf. [19]) we could apply Theorem 6 to obtain an encoding of size polynomial in the uncertain part. We show how to compute an instance of the gammoid intersection problem instead of an arbitrary encoding. In the same way, the following preprocessing result for the problem of intersecting two gammoids can also be applied to special cases of gammoids such as *transversal matroids* and *partition matroids*, but extra work would be needed to ensure that the output is again, e.g., a pair of transversal matroids rather than general gammoids.

▶ **Theorem 7.** *There exists a randomized polynomial-time algorithm that, given two gammoids $\mathcal{M}_1 = \mathcal{M}(G_1, S_1, T)$ and $\mathcal{M}_2 = \mathcal{M}(G_2, S_2, T)$ together with a subset $F \subseteq T$, returns two new gammoids $\mathcal{M}'_1 = \mathcal{M}(G'_1, S'_1, T')$ and $\mathcal{M}'_2 = \mathcal{M}(G'_2, S'_2, T')$ over a ground set T' of size $\mathcal{O}(|F|^3)$ together with an integer $\alpha \in \mathbb{N}$ such that $F \subseteq T'$ and for every $F' \subseteq F$, $\mathcal{M}_1 - F'$ and $\mathcal{M}_2 - F'$ have a maximum common independent set of size l if and only if $\mathcal{M}'_1 - F'$ and $\mathcal{M}'_2 - F'$ have a maximum common independent set of size $l - \alpha$.*

Proof. Because T appears in both graphs, G_1 and G_2, for each vertex $v \in T$ we rename it in G_1 by v_1 and G_2 by v_2 respectively. We obtain two sets T_1, T_2 where $T_i = \{v_i : v \in T\}$ plays the role of T in G_i and $\mathcal{M}_i$. In order to apply Corollary 4, we construct a graph G as follows. We first reverse all arcs in G_2 to obtain $\overleftarrow{G_2}$ and take the union of G_1 and $\overleftarrow{G_2}$. For each $v \in T$, we create a new vertex in G, also named v, and add two arcs $(v_1, v), (v, v_2)$ to G. Then we create a source s and a sink t together with arcs from s to each vertex in S_1 and arcs from each vertex in S_2 to t. Thus, we obtain a graph $G = (V, E)$ such that $T \subseteq V$ is an (s,t)-cut and there is no arc from $\overleftarrow{G_2}$ to G_1 in G.

▶ **Claim 1.** *For all $F' \subseteq F$, the maximum cardinality of a common independent set of $\mathcal{M}_1 - F'$ and $\mathcal{M}_2 - F'$ is equal to the maximum number of internally vertex-disjoint s,t-paths in $G - F'$.*

We apply the algorithm given by Corollary 4 for G and F to compute a graph $G' = (V', A')$ with $\mathcal{O}(|F|^3)$ vertices and an integer $\alpha' \in \mathbb{N}$ such that $F \subseteq V'$ and for any $F' \subseteq F$, the maximum number of internally vertex-disjoint s,t-paths in $G - F'$ is l if and only if the maximum number of internally vertex-disjoint s,t-paths in $G' - F'$ is $l - \alpha'$. Using graph G' we construct the two new gammoids $\mathcal{M}'_1$ and $\mathcal{M}'_2$. We obtain the graph G'_1 by adding two new vertices $s_v, \hat{v}$ as well as the arcs $(s_v, v), (s_v, \hat{v})$ to G' for all vertices $v \in F$. Let $S_F := \{s_v : v \in F\}$, and let N^+ (resp. N^-) denote the out-neighbors of s (resp. in-neighbors of t). The first gammoid $\mathcal{M}'_1 = \mathcal{M}(G'_1, S'_1, T')$ is defined by the graph G'_1, the set of sources $S'_1 = N^+ \cup S_F$ and the ground set $T' = N^- \cup F \cup \hat{F}$ and the second gammoid $\mathcal{M}'_2 = \mathcal{M}(G'_2, S'_2, T')$ is defined by the graph $G'_2 = (S'_2 \cup T', \{(s_v, v), (s_v, \hat{v}) : v \in F\})$, the set of sources $S'_2 = S_F \cup N^-$ and the ground set T'. For each $F' \subseteq F$, let $\hat{F}' := \{\hat{v} : v \in F'\}$.

▶ **Claim 2.** *For every $F' \subseteq F$, the maximum number of internally vertex-disjoint s,t-paths in $G' - F'$ is h if and only if the maximum cardinality of a common independent set of $\mathcal{M}'_1 - \hat{F}'$ and $\mathcal{M}'_2 - \hat{F}'$ is $h + |F|$.*

We conclude that for every $F' \subseteq F$, the maximum cardinality of a common independent set in $\mathcal{M}_1 - F'$ and $\mathcal{M}_2 - F'$ is l if and only if the maximum cardinality of a common independent set in $\mathcal{M}'_1 - \hat{F}'$ and $\mathcal{M}'_2 - \hat{F}'$ is $l - \alpha' + |F|$. Note that in the construction of $\mathcal{M}'_1$ and $\mathcal{M}'_2$, $\hat{F}$ is a copy of F, so if we identify $\hat{F}$ with the set F in the input gammoids and set $\alpha = \alpha' - |F|$, then we obtain the desired result. ◀

4.3 Rooted arborescence

In this subsection we consider the Rooted Arborescence problem, where we are given a directed graph $D = (V, A)$ with root r and we need to determine whether there exists an r-arborescence in D. An *arborescence* with root $r \in V$, or an *r-arborescence*, is an arc set $A' \subseteq A$ such that A' is a spanning tree if considered as an undirected subgraph and every $v \in V$ is reachable from r via arcs of A', i.e., there is a directed path from r to v using only arcs of A'. We are again interested in the case that there is uncertainty in the input, more precisely, that there are some arcs or vertices whose presence is not known. Rooted Arborescence can be considered as a special case of Matroid Intersection. Indeed, let $\mathcal{M}_1$ be the graphic matroid defined on the undirected underlying graph corresponding to D and $\mathcal{M}_2 = (A, \mathcal{I})$ where $\mathcal{I} = \{S \subseteq A : |S \cap \delta^-(v)| \leq 1 \text{ for all } v \in V \setminus r\}$. It can be checked that D has an r-arborescence if and only if $\mathcal{M}_1$ and $\mathcal{M}_2$ have a common independent set of size $|V| - 1$. Uncertainty of some elements in the two matroids corresponds to uncertain appearance of some arcs in D. By constructing reduction rules based on a well-known property of arborescences, we obtain the following result.

▶ **Theorem 8.** *There exists a polynomial-time algorithm that, given a directed graph $D = (V, A)$ with a root $r \in V$ and an arc set $F \subseteq A$, returns a directed graph $D' = (V(F) \cup \{r\}, A')$ with $F \subseteq A'$ such that for every $F' \subseteq F$, the graph $D - F'$ has an r-arborescence if and only if $D' - F'$ has an r-arborescence.*

In the next theorem, we consider the case that there are some uncertain vertices in our input graph and prove a lower bound for it. The construction is similar to the one used for maximum flow by Assadi et al. [2]. Note that this is not a special case of Matroid Intersection with uncertainty about ground set elements.

▶ **Theorem 9.** *There is no algorithm that, given an instance of* Rooted Arborescence *with k uncertain vertices, returns an encoding that requires fewer than $\binom{k}{\lceil k/2 \rceil}$ bits from which we can correctly extract the answer for all 2^k instantiations of the input instance.*

5 Conclusion

We have continued the study of preprocessing under uncertainty initiated by Assadi et al. [2] (who called their notion *dynamic sketching*) and Fafianie et al. [12]. Our main focus was on preprocessing for matroid intersection problems when the presence of certain ground set elements is uncertain. We obtained positive results for (i) intersecting two linear matroids, (ii) intersecting two gammoids, and (iii) the Rooted Arborescence problem. For the latter two problems our preprocessing returns an instance of the respective problem; for the former, the output is in form of a matrix. Additionally, we have revisited maximum flow, also studied by Assadi et al. [2]. We have tightened a lower bound construction and gave a variant of the result for preserving the parity of maximum flows. Furthermore, we obtained a positive result for the case that a small amount of capacity is uncertain, with output again an instance of MAXIMUM FLOW, which is used for the gammoid intersection result. Deriving a matrix encoding from this yields bitsize $\mathcal{O}(|F|^3)$, improving slightly over $\mathcal{O}((k+C')^4 \log n)$ [2].

It would be interesting whether our result for LINEAR MATROID INTERSECTION can be generalized to the weighted case, possibly with uncertain weights. Similarly, one can try to extend the result to arbitrary matroids that are given by an independence oracle. Generally, preprocessing under uncertainty (or dynamic sketching) in the oracle setting is interesting, as the presence of oracles precludes the lower bounds based on MEMBERSHIP.

References

1 Martin Aigner and Thomas A Dowling. Matching theory for combinatorial geometries. *Transactions of the American Mathematical Society*, 158(1):231–245, 1971.

2 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Val Tannen. Dynamic sketching for graph optimization problems with applications to cut-preserving sketches. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 52–68. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. `doi:10.4230/LIPIcs.FSTTCS.2015.52`.

3 Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011. `doi:10.1137/080734510`.

4 Hans-Georg Beyer and Bernhard Sendhoff. Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33):3190–3218, 2007.

5 Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: an international journal*, 8(2):239–287, 2009.

6 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

7 George B. Dantzig. Linear programming under uncertainty. *Management Science*, 50(12-Supplement):1764–1769, 2004. `doi:10.1287/mnsc.1040.0261`.

8 Reinhard Diestel. *Graph theory (Graduate texts in mathematics)*. Springer Heidelberg, 2005.

9 Randall Dougherty, Chris Freiling, and Kenneth Zeger. Network coding and matroid theory. *Proceedings of the IEEE*, 99(3):388–405, 2011.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. `doi:10.1007/978-1-4471-5559-1`.

11 Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial structures and their applications*, pages 69–87, 1970.

12 Stefan Fafianie, Stefan Kratsch, and Vuong Anh Quyen. Preprocessing under uncertainty. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 33:1–33:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.STACS.2016.33`.

13 Harold N Gabow and Ying Xu. Efficient theoretic and practical algorithms for linear matroid intersection problems. *Journal of Computer and System Sciences*, 53(1):129–147, 1996.

14 J. F. Geelen. Matching theory. *Lecture Notes from the Euler Institute for Discrete Mathematics and Its Applications*, 2001.

15 Nicholas J. A. Harvey. Algebraic structures and algorithms for matching and matroid problems. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 531–542. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.8.

16 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 450–459. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.46.

17 Stefan Kratsch and Magnus Wahlström. Compression via matroids: A randomized polynomial kernel for odd cycle transversal. *ACM Transactions on Algorithms*, 10(4):20:1–20:15, 2014. doi:10.1145/2635810.

18 Eugene L Lawler. Matroid intersection algorithms. *Mathematical programming*, 9(1):31–56, 1975.

19 Dániel Marx. A parameterized view on matroid optimization problems. *Theor. Comput. Sci.*, 410(44):4471–4479, 2009. doi:10.1016/j.tcs.2009.07.027.

20 Kazuo Murota. *Matrices and matroids for systems analysis*, volume 20. Springer Science & Business Media, 2009.

21 James Oxley. *Matroid Theory.* Oxford University Press, 2011.

22 Marcin Pilipczuk, Michal Pilipczuk, Piotr Sankowski, and Erik Jan van Leeuwen. Network sparsification for Steiner problems on planar and bounded-genus graphs. In *FOCS 2014*, pages 276–285. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.37.

23 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

24 Nobuaki Tomizawa and Masao Iri. Algorithm for determining rank of a triple matrix product axb with application to problem of discerning existence of unique solution in a network. *Electronics & Communications in Japan*, 57(11):50–57, 1974.

25 Magnus Wahlström. Abusing the Tutte Matrix: An Algebraic Instance Compression for the K-set-cycle Problem. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 341–352, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2013.341.

26 Dominic JA Welsh. *Matroid theory.* Courier Corporation, 2010.

27 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5_73.

Ride Sharing with a Vehicle of Unlimited Capacity*

Angelo Fanelli[1] and Gianluigi Greco[2]

1 **CNRS (UMR-6211), France.**
`angelo.fanelli@unicaen.fr`

2 **Department of Mathematics and Computer Science, University of Calabria, Italy.**
`ggreco@mat.unical.it`

Abstract

A ride sharing problem is considered where we are given a graph, whose edges are equipped with a travel cost, plus a set of objects, each associated with a transportation request given by a pair of origin and destination nodes. A vehicle travels through the graph, carrying each object from its origin to its destination without any bound on the number of objects that can be simultaneously transported. The vehicle starts and terminates its ride at given nodes, and the goal is to compute a minimum-cost ride satisfying all requests. This ride sharing problem is shown to be tractable on paths by designing a $O(h \log h + n)$ algorithm, with h being the number of distinct requests and with n being the number of nodes in the path. The algorithm is then used as a subroutine to efficiently solve instances defined over cycles, hence covering all graphs with maximum degree 2. This traces the frontier of tractability, since **NP**-hard instances are exhibited over trees whose maximum degree is 3.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems.

Keywords and phrases Vehicle Routing, Ride Sharing, Pick up and Delivery Problem.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.36

1 Introduction

Ride Sharing. Vehicle routing problems have been drawn to the attention of the research community in the late 50's [8]. Since then, they have attracted much attention in the literature due to their pervasive presence in real-world application scenarios, till becoming nowadays one of the most studied topics in the field of operation research and combinatorial optimization (see, e.g., [24, 29, 10] and the references therein).

Within the broad family of vehicle routing problems, a noticeable class is constituted by the pickup and delivery problems, where a given set of objects, such as passengers or goods, have to be picked at certain nodes of a transportation network and delivered at certain destinations [11]. Pickup and delivery problems can be divided in two main groups [27]. The first group refers to situations where we have a single type of object to be transported, so that pickup and delivery locations are unpaired (see, e.g., [21]). The second group deals, instead, with problems where each transportation request is associated with a specific origin and a specific destination, hence resulting in paired pickup and delivery points (see, e.g., [22, 9]).

* This work was partially supported by the project ANR-14-CE24-0007-01 *"CoCoRICo-CoDec"*. G. Greco's work was also supported by a Kurt Gödel Research Fellowship, awarded by the Kurt Gödel Society. We thank Jérôme Lang, from Université Paris-Dauphine, for introducing the subject to us.

In the paper, we focus on problems of the latter kind, and we deal with the most basic setting where *one vehicle* is available only. The vehicle is initially located at some given source node and it must reach a given destination node by means of a *feasible* ride, that is, of a ride satisfying all requests. The edges of the network are equipped with weights, and the goal is to compute an *optimal* ride, that is, a feasible ride minimizing the sum of the weights of the edges traversed by the vehicle.

Vehicles of Limited Capacity. Ride sharing with one vehicle has attracted much research in the literature and most of the foundational results in the area of vehicle routing precisely refer to this setting. In fact, earlier works have mainly focused on the case where the capacity of the vehicle is bounded by some given constant. In particular, based on whether or not we allow objects to be temporarily unloaded at some vertex of the transportation network, two versions of ride sharing problems emerge: *preemptive* (where drops are allowed) and *non-preemptive* (where drops are not allowed). An orthogonal classification comes, moreover, from the capacity c of the given vehicle. The setting with *unit* capacity ($c = 1$) has received much attention in the literature, where it often comes in the form of a *stacker crane problem* (see [15, 28] and the references therein). A natural generalization is then when the vehicle can carry more than one object at time, that is, when c is any given natural number possibly larger than 1.

Given these two orthogonal dimensions, a total of four different configurations can be studied (cf. [19]). In all the possible configurations, vehicle routing is known to be **NP**-hard [15, 16] when the underlying transportation network is an arbitrary graph. In fact, motivated by applications in a wide range of real-world scenarios, complexity and algorithms for ride sharing problems have been studied for networks with specific topologies, such as path, cycles, and trees. Consider first the unit capacity setting. In this case, ride sharing is known to be polynomial time solvable on both paths [2] and cycles [13], no matter of whether drops are allowed. Moving to trees, instead, the preemptive case remains efficiently solvable [14], while the non-preemptive case becomes **NP**-hard [12]. Consider now the case where $c \geq 1$ holds. Clearly enough, the intractability result over trees established for $c = 1$ still holds in this more general setting. In fact, in this setting, ride sharing appears to be intrinsically more complex. Indeed, it has been shown that the non-preemptive version of the problem is **NP**-hard on all the considered network topologies and that the preemptive version is **NP**-hard even on trees [18]. Good news comes instead when the problem is restricted over paths and cycles in the preemptive case. Indeed, the problem has been shown to be feasible in polynomial time on paths [19]. Moreover, the algorithm by [19] is also applicable to cycles, under the constraint that, for each object, the direction of the transportation (either clockwise, or anticlockwise) is a-priori given. More efficient algorithms are know for paths when the ride starts from one endpoint [18, 23].

Vehicles of Unlimited Capacity. There are application scenarios where the capacity of the vehicle can be better thought as being *unlimited*, as it happens, for instance, when we are transporting intangible objects, such as messages. More generally, we might know beforehand that the number of objects to be transported is less than the capacity of the vehicle; and, accordingly, we would like to use solution algorithms that are more efficient than those proposed in the literature and designed in a way that this knowledge is not suitably taken into account. In fact, the **NP**-hardness results discussed above exploit a given constant bound on the capacity and, hence, they do not immediately apply to the unbounded setting. However, specific reductions have been exhibited showing the **NP**-hardness on general graphs (cf. [30, 3]). Moreover, heuristic methods (see, e.g., [17, 25]) and approximation algorithms (see, e.g., [1, 20]) have been defined, too. On the other hand, a number of tractability results for vehicles with unlimited capacity transporting objects of the same type can be inherited even in the paired context we are considering. Indeed, for cases where such identical objects are initially stored at the same node (or, equivalently, have to be transported to the same destination) [3, 4, 6, 5], efficient algorithms have been designed for transportation networks that are trees and cycles [30]. Moreover, the algorithm for paths proposed by [19] can be still applied over the unlimited capacity scenario. But, it was not explored so far whether better performances can be obtained by means of algorithms specifically designed for vehicles with unlimited capacity.

Contributions. The goal of the paper is to address the above research question, and to study complexity and algorithmic issues arising with ride sharing problems in presence of one vehicle of unlimited capacity. The analysis has been conducted by considering different kinds of undirected graph topologies, which have been classified on the basis of the degree of their nodes. Let n be the number of nodes in the underlying graph, let q be the number of requests (hence, of objects to be transported), and let h denote the number of distinct requests (so, $h \leq q$ and $h \leq n^2$). Then, our results can be summarized as follows:

- In Section 3, we show that optimal rides can be computed in polynomial time over graphs that are *paths*. In particular, an algorithm is exhibited to compute an optimal ride in $O(h \log h + n)$. This improves the $O(qn + n^2)$ bound that we obtain with the state-of-the-art algorithm by Guan and Zhu [19] for vehicles with limited capacity, by naïvely setting the limit to k.
- The design and the analysis of the above algorithm is the main technical achievement of the paper. By using the algorithm as a basic subroutine, we are then able to show in Section 4 that optimal rides can be computed in polynomial time over *cycles* too, formally in $O(m^2 \cdot (h \log h + n))$, with m being the number of distinct nodes that are endpoints of some request, so that $m \leq 2h$ and $m \leq n$. The result has no counterpart in the limited capacity setting, since differently from [19], we do not require that the direction of the transportation of the objects is fixed beforehand.
- Path and cycles completely cover all graphs whose maximum degree is 2. In fact, this value precisely traces the frontier of tractability for the ride sharing problem we have considered, as **NP**-hard instances are exhibited over graphs whose maximum degree is 3 and which are moreover trees.

2 Ride Sharing Scenarios

Let $G = (V, E, w)$ be an undirected weighted graph, where V is a set of nodes and E is a set of edges. Each edge $e \in E$ is a set $e \subseteq V$ with $|e| = 2$, and it is equipped with a cost $w(e) \in \mathbb{Q}^+$. A *ride* π in G is a sequence of nodes $\pi_1, \ldots, \pi_k$ such that $\pi_i \in V$ is the node reached at the *time step* i and $\{\pi_i, \pi_{i+1}\} \in E$, for each i with $1 \leq i \leq k-1$. The time step $k > 0$ is called the *length* of π, hereinafter denoted by $len(\pi)$. The value $\sum_{i=1}^{k-1} w(\{\pi_i, \pi_{i+1}\})$ is the *cost* of π (w.r.t. w) and is denoted by $w(\pi)$. Moreover, $nodes(\pi)$ denotes the set of all nodes $v \in V$ occurring in π.

A *request* on $G = (V, E, w)$ is a pair (s, t) such that $\{s, t\} \subseteq V$. Note that s and t are not necessarily distinct, and they are called the starting and terminating nodes, respectively, of the request. We say that a ride π in G *satisfies* the request (s, t) if there are two time steps i and i' such that $1 \leq i \leq i' \leq len(\pi)$, $\pi_i = s$ and $\pi_{i'} = t$. If $\mathcal{C}$ is a set of requests on G, then

$V_\mathcal{C}$ is the set of all starting and terminating nodes occurring in it. A *ride-sharing* scenario consists of a tuple $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C}\rangle$, where $G = (V, E, w)$ is an undirected weighted graph, (s_0, t_0) is a request on G and $\mathcal{C}$ is a non-empty set of requests.

A ride $\pi = \pi_1, \ldots, \pi_k$ in G is *feasible* for $\mathcal{R}$ if $\pi_1 = s_0$, $\pi_k = t_0$, and π satisfies each request in $\mathcal{C}$. The set of all feasible rides for $\mathcal{R}$ is denoted by $\mathit{feasible}(\mathcal{R})$. A feasible ride π is *optimal* if $w(\pi') \geq w(\pi)$, for each feasible ride π'. The set of all optimal rides for $\mathcal{R}$ is denoted by $\mathit{opt}(\mathcal{R})$.

Let $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C}\rangle$ be a ride-sharing scenario, and let π be a ride in G. Let i and i' be two time steps such that $1 \leq i \leq i' \leq \mathit{len}(\pi)$. Then, we denote by $\pi[i, i']$ the ride $\pi_i, \ldots, \pi_{i'}$ obtained as the sequence of the nodes occurring in π from time step i to time step i'. If π and π' are two rides on G, then we write $\pi' \preceq \pi$ if either $\pi' = \pi$ or, recursively, if there are two time steps i and i' such that $1 \leq i < i' \leq \mathit{len}(\pi)$, $\pi_{i+1} = \pi_{i'}$ or $\pi_i = \pi_{i'-1}$, and $\pi' \preceq \pi[1, i], \pi[i', \mathit{len}(\pi)]$ (informally speaking, π' can be obtained from π by removing a subsequence of nodes).

▶ **Fact 1.** *Let π and π' be two rides such that $\pi' \preceq \pi$. Then: $w(\pi') \leq w(\pi)$; if π' satisfies a request $(s, t) \in \mathcal{C}$, then π satisfies (s, t), too; if π is feasible (resp., optimal) and $V_\mathcal{C} \cap (\mathit{nodes}(\pi) \setminus \mathit{nodes}(\pi')) = \emptyset$, then π' is feasible (resp., optimal), too.*

It is easily seen that computing optimal rides is an intractable problem (**NP**-hard), for instance, by exhibiting a reduction from the well-known traveling salesman problem (see, e.g., [16]). Actually, we can strengthen this result, in a way that suggest to focus our subsequent analysis on ride-sharing scenarios over graphs whose maximum degree is 2 (at most). In fact, these graphs must be either paths or cycles.

▶ **Theorem 2.** *Computing optimal rides is* **NP***-hard over trees whose maximum degree is 3.*

3 Optimal Rides on Paths

In this section we describe an algorithm that, given as input a ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C}\rangle$ where $G = (V, E, w)$ is a *path*, returns an optimal ride for $\mathcal{R}$. In order to keep notation simple, we assume that nodes in V are (indexed as) natural numbers, so that $V = \{1, \ldots, n\}$. Hence, for each node $v \in V \setminus \{n\}$, the edge $\{v, v+1\}$ is in E; and no further edge is in E. Moreover, let us define $\mathsf{left}(\mathcal{R}) = \mathsf{min}_{v \in V_\mathcal{C}} v$ and $\mathsf{right}(\mathcal{R}) = \mathsf{max}_{v \in V_\mathcal{C}} v$, as the extreme (left and right) endpoints of any request in $\mathcal{C}$.

Based on these notions, we distinguish two mutually exclusive cases:

"outer": where either $s_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq t_0$ or $t_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq s_0$; that is, the starting and the terminating nodes s_0 and t_0 are not properly included in the range $\{\mathsf{left}(\mathcal{R}), ..., \mathsf{right}(\mathcal{R})\}$.

"inner": where $\{s_0, t_0\} \cap \{v \in V \mid \mathsf{left}(\mathcal{R}) < v < \mathsf{right}(\mathcal{R})\} \neq \emptyset$; in particular, in this case, $\mathsf{left}(\mathcal{R}) < \mathsf{right}(\mathcal{R})$ necessarily holds.

In the following two subsections we describe methods to address the two different cases, while their complexity will be later analyzed in Section 3.3. A basic ingredient for both methods is the concept of concatenation of rides. Let $\pi = \pi_1, \ldots, \pi_k$ and $\pi' = \pi'_1, \ldots, \pi'_h$ be two rides. Their *concatenation* $\pi \mapsto \pi'$ is the ride inductively defined as follows: if $\pi_k = \pi'_1$ and $h > 1$, then $\pi \mapsto \pi' = \pi_1, \ldots, \pi_k, \pi'_2, \ldots, \pi'_h$; if $\pi_k = \pi'_1$ and $h = 1$, then $\pi \mapsto \pi' = \pi$; if

Algorithm 1: RideOnPath_Outer

```
Input: A scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C}\rangle$, where $G = (V, E, w)$ is a path,
and with $s_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq t_0$ or $t_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq s_0$;
Output: An optimal ride for $\mathcal{R}$;
1 if $s_0 > t_0$ then
2     $\pi \leftarrow$ RideOnPath_Outer($\mathsf{sym}(\mathcal{R})$);
3     return $\mathsf{sym}(\pi)$;
4 else
5     $\mathcal{C}^* = \{(s_1, t_1), \ldots, (s_h, t_h)\} \leftarrow$Normalize($\mathcal{C}$);        /* $s_1 \leq s_2 \cdots \leq s_h$ */
6     return $s_0 \mapsto s_1 \mapsto t_1 \mapsto s_2 \mapsto \ldots \mapsto s_h \mapsto t_h \mapsto t_0$;
```

$\pi_k \neq \pi'_1$, then $\pi \mapsto \pi'$ is defined as the concatenation[1] $\pi \mapsto \bar{\pi} \mapsto \pi'$, where $\bar{\pi} = \pi_k, \ldots, \pi'_1$ is the ride obtained as the sequence of nodes connecting π_k and π'_1 with the smallest length. Note that $\bar{\pi}$ is univocally determined on paths.

3.1 Solution to the "outer" case

Consider Algorithm 1, named RideOnPath_Outer. In the first step, it distinguishes the case $s_0 > t_0$ from the case $s_0 \leq t_0$. Indeed, the former can be reduced to the latter by introducing the concept of *symmetric* scenario. For every node $v \in V$, let $\mathsf{sym}(v) = n - v + 1$. Denote by $\mathsf{sym}(\pi)$ and $\mathsf{sym}(\mathcal{C})$ the ride and the set of requests derived from the ride π and the set of requests $\mathcal{C}$, respectively, by replacing each node v with its "symmetric" counterpart $\mathsf{sym}(v)$. Finally, denote by $\mathsf{sym}(\mathcal{R})$ the scenario $\langle G, (\mathsf{sym}(s_0), \mathsf{sym}(t_0)), \mathsf{sym}(\mathcal{C})\rangle$, referred to as the symmetric scenario of $\mathcal{R}$. Then, the following is immediately seen to hold.

▶ **Fact 3.** *Let π be a ride. Then, π is optimal for $\mathcal{R}$ if, and only if, $\mathsf{sym}(\pi)$ is optimal for $\mathsf{sym}(\mathcal{R})$.*

According to the previous observation, step 5 and step 6 are the core of the computation by addressing the case $s_0 \leq t_0$, where hence $s_0 \leq \mathsf{left}(\mathcal{R}) \leq \mathsf{right}(\mathcal{R}) \leq t_0$. The idea is to reduce the set of requests $\mathcal{C}$ to an "equivalent" set of requests $\mathcal{C}^*$, which presents a simpler structure that we call *normal form*. Formally, let $\mathcal{C}^* = \{(s_1, t_1), \ldots, (s_h, t_h)\}$, and let us say that $\mathcal{C}^*$ is in normal form if $t_i < s_i$ for each $i \in \{1, \ldots, h\}$, and $s_i < t_{i+1}$ for each $i \in \{1, \ldots, h-1\}$. The reduction is performed at step 5, where Normalize is invoked.

The definition of Normalize is shown in Algorithm 2: Step 1 is responsible of filtering out all requests (s, t) such that $s \leq t$. Steps 2 and 3 iteratively "merge" all pairs of requests (s, t) and (s', t') such that $t < s$, $t' < s'$ and $t' \leq t \leq s' \leq s$. Finally, steps 4 and 5 remove all requests (s, t) with $t < s$ and for which there is a request (s', t') such that $t' \leq t < s \leq s'$. It can be shown that the set of requests $\mathcal{C}^*$ returned by Normalize is in normal form and that the optimal ride for the ride-sharing scenario $\langle G, (s_0, t_0), \mathcal{C}^*\rangle$ is an optimal ride also for $\mathcal{R}$. In particular, it can be shown that an optimal ride for $\langle G, (s_0, t_0), \mathcal{C}^*\rangle$ can be obtained by concatenating the rides connecting s_i to t_i, incrementally from $i = 1$ to $i = h$, as it is implemented in step 6 of RideOnPath_Outer. Thus, the following can be established.

▶ **Theorem 4.** *Algorithm* RideOnPath_Outer *is correct.*

[1] The specific order of application of the operator $\mapsto$ is immaterial. Hence, we often avoid the use of parenthesis.

Algorithm 2: NORMALIZE

```
Input: A set $\mathcal{C}$ of requests with $s_0 \le \mathsf{left}(\mathcal{R}) \le \mathsf{right}(\mathcal{R}) \le t_0$;
Output: A set of requests $\mathcal{C}^*$ in normal form and such that $opt(\langle G, (s_0, t_0), \mathcal{C}^* \rangle) \subseteq opt(\mathcal{R})$;
1 $\mathcal{C}^* \leftarrow \mathcal{C} \setminus \{(s,t) \mid s \le t\}$;
2 while exist $(s,t), (s',t') \in \mathcal{C}^*$ such that $t < s$, $t' < s'$, and $t' \le t \le s' \le s$ do
3     $\mathcal{C}^* \leftarrow \mathcal{C}^* \setminus \{(s,t), (s',t')\} \cup \{(s,t')\}$;
4 while exist $(s,t), (s',t') \in \mathcal{C}^*$ such that $t' \le t < s \le s'$ do
5     $\mathcal{C}^* \leftarrow \mathcal{C}^* \setminus \{(s,t)\}$;
6 return $\mathcal{C}^*$;
```

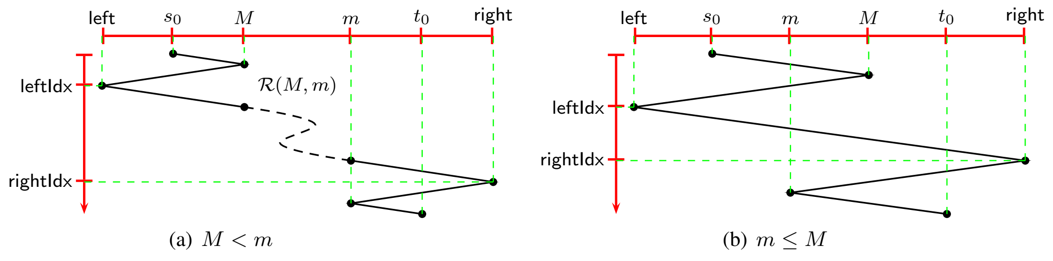


Figure 1 Example of (M,m)-canonical rides.

3.2 Solution to the "inner" case

Let us now move to analyze the "inner" case, where $\{s_0, t_0\} \cap \{v \in V \mid \mathsf{left}(\mathcal{R}) < v < \mathsf{right}(\mathcal{R})\} \neq \emptyset$ holds. Let us introduce some notation. For any feasible ride π, denote by $\mathsf{leftIdx}(\pi)$ (resp., $\mathsf{rightIdx}(\pi)$) the minimum time step i such that $\pi_i = \mathsf{left}(\mathcal{R})$ (resp., $\pi_i = \mathsf{right}(\mathcal{R})$). Note that $\mathsf{leftIdx}(\pi)$ and $\mathsf{rightIdx}(\pi)$ are well defined and, in particular, $\mathsf{leftIdx}(\pi) \neq \mathsf{rightIdx}(\pi)$ holds, since $\mathsf{left}(\mathcal{R}) < \mathsf{right}(\mathcal{R})$. Moveover, for every pair of nodes $x, y \in V$ with $x < y$, define $\mathcal{R}(x,y) = \langle G, (x,y), \{(s,t) \in \mathcal{C} \mid x \le s, t \le y\} \rangle$, that is, the scenario which inherits from $\mathcal{R}$ the graph G and every request with both starting and terminating nodes in the interval $\{x, ..., y\}$, and where the vehicle is asked to start from x and to terminate at y. Notice that, by definition, the set of all nodes occurring in any optimal ride for $\mathcal{R}(x,y)$ is a subset of $\{x, ..., y\}$.

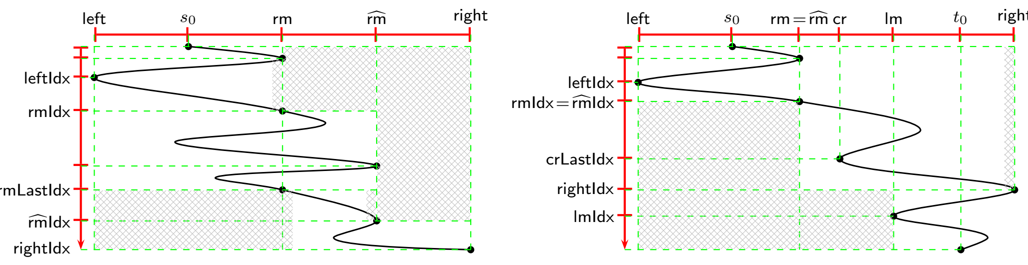
3.2.1 Canonical rides

A crucial role in our analysis is played by the concept of canonical ride, which is illustrated below.

▶ **Definition 5.** *Let $M, m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$ be two nodes. A ride π^{c} in $\mathcal{R}$ is said to be (M,m)-*canonical *if $\pi^{\mathsf{c}} = \pi' \mapsto \pi'' \mapsto \pi'''$ where: $\pi' = s_0 \mapsto M \mapsto \mathsf{left}(\mathcal{R}) \mapsto M$; $\pi'' = M \mapsto \mathsf{right}(\mathcal{R})$ (resp., $\pi'' = \bar{\pi} \mapsto \mathsf{right}(\mathcal{R})$, with $\bar{\pi}$ being an optimal ride for $\mathcal{R}(M,m)$) if $m \le M$ (resp., $m > M$); $\pi''' = \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$.* ☐

Two examples of canonical rides are in Figure 1. Note that if $m \le M$ holds, we can refer without ambiguities to *the* (M,m)-canonical ride, as there is precisely one ride enjoying the properties in Definition 5.

▶ **Fact 6.** *If $m \le M$, then the (M,m)-canonical ride is $s_0 \mapsto M \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$.*



(a) Some critical steps on a ride π^*

(b) Some critical steps on a ride π^{h+1}

Figure 2 Some critical steps of any feasible ride on a path. The gray areas denote the space that no feasible ride can cross for a given time interval.

Instead, whenever $m > M$, there can be more than one canonical ride. In this case, to compute a (M, m)-canonical ride, we need to compute an optimal ride for $\mathcal{R}(M, m)$, which is a scenario fitting the "outer" case and which can be hence addressed via the RideOnPath_Outer algorithm.

As claimed by the following theorem, the notion of canonical ride characterizes the optimal rides for $\mathcal{R}$. In particular, observe that in the following result, we focus on optimal rides π^* such that $\mathsf{leftIdx}(\pi^*) < \mathsf{rightIdx}(\pi^*)$. Indeed, the case where $\mathsf{leftIdx}(\pi^*) \geq \mathsf{rightIdx}(\pi^*)$ will be eventually addressed by working on the symmetric scenario $\mathsf{sym}(\mathcal{R})$, according to the approach discussed in Section 3.1 (see Fact 3).

▶ **Theorem 7.** *Assume that π^* is an optimal ride with $\mathsf{leftIdx}(\pi^*) < \mathsf{rightIdx}(\pi^*)$. Then, there are two nodes $M, m \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, with $s_0 \leq M$ and $m \leq t_0$, such that any (M, m)-canonical ride is optimal, too.*

The proof of the result is rather involved, and we provide an overview here. We repeatedly use *exchange arguments* to gradually transform a given optimal ride without hurting its quality. Hence, let us assume that π^* is a given optimal ride such that $\mathsf{leftIdx}(\pi^*) < \mathsf{rightIdx}(\pi^*)$.

We first define a number of critical time steps and nodes of the path which are useful to analyze the properties of any optimal ride π. To help the intuition, the reader is referred to Figure 2(a). Let $\mathsf{rm}(\pi) = \mathsf{max}_{1 \leq i \leq \mathsf{leftIdx}(\pi)} \pi_i$. Note that $\mathsf{rm}(\pi) < \mathsf{right}(\mathcal{R})$ necessarily holds. Let $\mathsf{rmIdx}(\pi)$ be the minimum time step $i \geq \mathsf{leftIdx}(\mathcal{R})$ such that $\pi_i = \mathsf{rm}(\pi)$. Note that that $\mathsf{rmIdx}(\pi)$ is well defined, because $\mathsf{leftIdx}(\pi) < \mathsf{rightIdx}(\pi)$ and, hence, the ride π has to cross the node $\mathsf{rm}(\pi)$ at least once between the time step $\mathsf{leftIdx}(\pi)$ and the time step $\mathsf{rightIdx}(\pi)$. In fact, it actually holds that $\mathsf{rmIdx}(\pi) < \mathsf{rightIdx}(\pi)$, since $\mathsf{rm}(\pi) < \mathsf{right}(\mathcal{R})$. Then, define $\mathsf{rmLastIdx}(\pi)$ as the maximum time step $i \leq \mathsf{rightIdx}(\pi)$ such that $\pi_i = \mathsf{rm}(\pi)$. Note that $\mathsf{rmLastIdx}(\pi)$ coincides with $\mathsf{rmIdx}(\pi)$ if, and only if, there is no time step i such that $\mathsf{rmIdx}(\pi) < i \leq \mathsf{rightIdx}(\pi)$ with $\pi_i = \mathsf{rm}(\pi)$. Again, observe that $\mathsf{rmLastIdx}(\pi) < \mathsf{rightIdx}(\pi)$ holds. Now, define $\widehat{\mathsf{rm}}(\pi) = \mathsf{max}_{\mathsf{rmIdx}(\pi) \leq i \leq \mathsf{rmLastIdx}(\pi)} \pi_i$. Since $\mathsf{rmLastIdx}(\pi) < \mathsf{rightIdx}(\pi)$ and since $\mathsf{rightIdx}(\pi)$ is the minimum time step where the ride reaches the extreme node $\mathsf{right}(\mathcal{R})$, we have that $\widehat{\mathsf{rm}}(\pi) < \mathsf{right}(\mathcal{R})$. Moreover, $\widehat{\mathsf{rm}}(\pi) \geq \mathsf{rm}(\pi)$ clearly holds. Therefore, there is some time step between $\mathsf{rmLastIdx}(\pi)$ and $\mathsf{rightIdx}(\pi)$ where π crosses $\widehat{\mathsf{rm}}(\pi)$. So, we can define $\widehat{\mathsf{rm}}\mathsf{Idx}(\pi)$ as the minimum index $i \geq \mathsf{rmLastIdx}(\pi)$ such that $\pi_i = \widehat{\mathsf{rm}}(\pi)$, by noticing that $\widehat{\mathsf{rm}}\mathsf{Idx}(\pi) < \mathsf{rightIdx}(\pi)$ holds.

A basic transformation preserving optimality is illustrated in the following lemma.

▶ **Lemma 8.** *Assume there is an optimal ride $\pi \in opt(\mathcal{R})$ such that $\mathsf{leftIdx}(\pi) < \mathsf{rightIdx}(\pi)$. Then, the ride $s_0 \mapsto \widehat{\mathsf{rm}}(\pi) \mapsto \mathsf{left}(\mathcal{R}) \mapsto \widehat{\mathsf{rm}}(\pi) \mapsto \pi[\widehat{\mathsf{rm}}\mathsf{Idx}(\pi), len(\pi)]$ is also optimal.*

Consider now the optimal ride π^*, and the succession of optimal rides π^j, with $j \geq 0$, obtained by repeatedly applying Lemma 8. First, we set $\pi^0 = \pi^*$. Then, for each $j \geq 0$, we define π^{j+1} as the optimal ride having the form: $s_0 \mapsto \widehat{\mathsf{rm}}(\pi^j) \mapsto \mathsf{left}(\mathcal{R}) \mapsto \widehat{\mathsf{rm}}(\pi^j) \mapsto \pi^j[\widehat{\mathsf{rm}}\mathsf{Idx}(\pi^j), len(\pi^j)]$.

In the above succession, there must exists an optimal ride π^h, with $h \geq 0$, such that $\widehat{\mathsf{rm}}(\pi^h) = \mathsf{rm}(\pi^h)$. Indeed, note that $\mathsf{rm}(\pi^{j+1}) = \widehat{\mathsf{rm}}(\pi^j)$ holds, for each $j \geq 0$, and we know that, for any optimal ride π, $\mathsf{rm}(\pi) \leq \widehat{\mathsf{rm}}(\pi) < \mathsf{right}(\mathcal{R})$. For this optimal ride π^h, we have that $\mathsf{rmLastIdx}(\pi^h) = \widehat{\mathsf{rm}}\mathsf{Idx}(\pi^h)$, by definition of these two time steps. Therefore, $\pi^{h+1} = s_0 \mapsto \mathsf{rm}(\pi^h) \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{rm}(\pi^h) \mapsto \pi^h[\mathsf{rmLastIdx}(\pi^h), len(\pi^h)]$. It is possible to prove that π^{h+1} satisfies a number of desirable properties, which can informally be summarised as follows: after time step $\widehat{\mathsf{rm}}\mathsf{Idx}(\pi^h)$ and before reaching $\mathsf{right}(\mathcal{R})$, π^{h+1} never crosses $\widehat{\mathsf{rm}}(\pi^h)$ anymore; moreover, let $\mathsf{lm}(\pi^{h+1}) = \mathsf{min}_{\mathsf{rightIdx}(\pi^{h+1}) \leq i \leq len(\pi^{h+1})} \pi_i$, there is no request $(s,t) \in \mathcal{C}$ such that $t < \mathsf{lm}(\pi^{h+1})$, $t < \widehat{\mathsf{rm}}(\pi^h)$, and $\widehat{\mathsf{rm}}(\pi^h) < s$. For $\mathsf{lm}(\pi^{h+1}) \geq \mathsf{rm}(\pi^h)$, π^{h+1} is depicted in Figure 2(b). Note that, under such condition, π^{h+1} has an addition critical node $\mathsf{cr}(\pi^{h+1})$. Informally speaking, $\widehat{\mathsf{rm}}(\pi^h)$ is the largest node v, such that it is smaller than $\mathsf{lm}(\pi^{h+1})$ and does not admit any crossing request, i.e., a request (s,t) with $t < v \leq s$.

The properties of π^{h+1} allow us to apply a final improving transformation. In particular, we are able to show that, if $\mathsf{lm}(\pi^{h+1}) < \mathsf{rm}(\pi^h)$ then, by setting $M = \mathsf{rm}(\pi^h)$ and $m = \mathsf{lm}(\pi^{h+1})$, π^{h+1} can be reduced to the (M,m)-canonical ride depicted in Figure 2(b); otherwise, by setting $M = \mathsf{rm}(\pi^h)$ and $m = \mathsf{cr}(\pi^{h+1})$, π^{h+1} can be reduced to any of the (M,m)-canonical ride depicted in Figure 2(a).

3.2.2 An algorithm for the "inner" case

It is not difficult to see that the result in Theorem 7 immediately provides us with an algorithm to compute an optimal ride, which is based on exhaustively enumerating all possible pairs M, m of elements, by computing the associated canonical ride for each of them (either by exploiting Fact 6 if $m \leq M$, or using the RIDEONPATH_OUTER algorithm on $\mathcal{R}(M,m)$ of $m > M$), and by eventually returning the feasible one having minimum cost. Actually, in order to deal with the case where all optimal rides π^* are such that $\mathsf{leftIdx}(\pi^*) > \mathsf{rightIdx}(\pi^*)$, we can just apply the approach over the symmetric scenario $\mathsf{sym}(\mathcal{R})$ too (see Fact 3), and return the best over the rides computed for $\mathcal{R}$ and $\mathsf{sym}(\mathcal{R})$.

Note that the approach sketched above requires the enumeration of $|V_\mathcal{C}|^2$ canonical rides. However, we can do better than a naïve enumeration. To this end, we explore the properties enjoyed by canonical rides that are optimal applying to the cases where $M < m$ and $M \geq m$, respectively, hold in Theorem 7.

▶ **Theorem 9.** *Assume that there are two nodes $M, m \in V_\mathcal{C} \cup \{s_0, t_0\}$, with $s_0 \leq M$, $m \leq t_0$ and $M < m$, such that a (M,m)-canonical ride is an optimal ride. Consider the two sets $\hat{X} = \{x \in \{s_0\} \cup V_\mathcal{C} \mid x \geq s_0 \ \wedge \nexists (s,t) \in \mathcal{C}$ with $t \leq x < s\}$ and $\hat{Y} = \{y \in \{t_0\} \cup V_\mathcal{C} \mid y \leq t_0 \ \wedge \nexists (s,t) \in \mathcal{C}$ with $t < y \leq s\}$. It holds that $\hat{X} \neq \emptyset$ and $\hat{Y} \neq \emptyset$. Moreover, let $\hat{M} = \mathsf{min}_{\hat{x} \in \hat{X}} \hat{x}$ and $\hat{m} = \mathsf{max}_{\hat{y} \in \hat{Y}} \hat{y}$, then $s_0 \leq \hat{M}$, $\hat{m} \leq t_0$, $\hat{M} < \hat{m}$ and any $(\hat{M}, \hat{m})$-canonical ride is an optimal ride, too.*

▶ **Theorem 10.** *Assume that there are two nodes $M, m \in V_\mathcal{C} \cup \{s_0, t_0\}$, with $s_0 \leq M$, $m \leq t_0$ and $m \leq M$, such that the (M,m)-canonical ride π^c is an optimal ride. Consider the set $\hat{Z}_m = \{z \in \{s_0, t_0\} \cup V_\mathcal{C} \mid m \leq z$ and $s_0 \leq z \ \wedge \nexists (s,t) \in \mathcal{C}$ with $t < m$ and $z < s\}$. It holds that $\hat{Z} \neq \emptyset$. Moreover, let $\hat{M}_m = \mathsf{min}_{\hat{z} \in \hat{Z}_m} \hat{z}$, then $s_0 \leq \hat{M}$, $m \leq \hat{M}_m$ and the $(\hat{M}_m, m)$-canonical ride $\hat{\pi}^\mathsf{c}$ is optimal, too.*

```
Algorithm 3: RideOnPath_Inner
Input: A ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C}\rangle$, where $G$ is a path and with
       $\{s_0, t_0\} \cap \{v \in V \mid \mathsf{left}(\mathcal{R}) < v < \mathsf{right}(\mathcal{R})\} \neq \emptyset$;
       Optionally, a Boolean value symmetric—set to false, if not provided;
Output: An optimal ride for $\mathcal{R}$;
    /* PHASE I: implementation of Theorem 9                                    */
 1  Compute $\hat{M}$ and $\hat{m}$, as defined in Theorem 9;   // note that $\hat{M} < \hat{m}$
 2  $\pi^* \leftarrow$ any $(\hat{M}, \hat{m})$-canonical ride;   // use RideOnPath_Outer as a subroutine for
    $\mathcal{R}(\hat{M}, \hat{m})$
    /* PHASE II: implementation of Theorem 10                                  */
 3  for each node $m \in V_\mathcal{C} \cup \{s_0, t_0\}$ with $m \leq t_0$ do
 4      Compute $\hat{M}_m$, as defined in Theorem 10;   // note that $\hat{M}_m \geq \hat{m}$
 5      $\pi \leftarrow$ the $(\hat{M}_m, m)$-canonical ride;   // $s_0 \mapsto \hat{M}_m \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$
 6      if $w(\pi) < w(\pi^*)$ then
 7          $\pi^* \leftarrow \pi$;
    /* PHASE III: working on the symmetric scenario                            */
 8  if symmetric is false then
 9      $\pi^*_{\mathsf{sym}} \leftarrow$ RideOnPath_Inner($\mathsf{sym}(\mathcal{R})$, true);
10      if $w(\pi^*_{\mathsf{sym}}) < w(\pi^*)$ then
11          $\pi^* \leftarrow \mathsf{sym}(\pi^*_{\mathsf{sym}})$;
12  return $\pi^*$;
```

In the light of Theorem 7, Theorem 9 and Theorem 10, consider then Algorithm 3, named RideOnPath_Inner. It computes an optimal ride π^* for the "inner" case, by proceeding in three phases.

In Phase I, the algorithm computes the values $\hat{M}$ and $\hat{m}$ defined in Theorem 9 (step 1), it builds a $(\hat{M}, \hat{m})$-canonical ride, and it assigns it to π^* (step 2). Note that, according to Definition 5 and given that $\hat{M} < \hat{m}$, in order to build a $(\hat{M}, \hat{m})$-canonical ride we need to compute an optimal ride for $\mathcal{R}(\hat{M}, \hat{m})$, which is a task that we can accomplish by exploiting RideOnPath_Outer as a subroutine—indeed, note that $\mathcal{R}(\hat{M}, \hat{m})$ fits the "outer" case. In Phase II, the algorithm iterates over all possible values for m in $V_\mathcal{C} \cup \{s_0, t_0\}$ with $m \leq t_0$. For each node m, the value $\hat{M}_m$, defined in Theorem 10, is calculated (step 4). Then, the $(\hat{M}_m, m)$-canonical ride π is built. In particular, since $\hat{M}_m \geq m$ holds, the ride π is completely determined by Fact 6. Eventually, if the cost of π is smaller than the cost of the current value of π^*, it updates π^* to π (step 7). Finally, Phase III is devoted to deal with the symmetric scenario $\mathsf{sym}(\mathcal{R})$. The idea is that the first two phases are executed again on $\mathsf{sym}(\mathcal{R})$. Let π^*_{sym} be the result of this computation (step 9). Then, we consider the symmetric ride $\mathsf{sym}(\pi^*_{\mathsf{sym}})$, which is a ride for $\mathcal{R}$, and we compare its cost with the cost of the current value of π^* (step 10). As usual, we keep the ride with the associated minimum cost, which is eventually returned as output (step 12).

Concerning the correctness, note that if $\mathcal{R}$ admits an optimal ride π with $\mathsf{leftIdx}(\pi) < \mathsf{rightIdx}(\pi)$, then by combining Theorem 7 with Theorem 9 and Theorem 10, we get that either any $(\hat{M}, \hat{m})$-canonical ride is optimal, or there is a node $m \in V_\mathcal{C} \cup \{s_0, t_0\}$ for which the $(\hat{M}_m, m)$-canonical ride is optimal. Instead, if every optimal ride π for $\mathcal{R}$ is such that $\mathsf{leftIdx}(\pi) > \mathsf{rightIdx}(\pi)$, then $\mathsf{sym}(\mathcal{R})$ admits an optimal ride that meets the fits the previous case. We can conclude that an optimal ride for $\mathcal{R}$ is one with the smallest cost among any $(\hat{M}, \hat{m})$-canonical ride and every $(\hat{M}_m, m)$-canonical ride, for every value of m in $V_\mathcal{C} \cup \{s_0, t_0\}$, both for $\mathcal{R}$ and for $\mathsf{sym}(\mathcal{R})$. Note that RideOnPath_Inner exhaustively searches among all the possible candidate optimal rides listed above. So, the algorithm is correct.

3.3 Implementation issues and running time

Note that checking whether an instance fits the "outer" or the "inner" case is feasible in $O(|\mathcal{C}|)$. Our goal is to show that both RideOnPath_Outer and RideOnPath_Inner can be made to run in $O(|\mathcal{C}|\log|\mathcal{C}|+|V|)$. So, we eventually establish the following result.

▶ **Theorem 11.** *Let $\mathcal{R}=\langle G,(s_0,t_0),\mathcal{C}\rangle$ be a ride-sharing scenario where $G=(V,E,w)$ is a path. Then, an optimal ride for $\mathcal{R}$ (together with its cost) can be computed in time $O(|\mathcal{C}|\log|\mathcal{C}|+|V|)$.*

In the implementation, we propose to sort these requests in order of starting node and, accordingly, we shall assume that $\hat{\mathcal{C}}=\{(s_1,t_1),(s_2,t_2),\dots,(s_{|\hat{\mathcal{C}}|},t_{|\hat{\mathcal{C}}|})\}$ holds with $s_i\le s_j$ whenever $i<j$. Similarly, we sort the nodes in $V_{\mathcal{C}}\cup\{s_0,t_0\}$, and hence we assume that $V_{\mathcal{C}}\cup\{s_0,t_0\}=\{w_1,w_2,\dots,w_r\}$ holds with $w_i\le w_j$ whenever $i<j$. Moreover, for each node $w_i\in V_{\mathcal{C}}\cup\{s_0,t_0\}$, we define the set $F(w_i)=\{j\mid(s_j,t_j)\in\hat{\mathcal{C}}\wedge(w_i=s_j \text{ or } w_i=t_j)\}$, maintained as linked list. And, finally, for each element j in $F(w_i)$ we keep a label $l_{ij}\in\{\mathsf{s},\mathsf{t}\}$ denoting whether w_i is a starting (s) or a terminating (t) node of request j. This is feasible in $O(|\mathcal{C}|\log|\mathcal{C}|)$. Given the pre-processing, it is not difficult to show that RideOnPath_Outer can be implemented in $O(|\mathcal{C}|\log|\mathcal{C}|+|V|)$, where the extra $O(|V|)$ factor comes from the need of explicitly building the ride and computing the associated cost.

Let us then analyze RideOnPath_Inner and let us focus on Phase I and Phase II, since it is immediate to check that Phase III has the same complexity.

Concerning Phase I, we first need to compute $\hat{M}$ and $\hat{m}$. To this end, we iterate through the nodes in $V_{\mathcal{C}}\cup\{s_0,t_0\}$ in order of increasing index, starting from w_1. Throughout the iteration, we maintain a set S of indexes of requests in $\hat{\mathcal{C}}$. Initially $S=\emptyset$; during the k-th iteration, we add to S every $j\in F(w_k)$ with $l_{kj}=\mathsf{t}$, and we remove from S every $j\in F(w_k)$ with $l_{kj}=\mathsf{s}$. Note that, at the end of the iteration, S contains all the elements in P_{w_k}, so that if $w_k\ge s_0$ and $S=\emptyset$, then we terminate by concluding that w_k is the smallest element in $\hat{X}$. Given the existence of $\hat{M}$, such procedure always terminates. For the complexity analysis, observe that every request in $\hat{\mathcal{C}}$ is added and removed from S exactly once. Hence, the time taken by the procedure is at most $O(|\hat{\mathcal{C}}|)$ times the maximum cost for performing each operation. If the set S is maintained as a binary min-heap, where the key of each request is its starting node, removing an element from S with label s corresponds to extract the element with smallest key, and both the insertion and the removal can be made to run in time $O(\log|\hat{\mathcal{C}}|)$. A similar approach can be used to compute $\hat{m}$. Thus, Phase I takes total time $O(|\hat{\mathcal{C}}|\log|\hat{\mathcal{C}}|)$, hence $O(|\mathcal{C}|\log|\mathcal{C}|)$, to define the pair $\hat{M},\hat{m}$. A canonical ride with its associated cost can be then computed in $O(|\mathcal{C}|\log|\mathcal{C}|+|V|)$, since the dominant operation is the invocation of the algorithm for the outer case.

Concerning Phase II, let $m\in V_{\mathcal{C}}\cup\{s_0,t_0\}$ with $m\le t_0$, and let $\hat{M}_m$ be the node as defined in Theorem 10. Consider the set $Q_m=\{(s',t')\in\mathcal{C}\mid t'<m<s'\}$, and let $u_m=\mathsf{max}\{m,\ s_0\}$ if $Q_m=\emptyset$, and $u_m=\mathsf{max}\{s_0,\ \mathsf{max}_{(s',t')\in Q_m}s'\}$ otherwise. Then, we can show that $\hat{M}_m=u_m$.

Therefore, for every node $w_i\in V_{\mathcal{C}}\cup\{s_0,t_0\}$, $\hat{M}_{w_i}$ is defined as the maximum between w_i and s_0, if Q_{w_i} is not empty, or the maximum between s_0 and $\mathsf{max}_{(s',t')\in Q_{w_i}}s'$, otherwise. So, the dominant operation is the computation of Q_{w_i}. To this end, for every $w_i\in V_{\mathcal{C}}\cup\{s_0,t_0\}$, we iterate through the nodes in $V_{\mathcal{C}}\cup\{s_0,t_0\}$ in order of increasing index. Note that $Q_{w_i}\subseteq\hat{\mathcal{C}}$, hence equivalently we can write $Q_{w_i}=\{(s',t')\in\hat{\mathcal{C}}\mid t'<w_i<s'\}$; this implies that, in order to compute Q_{w_i}, we need of only the requests in $\hat{\mathcal{C}}$ and we can use the usual data structures. More specifically, we iterate through the nodes in $V_{\mathcal{C}}\cup\{s_0,t_0\}$ in order of increasing index, starting from w_1. Initially, we define a set $S=\emptyset$. During the k-th iteration, we remove from S every $j\in F(w_k)$ with $l_{kj}=\mathsf{s}$, and if $k\ge2$ we add to S every $j\in F(w_{k-1})$ with

$l_{(k-1)j} = \mathsf{t}$. Note that, at the end of the iteration, S contains all the elements in Q_{w_k}. Thus, if $S = \emptyset$, then we set M_{w_k} to $\max\{m, s_0\}$, otherwise we set M_{w_k} to $\max\{s_0, \max_{(s',t')\in S} s'\}$. In the latter case, we need to calculate $\max_{(s',t')\in S} s'$, i.e., to search in S for the request with the largest starting node. We continue in this fashion until we run out of nodes. For the complexity analysis, observe that every request in $\hat{\mathcal{C}}$ is added and removed from S exactly once. Moreover, at the end of each iteration, we need to search in S for the request with the largest starting node, in order to calculate $\max_{(s',t')\in S} s'$. Hence, the time taken by the procedure is at most $O(|\hat{\mathcal{C}}|)$ times the maximum cost for performing each operation. If the set S is maintained as a binary min-max-heap, where the key of each request is its starting node, removing an element from S with label s corresponds to extract the element with smallest key, hence both the insertion and the removal can be made to run in time $O(\log |\hat{\mathcal{C}}|)$; moreover, calculating $\max_{(s',t')\in S} s'$ corresponds to search for the element with largest key, which takes only constant time. Thus, the computation of $\hat{M}_{w_i}$, for every node $w_i \in V_{\mathcal{C}} \cup \{s_0, t_0\}$, takes a total time $O(|\hat{\mathcal{C}}| \log |\hat{\mathcal{C}}|)$, hence $O(|\mathcal{C}| \log |\mathcal{C}|)$.

Now, note that the computation of the $(\hat{M}_m, m)$-canonical ride takes constant time, since by Fact 6, we know that this ride has the form $s_0 \mapsto \hat{M}_m \mapsto \mathsf{left}(\mathcal{R}) \mapsto \mathsf{right}(\mathcal{R}) \mapsto m \mapsto t_0$. Then, the remaining operation in Phase II is the comparison between the cost of the given best ride and cost of the current ride. We have already seen that the computation of the cost of rides built in Phase I can be accommodated in the overall $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$ cost. Now, we claim that the computation of the cost of the $(\hat{M}_m, m)$-canonical ride takes constant time, provided a suitable pre-processing. Indeed, observe that the $(\hat{M}_m, m)$-canonical ride is succinctly represented by a constant number of nodes. The idea is then to associate each node $x \in V$ with the value $cw(x) = \sum_{i=2}^{x} w(\{i, i+1\})$, which is overall feasible in $O(|V|)$. Then, the cost for a rides moving from a node x to a node y, along the unique path as defined in the notion of canonical ride, is just given by the value $|cw(y) - cw(x)|$. Therefore, with a constant overhead, the cost of the $(\hat{M}_m, m)$-canonical ride can be computed. Putting it all together, Phase II can be implemented in $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$, too.

4 Optimal Rides on Cycles

In this section, we consider scenarios $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C}\rangle$ such that the underlying graph $G = (V, E, w)$, with $V = \{1, \ldots, n\}$, is a *cycle*. Formally, for each node $v \in V \setminus \{n\}$, the edge $\{v, v+1\}$ is in E; moreover, the edge $\{n, 1\}$ is in E; and no further edge is in E. Without loss of generality, we assume $s_0 = 1$.

The solution approach we shall propose is to reuse the methods we have already developed to deal with scenarios over paths. In this section, we define the key technical ingredients, and based on them an algorithm will be subsequently illustrated. Let π be a ride on $\mathcal{R}$, and let us associate each of its time steps i with a "virtual" node $\tau_\pi(i) = \pi_i + (\ell_\pi(i) - \mathsf{min}_{j\in\{1,\ldots,len(\pi)\}}\ell_\pi(j)) \cdot n$, where $\ell_\pi(1) = 0$ and where, for each $i \in \{2, \ldots, len(\pi)\}$, $\ell_\pi(i)$ is an integer defined as follows: $\ell_\pi(i) = \ell_\pi(i-1)+1$ if $\pi_{i-1} = n$ and $\pi_i = 1$; $\ell_\pi(i) = \ell_\pi(i-1)-1$ if $\pi_{i-1} = 1$ and $\pi_i = n$; and $\ell_\pi(i) = \ell_\pi(i-1)$ otherwise.

Intuitively, the function τ_π keeps track of the number of times in which the cycle is completely traversed by the ride, either clockwise or anti clockwise. Note that $\tau_\pi(i) \texttt{ mod } n = \pi_i$.

Let $\mathsf{cw}(\pi)$ (resp., $\mathsf{acw}(\pi)$) be the maximum (resp., minimum) value of $\tau_\pi(i)$ over all time steps $i \in \{1, \ldots, len(\pi)\}$. Let $\mathsf{cwIdx}(\pi)$ (resp., $\mathsf{acwIdx}(\pi)$) be the minimum time step $i \in \{1, \ldots, len(\pi)\}$ such that $\tau_\pi(i) = \mathsf{acw}(\pi)$ (resp., $\tau_\pi(i) = \mathsf{cw}(\pi)$). Note that $1 \leq \mathsf{acw}(\pi) \leq n$ always hold, by definition of τ_π. In fact, over optimal rides, useful characterizations and bounds can be derived for both $\mathsf{acw}(\pi)$ and $\mathsf{cw}(\pi)$.

Algorithm 4: RideOnCycle

```
Input: A ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$, where $G$ is a cycle;
Output: An optimal ride for $\mathcal{R}$ ;
1 for each tuple $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ of elements as in Theorem 14 do
2     Let $\pi^\circ$ be an optimal ride for $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$;
3     if $\pi^*$ is not yet defined or $w^\circ(\pi^\circ) < w^\circ(\pi^*)$ then
4         $\pi^* \leftarrow \pi^\circ$;
5 return $\pi^*_1$ mod $n, \ldots, \pi^*_{len(\pi^*)}$ mod $n$;
```

▶ **Lemma 12.** *An optimal ride π exists with $\mathsf{cw}(\pi) \leq 3n$ and $\{\mathsf{cw}(\pi) \texttt{ mod } n, \mathsf{acw}(\pi) \texttt{ mod } n\} \subseteq V_\mathcal{C} \cup \{s_0, t_0\}$.*

Now, consider the path $G^\circ = (V^\circ, E^\circ, w^\circ)$, where $V^\circ = \{1, \ldots, 3n\}$ and where w° is the function such that $w^\circ(\{v, v+1\}) = w(\{v \texttt{ mod } n, (v+1)\texttt{mod } n\})$. For each pair of nodes $\alpha, \beta \in V^\circ$ with $\alpha \leq \beta$, let us define $V^\circ_{\alpha,\beta}$ as the set of nodes $v \in \{\alpha, \ldots, \beta\}$ for which no other distinct node $v' \in \{\alpha, \ldots, \beta\}$ exists such that $v \texttt{ mod } n = v' \texttt{ mod } n$. Note that if $\beta < \alpha + n$, then $V^\circ_{\alpha,\beta} = \{\alpha, \ldots, \beta\}$; if $\beta \geq \alpha + 2n - 1$, then $V^\circ_{\alpha,\beta} = \emptyset$; if $\alpha + n \leq \beta < \alpha + 2n - 1$, then $V^\circ_{\alpha,\beta} = \{\beta - n + 1, \ldots, \alpha + n - 1\}$. Moreover, define $\mathcal{C}^\circ_{\alpha,\beta} = \{(v_s, v_t) \mid (v_s \texttt{ mod } n, v_t \texttt{ mod } n) \in \mathcal{C}, v_s \in V^\circ_{\alpha,\beta}, v_t \in V^\circ_{\alpha,\beta}\}$.

▶ **Theorem 13.** *Let π be a feasible ride for $\mathcal{R}$ with $\mathsf{cw}(\pi) \leq 3n$ and such that $\mathsf{acwIdx}(\pi) \leq \mathsf{cwIdx}(\pi)$ (resp., $\mathsf{acwIdx}(\pi) > \mathsf{cwIdx}(\pi)$). Let $\alpha = \mathsf{acw}(\pi)$ and $\beta = \mathsf{cw}(\pi)$, and let $(s^\circ, t^\circ) = (\alpha, \beta)$ (resp., $(s^\circ, t^\circ) = (\beta, \alpha)$). Then, the ride $\tau_\pi(1), ..., \tau_\pi(len(\pi))$ is feasible for $\langle G^\circ, (\tau_\pi(1), \tau_\pi(len(\pi))), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$.*

Intuitively, the result tells us that feasible rides for $\mathcal{R}$ are mapped into feasible rides for a suitable defined scenario over a path. Below, we show that the converse also holds, under certain technical conditions.

▶ **Theorem 14.** *Assume that: **(i)** $\alpha, \beta \in V^\circ$ is a pair of nodes such that $\{\alpha \texttt{ mod } n, \beta \texttt{ mod } n\} \subseteq V_\mathcal{C} \cup \{s_0, t_0\}$, $1 \leq \alpha, \beta \leq 3n$, and such that, for each $x \in V_\mathcal{C} \cup \{s_0, t_0\}$, there is a node $v_x \in V^\circ$ with $\alpha \leq v_x \leq \beta$ and $x = v_x \texttt{ mod } n$. **(ii)** $v_{s_0}, v_{t_0} \in V^\circ$ is a pair of nodes such that $\alpha \leq v_{s_0} \leq \beta$, $\alpha \leq v_{t_0} \leq \beta$, $v_{s_0} \texttt{ mod } n = s_0$, and $v_{t_0} \texttt{ mod } n = t_0$. **(iii)** (s°, t°) is a request such that $(s^\circ, t^\circ) \in \{(\alpha, \beta), (\beta, \alpha)\}$. Let π° be a feasible ride for $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$. Then, $\pi^\circ_1 \texttt{ mod } n, \ldots, \pi^\circ_{len(\pi^\circ)} \texttt{ mod } n$ is a feasible ride for $\mathcal{R}$.*

Armed with the above technical ingredients, we can now illustrate Algorithm 4, which we refer to as RideOnCycle. This algorithm computes an optimal ride for any ride-sharing scenario $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$, with G being a cycle. The algorithm founds on the idea of enumerating each possible tuple $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ of elements as in Theorem 14. For each given configuration, the optimal ride π° over the scenario $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\} \rangle$ is computed. Eventually, π^* is defined (see step 3) as the ride with minimum cost (w.r.t. w°) over such rides π°. The ride $\pi^*_1 \texttt{ mod } n, \ldots, \pi^*_{len(\pi^*)} \texttt{ mod } n$ is then returned. Now, we claim the following.

▶ **Theorem 15.** *Let $\mathcal{R} = \langle G, (s_0, t_0), \mathcal{C} \rangle$ be a ride-sharing scenario where $G = (V, E, w)$ is a cycle. Then, an optimal ride for $\mathcal{R}$ can be computed in time $O(|V_\mathcal{C}|^2 \cdot (|\mathcal{C}| \log |\mathcal{C}| + |V|))$.*

In order to analyze the correctness, observe that by Theorem 14, the ride returned as output, say $\Lambda^* = \pi^*_1 \texttt{ mod } n, \ldots, \pi^*_{len(\pi^*)} \texttt{ mod } n$, is necessarily feasible for $\mathcal{R}$. Therefore, assume for the sake of contradiction that there is an optimal ride π for $\mathcal{R}$ such that $w(\pi) < w(\Lambda^*)$. In particular, by construction of w°, we derive that $w(\pi) < w(\Lambda^*) = w^\circ(\pi^*)$. Now, by Lemma 12, we can actually assume, w.l.o.g., that $\mathtt{cw}(\pi) \leq 3n$ and $\{\mathtt{cw}(\pi) \bmod n, \mathtt{acw}(\pi) \bmod n\} \subseteq V_\mathcal{C} \cup \{s_0, t_0\}$ hold. So, we can apply Theorem 13 and derive the existence of a tuple $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ of elements, with $v_{s_0} = \tau_\pi(1)$ and $v_{t_0} = \tau_\pi(len(\pi))$, satisfying properties *(i)*, *(ii)*, and *(iii)* in Theorem 14 and such that $\Upsilon = \tau_\pi(1), ..., \tau_\pi(len(\pi))$ is feasible for $\langle G^\circ, (v_{s_0}, v_{t_0}), \mathcal{C}^\circ_{\alpha,\beta} \cup \{(s^\circ, t^\circ)\}\rangle$. In particular, by construction of w°, we derive that $w^\circ(\Upsilon) = w(\pi)$. However, the algorithm has compared the weight of Υ and π^*, and hence we know that $w(\pi) = w^\circ(\Upsilon) \geq w^\circ(\pi^*)$, which is impossible.

Let us finally discuss about the implementation and running time of the algorithm. Before starting the loop, we first compute the sets $W = \{w \in V^\circ \mid 1 \leq w \leq 3n \text{ and } (w \bmod n) \in V_\mathcal{C} \cup \{s_0, t_0\}\}$ and $\mathcal{C}^\circ = \{(s,t) \in W \mid (s \bmod n, t \bmod n) \in \mathcal{C}\}$; this can be done in time $O(|\mathcal{C}|)$ by iterating through the requests in $\mathcal{C}$. Note that $|W| = O(|V_\mathcal{C}|)$ and $\mathcal{C}^\circ| = O(|\mathcal{C}|)$. Now, note that the number of iterations of RideOnCycle corresponds to the number tuples $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$ which satisfy the conditions of Theorem 14. The number of possible pairs (α, β) is $W^2 = O(|V_\mathcal{C}|^2)$. Checking whether condition *(i)* in Theorem 14 holds on them can be simply accomplished by checking that every element $x \in V_\mathcal{C} \cup \{s_0, t_0\}\}$ is such that $\alpha \bmod n \leq x \leq \beta \bmod n$. So, it can be done in constant time after that, in a pre-processing step costing $O(|V_C|)$, the minimum and maximum element in $V_\mathcal{C} \cup \{s_0, t_0\}\}$ have been computed. Moreover, note that since $1 \leq \alpha, \beta \leq 3n$, according to Theorem 14, there are at most 3 possible choices for s_0 (resp, t_0); in addition, there are just two alternatives for the pair s°, t°. Hence, summarizing we have that all tuples satisfying the conditions of Theorem 14 can be enumerated in $O(|V_\mathcal{C}|^2)$. Then, by inspecting the operations performed at each iteration, for each tuple $\langle \alpha, \beta, v_{s_0}, v_{t_0}, s^\circ, t^\circ \rangle$, we have to compute the set $\mathcal{C}^\circ_{\alpha,\beta}$. To this end, we search among the elements in $\mathcal{C}^\circ$ for the pairs (s,t) having both nodes in $V^\circ_{\alpha,\beta}$; this step takes $O(|\mathcal{C}|)$. Finally, on the resulting scenario defined on a path, we apply the algorithm for computing an optimal ride, which costs $O(|\mathcal{C}| \log |\mathcal{C}| + |V|)$. Hence the result in the theorem follows.

References

1 T. Asano, N. Katoh, H. Tamaki, and T. Tokuyama. Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k. In *Proc. of STOC*, pages 275–283, 1997.

2 M.J. Atallah and S.R. Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17(5):849–869, 1988.

3 P. Chalasani and R. Motwani. Approximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28(6):2133–2149, 1999.

4 P. Chalasani, R. Motwani, and A. Rao. Algorithms for robot grasp and delivery. In *2nd Int. Workshop on Algorithmic Foundations of Robotics*, 1996.

5 M. Charikar and B. Raghavachari. The finite capacity dial-a-ride problem. In *Proc. of FOCS*, pages 458–467, 1998.

6 M. Charikar, S. Khuller, and B. Raghavachari. Algorithms for capacitated vehicle routing. *SIAM Journal on Computing*, 31(3):665–682, 2001.

7 J.-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

8 G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

9 Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

10 B. Eksioglu, A.V. Vural, and A. Reisman. Survey: The vehicle routing problem: A taxonomic review. *Computers and Industrial Engineering*, 57(4):1472–1483, 2009.

11 J.-F. Cordeau, G. Laporte, J.Y. Potvin, and M.W.P. Savelsbergh. Transportation on demand. In *Handbooks in operations research and management*, 2007.

12 G.N. Frederickson and D.J. Guan. Nonpreemptive ensemble motion planning on a tree. *Journal of Algorithms*, 15(1):29–60, 1993.

13 G.N. Frederickson. A note on the complexity of a simple transportation problem. *SIAM Journal on Computing*, 22(1):57–61, 1993.

14 G.N. Frederickson and D.J. Guan. Preemptive ensemble motion planning on a tree. *SIAM Journal on Computing*, 21(6):1130–1152, 1992.

15 G.N. Frederickson, M.S. Hecht, and C.E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7(2):178–193, 1978.

16 M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979.

17 M. Gendreau, G. Laporte, and D. Vigo. Heuristics for the traveling salesman problem with pickup and delivery. *Computers and Operations Research*, 26(7):699–714, 1999.

18 D.J. Guan. Routing a vehicle of capacity greater than one. *Discrete Applied Mathematics*, 81(1-3):41–57, 1998.

19 D.J. Guan and X. Zhu. Multiple capacity vehicle routing on paths. *SIAM Journal on Discrete Mathematics*, 11(4):590–602, 1998.

20 M. Haimovich and A.H.G. Rinnooy Kan. Bounds and heuristics for capacitated routing problems. *Mathematics of Operations Research*, 10(4):527–542, 1985.

21 H. Hernandez-Perez and J.J. Salazar-Gonzalez. The one-commodity pickup-and-delivery travelling salesman problem. In *Combinatorial Optimization*, pages 89–104, 2003.

22 B. Kalantari, A.V. Hill, and S.R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386, 1985.

23 R.M. Karp. Two combinatorial problems associated with external sorting. *Combinatorial Algorithms, Courant Computer Science Symposium 9*, pages 17–29, 1972.

24 G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

25 G. Mosheiov. Vehicle routing with pick-up and delivery: tour-partitioning heuristics. *Computers and Industrial Engineering*, 34(3):669–684, 1998.

26 J. Park and B.-I. Kim. The school bus routing problem: A review. *European Journal of Operational Research*, 202(2):311–319, 2010.

27 SophieN. Parragh, KarlF. Doerner, and RichardF. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(1):21–51, 2008.

28 F. J. Srour and S. van de Velde. Are stacker crane problems easy? a statistical study. *Computers and Operations Research*, 40(3):674–690, 2013.

29 P. Toth and D. Vigo. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, 2002.

30 T.E. Tzoreff, D. Granot, F. Granot, and G. Sošić. The vehicle routing problem with pickups and deliveries on some special graphs. *Discrete Applied Mathemathics*, 116(3):193–229, 2002.

On the General Chain Pair Simplification Problem

Chenglin Fan[1], Omrit Filtser*[2], Matthew J. Katz[3], and Binhai Zhu[4]

1 **Montana State University**
Bozeman, MT 59717-3880, USA
bhz@montana.edu

2 **Ben-Gurion University of the Negev**
Beer-Sheva 84105, Israel
omritna@cs.bgu.ac.il

3 **Ben-Gurion University of the Negev**
Beer-Sheva 84105, Israel
matya@cs.bgu.ac.il

4 **Montana State University**
Bozeman, MT 59717-3880, USA
bhz@montana.edu

Abstract

The Chain Pair Simplification problem (CPS) was posed by Bereg et al. who were motivated by the problem of efficiently computing and visualizing the structural resemblance between a pair of protein backbones. In this problem, given two polygonal chains of lengths n and m, the goal is to simplify both of them simultaneously, so that the lengths of the resulting simplifications as well as the discrete Fréchet distance between them are bounded. When the vertices of the simplifications are arbitrary (i.e., not necessarily from the original chains), the problem is called General CPS (GCPS).

In this paper we consider for the first time the complexity of GCPS under both the discrete Fréchet distance (GCPS-3F) and the Hausdorff distance (GCPS-2H). (In the former version, the quality of the two simplifications is measured by the discrete Fréchet distance, and in the latter version it is measured by the Hausdorff distance.) We prove that GCPS-3F is polynomially solvable, by presenting an $\widetilde{O}((n+m)^6 \min\{n, m\})$ time algorithm for the corresponding minimization problem. We also present an $O((n+m)^4)$ 2-approximation algorithm for the problem. On the other hand, we show that GCPS-2H is **NP**-complete, and present an approximation algorithm for the problem.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Chain simplification, discrete Fréchet distance, dynamic programming, geometric arrangements, protein structural resemblance

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.37

1 Introduction

Polygonal curves play an important role in many applied areas, such as 3D modeling, map matching, and protein backbone structural alignment and comparison. There exist many methods for comparing curves in these (and in many other) applications, where one of the more prevalent methods is the Fréchet distance.

* Work by O. Filtser was supported by the Ministry of Science, Technology & Space, Israel, and by the Lynn and William Frankel Center.

The *Fréchet distance* between two curves is often described through the man-dog analogy. Imagine a man and a dog connected by a leash, each walking along his own curve from its starting point to its end point. Both of them can control their speed, but they can only move forward. The Fréchet distance between the two curves is the length of a minimum-length leash that allows them to reach the end point of their curves.

In the *discrete Fréchet distance* we are given finite sequences of points instead of continuous curves. The same rules apply, but now the man and the dog are hopping between the points of their sequence. The discrete Fréchet distance is a simpler version, and is considered a good approximation of the continuous distance.

Recently, the discrete Fréchet distance was used to align and compare protein backbones, yielding favorable results in many instances [11, 12]. A protein backbone may consists of as many as 500$\sim$600 α-carbon atoms, which are the vertices (i.e., points) of our chain. Thus, a natural approach to accelerate computations is to use a simplification of the chain. In general, given a chain A of n vertices, a simplification of A is a chain A' such that A' is "close" to A and the number of vertices in A' is significantly smaller than n. The vertices of the simplification A' can be arbitrary, or restricted to the vertices of A (in order).

Simplifying two aligned chains independently does not necessarily preserve the resemblance between them. Thus, the following question arises: Is it possible to simplify both chains in a way that will retain the resemblance between them? This question has led Bereg et al. [3] to pose the Chain Pair Simplification problem (CPS). In this problem, the goal is to simplify both chains simultaneously, so that the discrete Fréchet distance between the resulting simplifications is bounded. More precisely, given two chains A and B of lengths n and m, respectively, an integer k and three real numbers δ_1,δ_2,δ_3, one needs to find two chains A',B' with vertices from A,B, respectively, each of length at most k, such that $d_1(A, A') \le \delta_1$, $d_2(B, B') \le \delta_2$, $d_{dF}(A', B') \le \delta_3$ (d_1 and d_2 can be any similarity measures and d_{dF} is the discrete Fréchet distance).

When the chains are simplified using the Hausdorff distance, i.e., d_1, d_2 is the Hausdorff distance (CPS-2H), the problem becomes **NP**-complete [3]. When the chains are simplified using the Fréchet distance, i.e., d_1, d_2 is the Fréchet distance (CPS-3F), the problem is polynomially solvable, as shown by Fan et al. [9] who presented an $O(m^2n^2 \min\{m, n\})$-time algorithm for the minimization problem of CPS-3F.

In this paper we consider, for the first time, the problem where the vertices of the simplifications A', B' may be arbitrary points, Steiner points, i.e., they are not necessarily from A, B, respectively. Since this problem is more general, we call it General CPS, or GCPS for short. Our main contribution, see below, is a (relatively) efficient polynomial-time algorithm for GCPS, or more precisely, for its corresponding optimization problem. As a first step towards devising such an algorithm, we had to characterize the structure of a solution to the problem. This was quite difficult, since on the one hand, we have full freedom in determining the vertices of the simplifications, but, on the other hand, the definition of the problem induces an implicit dependency between the two simplifications. The second challenge in devising such an algorithm, is to reduce its time complexity (which is unavoidably high), by making some non-trivial observations on the combinatorial complexity of an arrangement of complex objects that arises, and by applying some sophisticated tricks.

Since the time complexity of our algorithm is still rather high, it makes sense to resort to more realistic approximation algorithms. See below for a detailed description of our results in this direction and of the rest of our results.

Related work

The Fréchet distance and its variants have been studied extensively in the past two decades. Given two polygonal curves of lengths m and n, Alt and Godau [2] gave an $O(mn \log mn)$-time algorithm for computing the Fréchet distance between them. This result in the plane was recently improved by Buchin et al [6]. The discrete Fréchet distance was originally defined by Eiter and Mannila [8], who also presented an $O(mn)$-time algorithm for computing it. A slightly sub-quadratic algorithm was given recently by Agarwal et al. [1]. Bringmann [4], and later Bringmann and Mulzer [5], presented a conditional lower bound implying that strongly subquadratic algorithms for the discrete Fréchet distance are unlikely to exist, even in the one-dimensional case and even if the solution may be approximated up to a factor of 1.399.

Bereg et al. [3] were the first to study simplification problems under the discrete Fréchet distance. They considered several versions of the problem, and presented polynomial-time exact algorithms. Driemel and Har-Peled [7] presented an algorithm for finding an approximate simplification in near linear time.

Our results

In Section 3, we show that GCPS-3F is polynomially solvable by presenting a sophisticated polynomial-time algorithm for the corresponding optimization problem. In Section 4 we give an $O(m+n)^4$-time 2-approximation algorithm for the problem. In Section 5 we consider the 1-sided version of the problem and present a simpler and more efficient algorithm for this problem. Finally, in Section 6 we investigate GCPS-2H, showing that it is **NP**-complete and presenting an approximation algorithm for the problem.

2 Preliminaries

There are several equivalent definitions for the discrete Fréchet distance. In this paper, we use the one that is based on the notion of a paired walk, following [10], [3] and [7].

Let $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_m)$ be two sequences of points in $\mathbb{R}^d$. We denote by $d(a, b)$ the distance between two points $a, b \in \mathbb{R}^d$. For $1 \le i \le j \le n$, we denote by $A[i, j]$ the subchain $a_i, a_{i+1}, \dots, a_j$ of A.

A *paired walk* along A and B is a sequence of pairs (or matchings) $W = \{(A_i, B_i)\}_{i=1}^{k}$, such that $A = A_1 \cdot A_2 \cdots A_k$ and $B = B_1 \cdot B_2 \cdots B_k$, and for any i it holds that $|A_i| = 1$ or $|B_i| = 1$ (where $|A_i|, |B_i| \ge 1$). The *cost of a paired walk* W along A and B is $d_{dF}^{W}(A, B) = \max_i \max_{(a,b)\in A_i \times B_i} d(a, b)$.

The *discrete Fréchet distance* between A and B is $d_{dF}(A, B) = \min_W d_{dF}^{W}(A, B)$. A *Fréchet walk* along A and B is a paired walk W along A and B for which $d_{dF}^{W}(A, B) = d_{dF}(A, B)$.

A δ-simplification of A w.r.t. distance d_1, is a sequence of points $A' = (a'_1, \dots, a'_k)$, such that $k \le n$ and $d_1(A, A') \le \delta$. The points of A' can be arbitrary (the *general* case), or a subset of the points in A appearing in the same order as in A, i.e., $A' = (a_{i_1}, \dots, a_{i_k})$ and $i_1 \le \cdots \le i_k$ (the *restricted* case).

The different versions of the chain pair simplification problem are defined as follows.

▶ **Problem 1.**
Instance: Given a pair of polygonal chains A and B of lengths n and m, respectively, an integer k, and three real numbers $\delta_1, \delta_2, \delta_3 > 0$.
Problem: Does there exist a pair of chains A',B', each of at most k vertices, such that A'

is a δ_1-simplification of A w.r.t. d_1 ($d_1(A, A') \leq \delta_1$), B' is a δ_2-simplification of B w.r.t. d_2 ($d_2(B, B') \leq \delta_2$), and $d_{dF}(A', B') \leq \delta_3$?

When the vertices of the simplifications are from A and B (restricted simplifications), the problem is called CPS, and when the vertices of the simplifications are not necessarily from A and B (arbitrary simplifications), we call the problem GCPS. For each problem, we distinguish between two versions:

1. When $d_1 = d_2 = d_H$, the problems are called CPS-2H and GCPS-2H, respectively.
2. When $d_1 = d_2 = d_{dF}$, the problems are called CPS-3F and GCPS-3F, respectively.

▶ Remark. We sometimes say that a set D of disks of radius δ covers a chain C. By this we mean that there exists a partition of C into consecutive subchains $C = C_1 \cdot C_2 \cdots C_t$, such that for each $1 \leq i \leq t$ there exists a disk in D that contains all the points of C_i.

3 GCPS under the Fréchet distance

In order to solve GCPS-3F, we consider the optimization problem: Given a pair of polygonal chains A and B of lengths n and m, respectively, and three real numbers $\delta_1, \delta_2, \delta_3 > 0$, what is the smallest number k such that there exist a pair of chains A',B', each of at most k (arbitrary) vertices, for which $d_{dF}(A, A') \leq \delta_1$, $d_{dF}(B, B') \leq \delta_2$, and $d_{dF}(A', B') \leq \delta_3$?

We begin by describing some properties that are required from an optimal solution to the problem. Then, based on these properties, we are able to refine our search for the optimal solution.

3.1 What does an optimal solution look like?

Let (A', B') be an optimal solution, that is, let A' and B' be two arbitrary simplifications of A and B respectively, such that $d_{dF}(A, A') \leq \delta_1$, $d_{dF}(B, B') \leq \delta_2$, $d_{dF}(A', B') \leq \delta_3$, and $\max\{|A'|, |B'|\}$ is minimum. Moreover, we assume that the shorter of the chains A', B' is as short as possible.

Let $W_{A'B'} = \{(A'_i, B'_i)\}_{i=1}^{t}$ be a Fréchet walk along A' and B'. Notice that, by definition, for any i it holds that $|A'_i| = 1$ or $|B'_i| = 1$.

Let $W_{AA'}$ be a Fréchet walk along A and A'. Notice that unlike in regular (one-sided) simplifications, the pairs in $W_{AA'}$ may match several points from A' to a single point from A, because A' does not depend only on A but also on B' and B. Similarly, let $W_{BB'}$ be a Fréchet walk along B and B' (see Figure 1).

With each pair $(A'_i, B'_i) \in W_{A'B'}$, we associate a pair of subchains A_i of A and B_i of B, which we call a *pair component*. Assume $A'_i = A'[p, q]$, then A_i is defined as follows:

1. If $p \neq q$, then each $a'_k \in A'[p, q]$ appears as a singleton in $W_{AA'}$ (since otherwise A' can be shortened). Let A^k be the subchain of A that is matched to a'_k, i.e., $(A^k, a'_k) \in W_{AA'}$, for $k = p, \ldots, q$. Then, we set $A_i = A^p A^{p+1} \cdots A^q$.
2. If $p = q$ and a'_p appears as a singleton in $W_{AA'}$, then we set $A_i = A^p$.
3. If $p = q$ and a'_p belongs to some subchain of A' of length at least two that is matched (in $W_{AA'}$) to a single element $a_l \in A$, we set $A_i = a_l$.

The subchains $B_1, \ldots, B_t$ are defined analogously.

We need two observations. The first one is that A_i and B_i are indeed subchains (consecutive sets of points). This is simply because the matchings of the points from A'_i and B'_i in $W_{AA'}$ and $W_{BB'}$, respectively, are sub-chains, and by definition $A_i = A^p A^{p+1} \cdots A^q$ is also a consecutive set of points. The second observation is that the subchains $A_1, \ldots, A_t$ (resp. $B_1, \ldots, B_t$) are almost-disjoint, in the sense that there can be only one point a_x that belongs

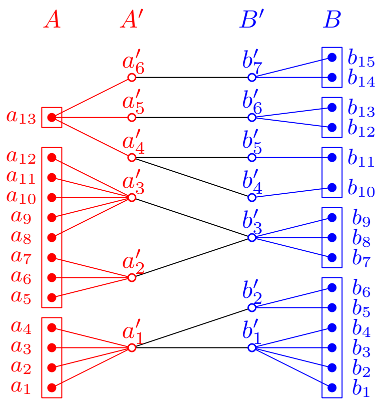


Figure 1 How does an optimal solution look like? a composition of pair-components: $W_{A'B'} = \{(\{a'_1\},\{b'_1,b'_2\}), (\{a'_2,a'_3\},\{b'_3\}), (\{a'_4\},\{b'_4,b'_5\}), (\{a'_5\},\{b'_6\}), (\{a'_6\},\{b'_7\})\}$
$(A_1 = A[1,4], B_1 = [1,6])$, $(A_2 = A[5,12], B_2 = B[7,9])$, $(A_3 = A[13], B_3 = B[10,11])$, $(A_4 = A[13], B_4 = B[12,13])$, $(A_5 = A[13], B_5 = B[14,15])$.

to both A_i and A_{i+1}, and in that case $A_i = A_{i+1} = (a_x)$. This is because if there were more than one point in common, or, if one of A_i, A_{i+1} contained more points, then the sets in $W_{AA'}$ (resp. $W_{BB'}$) were not disjoint.

So what does an optimal solution look like? It is composed of such almost-disjoint *pair-components*. A pair-component is a pair of sub-chains, (A_i, B_i), $A_i \subseteq A$, $B_i \subseteq B$, such that the points of A_i (resp. B_i) can be covered by one disk c of radius δ_1 (resp. δ_2), the points of B_i (resp. A_i) can be covered by a set C of disks of radius δ_2 (resp. δ_1), and for any $c' \in C$, the distance between the center of c and c' is at most δ_3.

The idea of the algorithm is to compute all the possible components (and that there are not too many of them), and then use dynamic programming to compute the optimal solution that is composed of pair-components.

3.2 The algorithm

For any two sub-chains $A[i,i']$ and $B[j,j']$ there are two possible types of pair-components. In the first type, there is only one disk that covers $A[i,i']$, and in the second type, there is only one disk that covers $B[j,j']$.

We denote by $PC_1[i,i',j,j']$ the size of the minimum-cardinality set C of disks of radius δ_2 needed in order to cover $B[j,j']$, such that there exists a disk c of radius δ_1 that covers $A[i,i']$, and for any $c' \in C$, the distance between the centers of c and c' is at most δ_3. Symmetrically, we define $PC_2[i,i',j,j']$. For any 4-tuple of indices (i,i',j,j') we need to compute $PC_1[i,i',j,j']$ and $PC_2[i,i',j,j']$.

Now, in order to compute an optimal solution, we need to combine pair-components in a way that will result in a simplification of minimum size. We use dynamic programming.

Let $OPT[i,j][r]$ be the minimum number of points in a simplification of $B[1,j]$ in an optimal solution for $A[1,i], B[1,j]$ in which the number of points in the simplification of $A[1,i]$ is at most r. Then we have the following dynamic programming algorithm: $OPT[1,1][r] = 1$ if and only if $||a_1 - b_1|| \le \delta_1 + \delta_2 + \delta_3$, and

$$OPT[1,j][r] = \min_{q \le j}\{OPT[1,q-1][r-1] + PC_1[1,1,q,j]\},$$

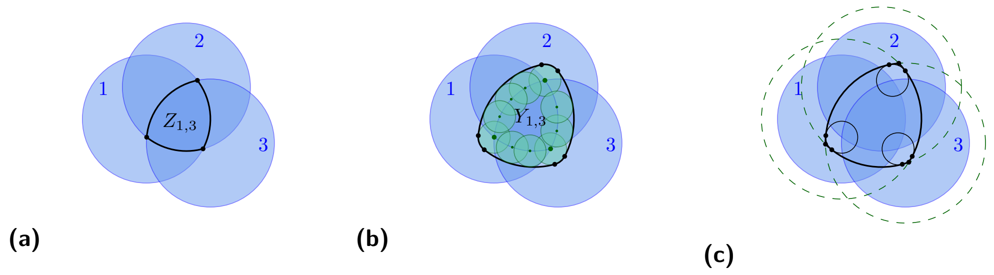


Figure 2 The blue filled disks represent $D(b_j, \delta_2)$ and the empty dashed green disks represent $D(b_j, \delta_2 + \delta_3)$. The small disks has radius δ_3.

$$OPT[i,1][r] = \min_{p \leq i}\{OPT[p-1,1][r - PC_2[p,i,1,1]] + 1\},$$

$$\begin{aligned} OPT[i,j][r] = \min_{p \leq i, q \leq j}\{&OPT[p-1,q-1][r-1] + PC_1[p,i,q,j], \\ &OPT[i,q-1][r-1] + PC_1[i,i,q,j], \\ &OPT[p-1,q-1][r - PC_2[p,i,q,j]] + 1, \\ &OPT[p-1,j][r - PC_2[p,i,j,j]] + 1\}. \end{aligned}$$

▶ **Theorem 1.** *For any i,j and r, $OPT[i,j][r]$ is the minimum number of points in a simplification of $B[1,j]$ in an optimal solution for $A[1,i], B[1,j]$ in which the number of points in the simplification of $A[1,i]$ is at most r.*

Proof. The proof is by induction on i, j, and r. For $i = 1$ and $j = 1$ the theorem holds by definition. Let A' and B' be an optimal solution for $A[1,i], B[1,j]$, s.t. $|A'| \leq r$. Let $[p,i,q,j]$ be the last pair-component in this solution. If $[p,i,q,j]$ is of type 1, i.e. there is one disk that covers $A[p,i]$ and $PC_1[p,i,q,j]$ disks that cover $B[q,j]$, then there are two possibilities: if $p = i$ and the pair-component that came before $[p,i,q,j]$ is $[i,i,q',q-1]$ for some $q' \leq q-1$, then $OPT[i,j][r] = OPT[i,q-1][r-1] + PC_1[i,i,q,j]$, else, $OPT[i,j][r] = OPT[p-1,q-1][r-1] + PC_1[p,i,q,j]$. If $[p,i,q,j]$ is of type 2, i.e. there is one point that covers $B[q,j]$ and $PC_2[p,i,q,j]$ points that cover $A[p,i]$, then again we have two possibilities, $OPT[i,j][r] = OPT[p-1,j][r - PC_2[p,i,j,j]] + 1$ or $OPT[i,j][r] = OPT[p-1,q-1][r - PC_2[p,i,q,j]] + 1$. ◀

3.3 Computing the components

Let $D(p, \delta)$ denote the disk centred at p with radius δ.

Recall that $PC_1[i,i',j,j']$ is the size of a minimum-cardinality set C of disks of radius δ_2 needed in order to cover $B[j,j']$, such that there exists a disk c of radius δ_1 that covers $A[i,i']$, and for any $c' \in C$, the distance between the centers of c and c' is at most δ_3.

We show how to find $PC_1[i,i',j,j']$ for all $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$ ($PC_2[i,i',j,j']$ is symmetric). We begin with a few observations to give an intuition for the algorithm.

Consider $PC_1[i,i',j,j']$. First, notice that the center of c is in the region $X_{i,i'} = \bigcap_{i \leq k \leq i'} D(a_k, \delta_1)$, because the distance from c to any point in $A[i,i']$ is at most δ_1.

Any $c' \in C$ is covering a consecutive subchain of $B[j,j']$. Thus, for any $j \leq t \leq t' \leq j'$, the center of a disk c' that covers the subsequence $B[t,t']$ (if exists) is in the region $Z_{t,t'} = \bigcap_{t \leq k \leq t'} D(b_k, \delta_2)$ (see Figure 2(a)). There are $O((j'-j)^2) = O(m^2)$ such regions.

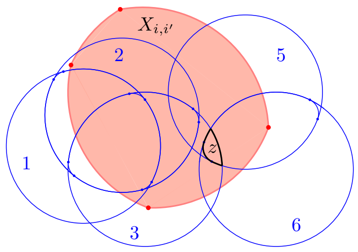


Figure 3 The arrangement $\mathcal{A}(D_A)$. After computing $Size_A(X_{1,4}, j, j')$, we know that $Size_A(X_{1,3}, j, j')$ is the minimum between $Size_A(X_{1,4}, j, j')$ and the values of the cells in $O_{1,3}$.

Each such region is convex and composed of linear number of arcs. Any point placed inside $Z_{t,t'}$ can cover $B[t, t']$, and we need a point with distance at most δ_3 to the center of c. For each $Z_{t,t'}$, consider the Minkowski sum $Y_{t,t'} = Z_{t,t'} \oplus \delta_3$ (see Figure 2(b)).

Now, consider the arrangement obtained by the intersection of $X_{i,i'}$ and the arrangement of $\{Y_{t,t'} \mid j \leq \ t \leq t' \leq j'\}$ (see Figure 3). Each cell in this arrangement corresponds to a set of $Y_{t,t'}$'s, each has some point with distance at most δ_3 to the same points in $X_{i,i'}$. Each cell corresponds to a possible choice of the center of c, or, in other words, a possible pair-component of type 1.

We now describe an algorithm for computing $PC_1[i, i', j, j']$ for all $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$. The algorithm is quite complex and has several sub-procedures.

Let $X = \{X_{i,i'} = \bigcap\limits_{i \leq k \leq i'} D(a_k, \delta_1) \mid 1 \leq i \leq i' \leq n\}$. The number of shapes in X is $O(n^2)$.

Let $Y = \{Y_{j,j'} \mid 1 \leq j \leq j' \leq m,\ Z_{j,j'} \neq \emptyset\}$. The number of shapes in Y is $O(m^2)$, each shape is of complexity $O(m)$.

Consider the arrangement $\mathcal{A}(Y)$ of the shapes in Y.

▶ **Lemma 2.** *The number of cells in $\mathcal{A}(Y)$ is $O(m^4)$.*

Proof. Let P be the set of intersection points between the disks in $\{D(b_j, \delta_2) \mid 1 \leq j \leq m\}$. Consider the following set of disks: $D = \{D(b_i, \delta_2 + \delta_3) \mid 1 \leq i \leq m\} \cup \{D(p, \delta_3) \mid p \in P\}$. Notice that the arcs and vertices of $\mathcal{A}(Y)$ are a subset of the arcs and vertices of $\mathcal{A}(D)$ (see Figure 2(c)). Since the number of points in P is $O(m^2)$, we get that the number of disks in $\mathcal{A}(D)$ is $O(m^2)$, and thus the complexity of $\mathcal{A}(D)$ is $O(m^4)$. ◀

Notice that for any shape $Y_{j,j'} \in Y$ and a cell $z \in \mathcal{A}(Y)$ it holds that $Y_{j,j'} \cap z \neq \emptyset$ if and only if $z \subseteq Y_{j,j'}$. For each cell $z \in \mathcal{A}(Y)$, let Y_z be the set of $O(m^2)$ shapes from Y that contain z. The algorithm has two main steps:

1. For each cell $z \in \mathcal{A}(Y)$, and for any two indices $1 \leq j \leq j' \leq m$, compute $Size_B(z, j, j')$ – the minimum number of shapes from Y_z needed in order to cover the points of $B[j, j']$. Recall that a shape $Y_{t,t'} \in Y_z$ covers the subsequence $B[t, t']$, in other words, there exists a point q in $Y_{t,t'}$ s.t. $d(q, b_k) \leq \delta_2$ for any $t \leq k \leq t'$.
2. For each shape $X_{i,i'} \in X$, and for any two indices $1 \leq j \leq j' \leq m$, compute $Size_A(X_{i,i'}, j, j') = \min\limits_{z \cap X_{i,i'} \neq \emptyset} Size_B(z, j, j')$.

Note that $Size_A(X_{i,i'}, j, j') = PC_1[i, i', j, j']$.

Algorithm 1 $Size_B(Y_z)$

```
For $j$ from 1 to $m$:
  1. Set $counter \leftarrow 1$
  2. Set $j' \leftarrow j$.
  3. Set $p \leftarrow \max\{next(Y_{j,j'}), max(j'+1)\}$.
  4. While $p \neq -\infty$:
       For $k$ from $j'$ to $p$: Set $Size_B(z,j,k) \leftarrow counter$.
       Set $counter \leftarrow counter + 1$
       Set $p \leftarrow \max\{next(Y_{j',k}), max(k+1)\}$.
       Set $j' \leftarrow k$.
```

Step 1

First we have to find the set Y_z for each cell $z \in \mathcal{A}(Y)$. We start by computing Y: for any j, j' we check whether $\bigcap_{j \le k \le j'} D(b_k, \delta_2) \neq \emptyset$. If yes, we add $Y_{j,j'}$ to Y. This can be done in $O(m^3)$ time. Then we compute the arrangement $\mathcal{A}(Y)$, while maintaining the lists Y_z for any cell $z \in \mathcal{A}(Y)$. This can be done in $O(m^4)$ as the complexity of $\mathcal{A}(Y)$ is $O(m^4)$.

Now, for each cell $z \in \mathcal{A}(Y)$ we compute $Size_B(z, j, j')$ for all $1 \le j \le j' \le m$ as follows: Notice that the problem of finding a minimum cover to $B[j, j']$ from a set of subsequences, is actually an interval-cover problem. We refer to the shapes in Y_z as intervals (between 1 and m), and the goal is to find the minimum number of intervals from Y_z needed in order to cover the interval $[j, j']$.

First, for every $1 \le j \le n$ we find $max(j)$ - the largest interval from Y_z that starts at j. This can be done simply in $O(m^2 \log m)$ time, by sorting the intervals first by their lower bound and then by their upper bound.

Next, for an interval $Y_{t,t'} \in Y_z$, consider the intervals in Y_z whose lower bound lies in $[t, t']$ and whose upper bound is greater than t'. Let $next(Y_{t,t'})$ be the largest upper bound among the upper bounds of these intervals. We can find $next(Y_{t,t'})$, for each $Y_{t,t'} \in Y_z$, in total time $O(m^2 \log m)$, using a segment tree as follows: Let $S = \{s_1, \ldots, s_n\}$ be a set of line segments on the x-axis, $s_i = [a_i, b_i]$. Construct a segment tree for the set S. With each vertex v of the tree, store a variable r_v, whose initial value is $-\infty$. Query the tree with each of the left endpoints. When querying with a_i, in each visited vertex v with non-empty list of segments do: if $b_i > r_v$, then set r_v to b_i. Finally, for each segment s, let $next(s)$ be the maximum over the values r_v of the vertices storing s.

After computing $next(Y_{t,t'})$ for all $Y_{t,t'} \in Y_z$ (assume $next(Y_{t,t'}) = -\infty$ for $Y_{t,t'} \notin Y_z$), we use Algorithm 1 to compute $Size_B(z, j, j')$ for all $1 \le j \le j' \le m$. The running time of Algorithm 1 is $O(m^2)$. Thus, computing $Size_B(z, j, j')$ for all cells $z \in \mathcal{A}(Y)$ and all indices $1 \le j \le j' \le m$ takes $O(m^6 \log m)$ time.

Step 2

Recall that $\mathcal{A}(Y)$ is the arrangement obtained from the shapes in Y. Let $\mathcal{A}(D_A)$ be the arrangement of the disks $D_A = \{D(a_k, \delta_1) \mid 1 \le k \le n\}$. The number of cells in $\mathcal{A}(D_A)$ is $O(n^2)$.

A trivial algorithm to compute the value $Size_A(X_{i,i'}, j, j')$ is by considering the values $Size_B(z, j, j')$ of $O(m^4)$ cells from $\mathcal{A}(Y)$. Since there are $O(n^2)$ shapes $X_{i,i'} \in X$ and $O(m^2)$ pairs of indices $1 \le j \le j' \le m$, the running time will be $O(n^2 m^6)$. We manage to reduce

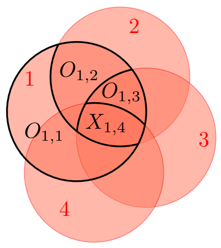


Figure 4 The arrangement $\mathcal{A}(D_A)$. After computing $Size_A(X_{1,4}, j, j')$, we know that $Size_A(X_{1,3}, j, j')$ is the minimum between $Size_A(X_{1,4}, j, j')$ and the values of the cells in $O_{1,3}$.

the running time by a factor of $O(n)$, using some properties of the arrangement of disks.

Let $\mathcal{U}$ be the arrangement of the shapes in Y and the disks in D_A. Notice that $\mathcal{U}$ is a union of the arrangements $\mathcal{A}(D_A)$ and $\mathcal{A}(Y)$.

▶ **Lemma 3.** *The number of cells in $\mathcal{U}$ is $O((m^2 + n)^2)$.*

The proof is similar to the proof of Lemma 2.

We make a few quick observations:

▶ **Observation 1.** For any two cells $w \in \mathcal{U}, x \in \mathcal{A}(D_A)$, $x \cap w \neq \emptyset$ if and only if $w \subseteq x$. Similarly, for any cell $z \in \mathcal{A}(Y)$, $z \cap w \neq \emptyset$ if and only if $w \subseteq z$.

▶ **Observation 2.** For any cell $x \in \mathcal{A}(D_A)$, if $X_{i,i'} \cap x \neq \emptyset$, then $x \subseteq X_{i,i'}$.

▶ **Observation 3.** For any $1 \leq i \leq i' \leq n$ we have $X_{i,i'+1} \subseteq X_{i,i'}$.

Given $w \in \mathcal{U}$, let z_w be the cell from $\mathcal{A}(Y)$ that contains w. We have $Size_B(w, j, j') = Size_B(z, j, j')$.

Let $O_{i,i'}$ be the set of cells $w \in \mathcal{U}$ s.t. $w \subseteq X_{i,i'}$ and $w \nsubseteq X_{i,i'+1}$.

For fixed $1 \leq j \leq j' \leq m$ and $1 \leq i \leq n$, the idea is to compute the values $Size_A(X_{i,n}, j, j'), Size_A(X_{i,n-1}, j, j'), \ldots, Size_A(X_{i,i}, j, j')$ in this order, so we can use the value of $Size_A(X_{i,i'+1}, j, j')$ in order to compute $Size_A(X_{i,i'}, j, j')$, adding only the values of the cells in $O_{i,i'}$ (see Figure 4). This way, any cell in $\mathcal{U}$ will be checked only once (for any fixed $1 \leq j \leq j' \leq m$ and $1 \leq i \leq n$), and the running time will be $O(m^2 n(n + m^2)^2)$.

Now we have to show how to find the sets $O_{i,i'}$.

First, for any cell $x \in \mathcal{A}(D_A)$ we find all the cells $w \in \mathcal{U}$ such that $w \subseteq x$. There are $O(n^2)$ cells in $\mathcal{A}(D_A)$, but from Observation 1 we keep a total of $O((m^2 + n)^2)$ cells from $\mathcal{U}$.

Then, for any shape $X_{i,i'} \in X$ we find the set of cells $P_{i,i'}\{x \in \mathcal{A}(D_A) \mid x \subseteq X_{i,i'}\}$. There are $O(n^2)$ shapes in X, and for each shape we keep $O(n^2)$ cells from $\mathcal{A}(D_A)$.

Now we have $O_{i,i'} = P_{i,i'} \setminus P_{i,i'+1}$. The size of $P_{i,i'}$ is $O(n^2)$, so computing $O_{i,i'}$ for all $1 \leq i \leq i' \leq n$ takes $O(n^4)$ time.

The total running time for all $PC_1[i, i', j, j']$ is $O(m^6 \log m + m^2 n(n + m^2)^2)$

Total running time

For computing $PC_2[i, i', j, j']$ we get symmetrically a total running time of $O(n^6 \log n + n^2 m(m + n^2)^2)$, so the running time for computing all the components is $\widetilde{O}((m + n)^6 \min\{m, n\})$. Calculating $OPT[i, j][r]$ takes $O(m^2 n^2 \min\{m, n\})$ time, all together takes $\widetilde{O}((m + n)^6 \min\{m, n\})$ time.

Algorithm 2

```
Find $X_{i,i'} = \bigcap_{i \le k \le i'} D(a_k, \delta_1)$.
Set $R \leftarrow \mathbb{R}$.
Set $counter \leftarrow 1$.
Set $k \leftarrow j$.
While $k \le j'$ and $counter \neq \infty$:
  1. Set $R \leftarrow R \cap D(b_k, \delta_2)$.
  2. If $(X_{i,i'} \oplus \delta_3) \cap R \neq \emptyset$, set $APC_1[i,i',j,k] \leftarrow counter$.
  3. Else,
       Set $R \leftarrow D(b_k, \delta_2)$.
       If $(X_{i,i'} \oplus \delta_3) \cap R \neq \emptyset$, set $counter \leftarrow counter + 1$.
       Else, set $counter \leftarrow \infty$.
       Set $APC_1[i,i',j,k] \leftarrow counter$.
  4. Set $k \leftarrow k + 1$.
```

4 Approximating GCPS

All the missing proofs of this section can be found in the full version of the paper.

To approximate GCPS, we use approximated pair-components which are easier to compute.

Let $APC_1[i,i',j,j']$ be the minimum number of disks with radius δ_2 needed in order to cover the points of $B[j,j']$ (in order), and whose centers are located in $X_{i,i'} \oplus \delta_3$. Similarly, let $APC_2[i,i',j,j']$ be the minimum number of disks with radius δ_1 needed in order to cover the points of $A[i,i']$ (in order), and whose centers are located in $Z_{j,j'} \oplus \delta_3$.

▶ **Lemma 4.** *For any $1 \le i \le i' \le n, 1 \le j \le j' \le m$, $APC_1[i,i',j,j'] \le PC_1[i,i',j,j']$.*

4.1 Computing the approximated components

We present a greedy algorithm that given $1 \le i \le i' \le n, 1 \le j \le j' \le m$, computes $APC_1[i,i',j,k]$ for all $j \le k \le j'$ (resp. $APC_2[i,k,j,j']$ for all $i \le k \le i'$). The algorithm runs in $O((j'-j)(j'-j+i'-i))$ time (See Algorithm 2).

Running time

Finding $X_{i,i'}$ takes $O(i'-i)$ time, and step 1 takes $O(j'-j)$ time. Step 2 takes $O(j'-j+i'-i)$ time, since the complexity of $X_{i,i'} \oplus \delta_3$ is $O(i'-i)$, the complexity of R is $O(j'-j)$, and both regions are convex. The while loop runs $O(j'-j)$ times, so the total running time is $O((j'-j)(j'-j+i'-i))$.

Computing all the approximated pair components using Algorithm 2 takes $O(n^2m^2(m+n))$ time. The idea of our algorithm is to compute only a small part of the components, and then approximate the others using the ones that were computed.

▶ **Lemma 5.** *Fix $1 \le i \le i' \le n, 1 \le j \le j' \le m$, then for any $i \le x \le i'$ and $j \le y \le j'$:*
1. *$APC_1[i,x,j,j'] \le APC_1[i,i',j,j']$ and $APC_1[x,i',j,j'] \le APC_1[i,i',j,j']$.*
2. *$APC_1[i,i',j,y] + APC_1[i,i',y,j'] \le APC_1[i,i',j,j'] + 1$.*
3. *$APC_1[i,x,j,y] + APC_1[x,i',y,j'] \le APC_1[i,i',j,j'] + 1$.*

We only compute $APC_1[i,i,j,j']$,$APC_2[i,i,j,j']$ for all $1 \le i \le n$ and $1 \le j \le j' \le m$, and $APC_1[i,i',j,j]$,$APC_2[i,i',j,j]$ for all $1 \le i \le i' \le n$ and $1 \le j \le m$. This takes $O(nm^3 + n^2m^2)$ time using Algorithm 2.

4.2 Composing the approximated solution

Let $AAPC_1[i,i',j,j'] = APC_1[i,i,j,j'] + APC_1[i,i',j',j']$. By Lemma 5(3), choosing $x = i$ and $y = j'$, we have $APC_1[i,i,j,j'] + APC_1[i,i',j',j'] \leq APC_1[i,i',j,j'] + 1$, and by Lemma 4 we have $AAPC_1[i,i',j,j'] \leq PC_1[i,i',j,j'] + 1$.

Now let $APX[i,j]$ be the approximate solution for $A[1,i]$ and $B[1,j]$. We set

$$APX[i,j] = \min_{p<i,q<j} APX[p,q] + \min\{AAPC_1[p+1,i,q+1,j], AAPC_2[p+1,i,q+1,j]\}$$

Obviously, given the values of $AAPC_1$ and $AAPC_2$, $APX[n,m]$ can be computed in $O(m^2n^2)$ time.

▶ **Lemma 6.** *Let OPT be the size of an optimal solution, i.e. OPT is the smallest number such that there exists a pair of chains A',B' each of at most OPT (arbitrary) vertices, such that $d_1(A,A') \leq \delta_1$, $d_2(B,B') \leq \delta_2$, and $d_{dF}(A',B') \leq \delta_3$. Then $APX[n,m] \leq 2 \cdot OPT$.*

Thus we have the following theorem:

▶ **Theorem 7.** *A 2-approximation for GCPS can be computed in $O(nm^3 + n^2m^2 + n^3m)$ time.*

▶ Remark. Notice that we do not have to actually compute a solution to GCPS, just to return the minimum k. A solution of size $2 \cdot OPT$ can be computed as follows: for each approximated component $APC_1[i,i',j,j']$ (or $APC_2[i,i',j,j']$) keep the set C_1 of centers of disks that are located in $X_{i,i'} \oplus \delta_3$. For each such center $c_1 \in C_1$, find a point c_2 in $X_{i,i'}$ s.t. $d(c_1,c_2) \leq \delta_3$, and put c_2 in a new set C_2. If our solution $APX[n,m]$ uses the approximated component $APC_1[i,i',j,j']$, then the points of C_1 will be used to cover $B[j,j']$ and the points of C_2 will be used to cover $A[i,i']$.

5 1-Sided GCPS

As in [9], we consider the 1-sided variant of GCPS. In this variant we can imagine there are two dogs, one is walking on a path A and the other on a path B, and a man has to walk both of them, one with a leash of length δ_1 and the other with a leash of length δ_2. We have to find a minimum-size polygonal path for the man, such that he can walk both dogs together.

▶ **Problem 2** (1-Sided General Chain Pair Simplification).
Instance: Given a pair of polygonal chains A and B of lengths n and m, respectively, an integer k, and two real numbers $\delta_1, \delta_2 > 0$.
Problem: Does there exist a chain C of at most k (arbitrary) vertices, such that $d_{dF}(A,C) \leq \delta_1$ and $d_{dF}(B,C) \leq \delta_2$?

Denote $X_{i,i'} = \bigcap_{i \leq k \leq i'} D(a_k, \delta_1)$ and $Z_{j,j'} = \bigcap_{j \leq k \leq j'} D(b_k, \delta_2)$ as before.

For any $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$, let $I[i,i',j,j'] = \begin{cases} 1, & X_{i,i'} \cap Z_{j,j'} \neq \emptyset \\ 0, & otherwise \end{cases}$.
Notice that $I[i,i',j,j'] = 1$ if and only if there exists one point that covers both $A[i,i']$ and $B[j,j']$. The values of $I[i,i',j,j']$ can be computed in $O((n+m)^4)$ time (the details can be found in the full version of the paper).

Now we use a dynamic programming algorithm as follows: Let $OPT[i,j]$ be the length of the minimum-length sequence C such that $d_{dF}(A[1,i],C) \leq \delta_1$ and $d_{dF}(B[1,j],C) \leq \delta_2$. Fix $i,j > 1$, we have $OPT[i,j] = \min_{p,q:I[p,i,q,j]=1} \{OPT[p-1,q-1]+1\}$.

Running time

The values of $I[i,i',j,j']$ can be computed in $O((n+m)^4)$ time. For each $i,j>1$, we have $O(mn)$ values to check. Thus, the running time is $O((m+n)^4)$.

6 GCPS under the Hausdorff distance

The Hausdorff distance between two sets of points A and B is defined as follows:

$$d_H(A,B) = \max\{\max_{a\in A}\min_{b\in B} d(a,b),\ \max_{b\in B}\min_{a\in A} d(a,b)\}.$$

As mentioned above, the chain pair simplification under the Hausdorff distance (CPS-2H) is **NP**-complete. In this section we investigate the general version of this problem. We prove that it is also **NP**-complete, and give an approximation algorithm for the problem.

6.1 GCPS-2H is NP-complete

We show that GCPS under Hausdorff distance (GCPS-2H) is **NP**-complete, we use a simple reduction from geometric set cover: Given a set P of n points, and a radius δ, find the minimum number of disks with radius δ that cover P.

Let the sequence A be the points of P in some order (the order does not matter), and the sequence B be one point b with distance 2δ from P. Let $\delta_1=\delta_2=\delta$ and $\delta_3=4\delta+diam(P)$. Now a simplification for B is just one point anywhere in $D(b,\delta)$, and finding a simplification for A is equivalent to finding the minimum-cardinality set of disks that covers P.

▶ **Theorem 8.** *GCPS-2H is* ***NP****-complete.*

6.2 An approximation algorithm for GCPS-2H

Consider the variant of GCPS-2H where $d_1=d_2=d_H$ and the distance between the simplifications A' and B' is measured with Hausdorff distance and not Fréchet distance (i.e. $d_H(A',B')\le\delta_3$ instead of $d_{dF}(A',B')\le\delta_3$). We call this variant GCPS-3H, and show that GCPS-3H=GCPS-2H.

▶ **Lemma 9.** *Given two sets of points A and B, if $d_H(A,B)\le\delta$, then there exist an ordering A' of the points in A and an ordering B' of the points in B, such that $d_{dF}(A',B')\le\delta$.*

Proof. We construct a bipartite graph $G(V=A\cup B,E)$, where $E=\{(a,b)\mid a\in A,\ b\in B,\ d(a,b)\le\delta\}$. Notice that since $d_H(A,B)\le\delta$, there are no isolated vertices. Now, while there exists a path with three edges in the graph, delete the middle edge. The maximal path in the resulting graph G' has at most two edges, and there are still no isolated vertices (because we only delete the middle edge). Let $C_1,\dots,C_t$ be the connected components of G'. Notice that each C_i has exactly one point from A or exactly one point from B. Let A' be the sequence of points $C_1\cap A,\dots,C_t\cap A$, and B' be the sequence $C_1\cap B,\dots,C_t\cap B$. We get that $C_1,\dots,C_t$ are a paired walk along A' and B' with cost at most δ. ◀

Since we can choose the order of points in the simplifications A' and B' in the GCPS-2H problem, we get by the above lemma that any solution for GCPS-3H is also a solution for GCPS-2H. Now, since for any two sequence P,Q we have $d_H(P,Q)\le d_{dF}(P,Q)$, we get that any solution for GCPS-2H is also a solution for GCPS-3H.

Let $S_1=\{p_1,\dots,p_k\}$ be the smallest set of points such that for each $a_i\in A$ there exists some $p_j\in S_1$ s.t. $d(a_i,p_j)\le\delta_1$ and for each $p_j\in S_1$ there exists some $b_i\in B$ s.t.

$d(p_j, b_i) \le \delta_2 + \delta_3$. Notice that since S_1 is minimum, we also know that for each $p_j \in S_1$ there exists some $a_i \in A$ s.t. $d(a_i, p_j) \le \delta_1$ (or, we can just delete the points of S_1 that do not cover any points from A).

We can find a c-approximation for S_1, using a c-approximation algorithm for discrete unit disk cover (DUDC). The DUDC problem is defined as follows: Given a set P of t points and a set D of k unit disks on a 2-dimensional plane, find a minimum-cardinality subset $D' \subseteq D$ such that the unit disks in D' cover all the points in P. We denote by $T_c(k, t)$ the running time for a c-approximation algorithm for the DUDC problem with k unit disks and t points.

▶ **Lemma 10.** *Given a c-approximation algorithm for the DUDC problem that runs in $T_c(k, t)$ time, we can find a c-approximation for S_1 in $T_c(n, (m + n)^2) + O((m + n)^2)$ time.*

Proof. Compute the arrangement of $\{D(a_i, \delta_1)\}_{1 \le i \le m} \cup \{D(b_j, \delta_2 + \delta_3)\}_{1 \le j \le n}$ (there are $(m + n)^2$ disjoint cells in the arrangement). Clearly, it is enough to choose one candidate from each cell. Now we can use the c-approximation algorithm for the DUDC problem. ◀

Symmetrically, let $S_2 = \{q_1, \dots, q_l\}$ be the smallest set of points such that for each $b_i \in B$ there exists some $q_j \in S_2$ s.t. $d(b_i, q_j) \le \delta_2$ and for each $q_j \in S_2$ there exists some $a_i \in A$ s.t. $d(q_j, a_i) \le \delta_1 + \delta_3$.

For each point $p_j \in S_1$ there exists some $b_i \in B$ s.t. $d(p_j, b_i) \le \delta_2 + \delta_3$, so we can find a point p'_j such that $d(p'_j, b_i) \le \delta_2$ and $d(p'_j, p_j) \le \delta_3$. Denote $S'_1 = \{p'_1, \dots, p'_k\}$. We do the same for the points of S_2, and find a set $S'_2 = \{q'_1, \dots, q'_k\}$ such that for any $q'_j \in S'_2$,$d(q'_j, q_j) \le \delta_3$ and there exists some $a_i \in A$ s.t. $d(q'_j, a_i) \le \delta_1$.

Now, we know that for each $a_i \in A$ there exists some $p \in S_1 \cup S'_2$ s.t. $d(a_i, p) \le \delta_1$, and, on the other hand, for each $p \in S_1 \cup S'_2$ there exists some $a_i \in A$ s.t. $d(a_i, p) \le \delta_1$. So we have $d_H(A, S_1 \cup S'_2) \le \delta_1$. Similarly, we have $d_H(B, S_2 \cup S'_1) \le \delta_2$. We also know that for each $p_j \in S_1$ we have a point $p'_j \in S'_1$ s.t. $d(p'_j, p_j) \le \delta_3$, and for each $q'_j \in S'_2$ we have a point $q_j \in S_2$ s.t. $d(q'_j, q_j) \le \delta_3$. So we also have $d_H(S_1 \cup S'_2, S_2 \cup S'_1) \le \delta_3$, and since CPS-2H=CPS-3H, we get that $S_1 \cup S'_2$ and $S_2 \cup S'_1$ is a possible solution for CPS-2H.

The size of the optimal solution OPT is at least $\max\{|S_1|, |S_2|\}$. Using a c-approximation algorithm for finding S_1 and S_2, the size of the approximate solution will be $c(|S_1| + |S_2|) \le 2c \max\{|S_1| + |S_2|\} = 2c \cdot OPT$.

▶ **Theorem 11.** *Given a c-approximation algorithm for the DUDC problem that runs in $T_c(k, t)$ time, our algorithm gives a $2c$-approximation for the GCPS-2H problem, and runs in $T_c(n, (m + n)^2) + T_c(m, (m + n)^2) + O((m + n)^2)$ time.*

Acknowledgements. The authors would like to Michael Kerber for suggesting the problem.

References

1 Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geometry Appl.*, 5:75–91, 1995.

3 Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, and Binhai Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proc. 8th Latin American Theoretical Informatics Sympos., LATIN'08*, pages 630–641, 2008.

4 Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th IEEE Annual Sympos. on Foundations of Computer Science, FOCS'14*, pages 661–670, 2014.

5 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *JoCG*, 7(2):46–76, 2016.

6 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog — with an application to Alt's conjecture. In *Proc. 25th Annual ACM-SIAM Sympos. on Discrete Algorithms, SODA'14*, pages 1399–1413, 2014.

7 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013.

8 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Information Systems Dept., Technical University of Vienna, 1994.

9 Chenglin Fan, Omrit Filtser, Matthew J. Katz, Tim Wylie, and Binhai Zhu. On the chain pair simplification problem. In *Algorithms and Data Structures - 14th Internat. Symp., WADS 2015*, pages 351–362, 2015.

10 Michael Godau. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *STACS 91, 8th Annual Sympos. on Theoretical Aspects of Computer Science*, pages 127–136, 1991.

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

12 Tim Wylie, Jun Luo, and Binhai Zhu. A practical solution for aligning and simplifying pairs of protein backbones under the discrete Fréchet distance. In *Proc. Internat. Conf. Computational Science and Its Applications, ICCSA'11, Part III*, pages 74–83, 2011.

Computing DAWGs and Minimal Absent Words in Linear Time for Integer Alphabets

Yuta Fujishige[1], Yuki Tsujimaru[2], Shunsuke Inenaga[3], Hideo Bannai[4], and Masayuki Takeda[5]

1 **Department of Informatics, Kyushu University, Japan**
`yuta.fujishige@inf.kyushu-u.ac.jp`
2 **Department of Electrical Engineering and Computer Science, Kyushu University, Japan**
3 **Department of Informatics, Kyushu University, Japan**
`inenaga@inf.kyushu-u.ac.jp`
4 **Department of Informatics, Kyushu University, Japan**
`bannai@inf.kyushu-u.ac.jp`
5 **Department of Informatics, Kyushu University, Japan**
`takeda@inf.kyushu-u.ac.jp`

Abstract

The *directed acyclic word graph* (*DAWG*) of a string y is the smallest (partial) DFA which recognizes all suffixes of y and has only $O(n)$ nodes and edges. We present the first $O(n)$-time algorithm for computing the DAWG of a given string y of length n over an integer alphabet of polynomial size in n. We also show that a straightforward modification to our DAWG construction algorithm leads to the first $O(n)$-time algorithm for constructing the *affix tree* of a given string y over an integer alphabet. Affix trees are a text indexing structure supporting bidirectional pattern searches. As an application to our $O(n)$-time DAWG construction algorithm, we show that the set $MAW(y)$ of all minimal absent words of y can be computed in *optimal* $O(n + |MAW(y)|)$ time and $O(n)$ working space for integer alphabets.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases string algorithms, DAWGs, suffix trees, minimal absent words

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.38

1 Introduction

Text indexes are fundamental data structures that allow for efficient processing of string data, and have been extensively studied. Although there are several alternative data structures which can be used as an index, such as suffix trees [18] and suffix arrays [11], in this paper, we focus on *directed acyclic word graphs* (*DAWGs*) proposed by Blumer et al. [3]. Intuitively, the DAWG of string y, denoted $DAWG(y)$, is an edge-labeled DAG obtained by merging isomorphic subtrees of the trie representing all suffixes of string y, called the suffix trie of y. Hence, $DAWG(y)$ can be seen as an automaton recognizing all suffixes of y. Let n be the length of the input string y. Despite the fact that the number of nodes and edges of the suffix trie can be as large as $O(n^2)$, Blumer et al. [3] proved that, surprisingly, $DAWG(y)$ has at most $2n - 1$ nodes and $3n - 4$ edges for $n > 2$. Crochemore [5] showed that $DAWG(y)$ is the smallest (partial) automaton recognizing all suffixes of y, namely, the sub-tree merging operation which transforms the suffix trie to $DAWG(y)$ indeed minimizes the automaton.

Since $DAWG(y)$ is a DAG, in general, more than one string can be represented by its node. It is known that every string represented by the same node of $DAWG(y)$ has the

Table 1 Space requirements and construction times for text indexing structures for input strings of length n over an alphabet of size σ.

	space (in words)	construction time		
		ordered alphabet	integer alphabet	constant alphabet
suffix tries	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
suffix trees	$O(n)$	$O(n \log \sigma)$ [12]	$O(n)$ [8]	$O(n)$ [18]
suffix arrays	$O(n)$	$O(n \log \sigma)$ [12]+[11]	$O(n)$ [8]+[11]	$O(n)$ [18]+[11]
DAWGs	$O(n)$	$O(n \log \sigma)$ [3]	$O(n)$ [this work]	$O(n)$ [3]
CDAWGs	$O(n)$	$O(n \log \sigma)$ [4]	$O(n)$ [14]	$O(n)$ [4]
affix trees	$O(n)$	$O(n \log \sigma)$ [10]	$O(n)$ [this work]	$O(n)$ [10]

same set of ending positions in the string y. Due to this property, if z is the longest string represented by a node v of $DAWG(y)$, then any other string represented by the node v is a proper suffix of z. Hence, the *suffix link* of each node of $DAWG(y)$ is well-defined; if ax is the shortest string represented by node v where a is a single character and x is a string, then the suffix link of ax points to the node of $DAWG(y)$ that represents string x.

One of the most intriguing properties of DAWGs is that the suffix links of $DAWG(y)$ for any string y forms the suffix tree [18] of the reversed string of y. Hence, $DAWG(y)$ augmented with suffix links can be seen as a *bidirectional* text indexing data structure. This line of research was followed by other types of bidirectional text indexing data structures such as *symmetric compact DAWGs* (*SCDAWGs*) [4] and *affix trees* [15, 10]. DAWGs with suffix links also have applications to other kinds of string processing problems which are not always easily solvable by using suffix trees or arrays, such as: finding *minimal absent words* for a given string [7, 16], finding *α-gapped repeats* that occur in a given string [17], finding *maximal-exponent repeats* in a given overlap-free string [1], computing the *Lempel-Ziv 77 factorization* [20] of a given string in an online manner and with compact space [19].

Time complexities for constructing text indexing data structures depend on the underlying alphabet. See Table 1. For a given string y of length n over an ordered alphabet of size σ, the suffix tree [12], the suffix array [11], the DAWG, and the *compact DAWGs* (*CDAWGs*) [4] of y can all be constructed in $O(n \log \sigma)$ time. These immediately lead to $O(n)$-time construction algorithms for a constant alphabet.

In this paper, we are particularly interested in input strings of length n over an *integer alphabet* of polynomial size in n. Farach-Colton et al. [8] proposed the first $O(n)$-time suffix tree construction algorithm for integer alphabets. Since the out-edges of every node of the suffix tree constructed by McCreight's [12] and Farach-Colton et al.'s algorithms are lexicographically sorted, and since sorting is an obvious lower-bound for constructing edge-sorted suffix trees, the above-mentioned suffix-tree construction algorithms are optimal for ordered and integer alphabets, respectively. Since the suffix array of y can be easily obtained in $O(n)$ time from the edge-sorted suffix tree of y, suffix arrays can also be constructed in optimal time. In addition, since the edge-sorted suffix tree of y can easily be constructed in $O(n)$ time from the edge-sorted CDAWG of y, and since the edge-sorted CDAWG of y can be constructed in $O(n)$ time from the edge-sorted DAWG of y [4], sorting is also a lower-bound for constructing edge-sorted DAWGs and edge-sorted CDAWGs. Using the technique of Narisawa et al. [14], edge-sorted CDAWGs can be constructed in optimal $O(n)$ time for integer alphabets. On the other hand, the only known algorithm to construct DAWGs was Blumer et al.'s $O(n \log \sigma)$-time online algorithm [3] for ordered alphabets of size σ, which results in $O(n \log n)$-time DAWG construction for integer alphabets.

In this paper, we close the gap between the upper and lower bounds for DAWG construction, by proposing the first $O(n)$-time algorithm to construct edge-sorted DAWGs for integer alphabets. Our algorithm also computes the suffix links, and can thus be applied to various kinds of string processing problems. Our algorithm builds $\mathit{DAWG}(y)$ for a given string y by transforming the suffix tree of y to $\mathit{DAWG}(y)$. In other words, our algorithm simulates the minimization of the suffix trie of y to $\mathit{DAWG}(y)$ using only $O(n)$ time and space.

A simple modification to our $O(n)$-time DAWG construction algorithm also leads us to the first $O(n)$-time algorithm to construct affix trees for integer alphabets. We remark that the previous best known affix-tree construction algorithm of Maaß [10] requires $O(n \log n)$ time for integer alphabets.

As an application of our $O(n)$-time DAWG construction algorithm, we present the first optimal time algorithm to compute *minimal absent words* for a given string. Let $\mathit{MAW}(y)$ be the set of minimal absent words of y. Crochemore et al. [7] proposed an algorithm to compute $\mathit{MAW}(y)$ in $\Theta(n\sigma)$ time and $O(n)$ working space. Their algorithm first constructs $\mathit{DAWG}(y)$ with suffix links in $O(n \log \sigma)$ time and compute $\mathit{MAW}(y)$ in $O(n\sigma)$ time using $\mathit{DAWG}(y)$ and its suffix links. Since $|\mathit{MAW}(y)| = O(n\sigma)$, the output size $|\mathit{MAW}(y)|$ is hidden in the running time of their algorithm. In this paper, we show that $\mathit{MAW}(y)$ can be computed in *output-sensitive* $O(n + |\mathit{MAW}(y)|)$ optimal time for integer alphabets. We first construct edge-sorted $\mathit{DAWG}(y)$ in $O(n)$ time using the algorithm we propose in this paper. Then, we show that a slight modification to Crochemore et al.'s algorithm [7] finds $\mathit{MAW}(y)$ in $O(n + |\mathit{MAW}(y)|)$ time. We emphasize that for non-constant alphabets Crochemore et al.'s algorithm takes super-linear time in terms of the input string length independently of the output size $|\mathit{MAW}(y)|$, and thus our results greatly improves the efficiency for integer alphabets. Belazzougui et al. [2] showed that using a representation of the bidirectional BWT of the input string y, $\mathit{MAW}(y)$ can be computed in $O(n + |\mathit{MAW}(y)|)$ time. However, the construction time for the representation of the bidirectional BWT is not given in [2].

Our result can also be applied to recent work by Crochemore et al. [6] for string comparison with minimal absent words, resulting in a more efficient algorithm for string comparison with minimal absent words for integer alphabets.

2 Preliminaries

2.1 Strings

Let Σ denote the alphabet. An element of Σ^* is called a *string*. Let ε denote the empty string, and let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. For any string y, we denote its length by $|y|$. For any $1 \leq i \leq |y|$, we use $y[i]$ to denote the ith character of y. If $y = uvw$ with $u, v, w \in \Sigma^*$, then u, v, and w are said to be a *prefix*, *substring*, and *suffix* of y, respectively. For any $1 \leq i \leq j \leq |y|$, $y[i..j]$ denotes the substring of y which begins at position i and ends at position j. For convenience, let $y[i..j] = \varepsilon$ if $i > j$. Let $\mathit{Substr}(y)$ and $\mathit{Suffix}(y)$ denote the set of all substrings and that of all suffixes of y, respectively.

Throughout this paper, we will use y to denote the input string. For any string $x \in \Sigma^*$, we define $\mathit{BegPos}(x) = \{i \mid i \in [1, |y| - |x| + 1], y[i..i + |x| - 1] = x\}$, $\mathit{EndPos}(x) = \{i \mid i \in [|x|, |y|], y[i - |x| + 1..i] = x\}$, i.e., the set of beginning and end positions of occurrences of x in y. For any strings u, v, we write $u \equiv_L v$ (resp. $u \equiv_R v$) when $\mathit{BegPos}(u) = \mathit{BegPos}(v)$ (resp. $\mathit{EndPos}(u) = \mathit{EndPos}(v)$). For any string $x \in \Sigma^*$, the equivalence classes with respect to $\equiv_L$ and $\equiv_R$ that x belongs to, are respectively denoted by $[x]_L$ and $[x]_R$. Also, $\overrightarrow{x}$ and $\overleftarrow{x}$ respectively denote the longest elements of $[x]_L$ and $[x]_R$.

For any set S of strings where no two strings are of the same length, let $\mathrm{long}(S) = \arg\max\{|x| \mid x \in S\}$ and $\mathrm{short}(S) = \arg\min\{|x| \mid x \in S\}$.

In this paper, we assume that the input string y of length n is over the integer alphabet $[1, n^c]$ for some constant c, and that the last character of y is a unique character denoted by \$ that does not occur elsewhere in y. Our model of computation is a standard word RAM of machine word size $\log_2 n$. Space complexities will be evaluated by the number of words (not bits).

2.2 Suffix trees and DAWGs

Suffix trees [18] and directed acyclic word graphs (*DAWGs*) [3] are fundamental text data structures. Both of these data structures are based on suffix tries. The *suffix trie* for string y, denoted $\mathit{STrie}(y)$, is a trie representing $\mathit{Substr}(y)$, formally defined as follows.

▶ **Definition 1.** $\mathit{STrie}(y)$ for string y is an edge-labeled rooted tree (V_T, E_T) such that

$$\begin{aligned} V_T &= \{x \mid x \in \mathit{Substr}(y)\} \\ E_T &= \{(x, b, xb) \mid x, xb \in V_T, b \in \Sigma\}. \end{aligned}$$

The second element b of each edge (x, b, xb) is the label of the edge. We also define the set L_T of labeled "reversed" edges called the *suffix links* of $\mathit{STrie}(y)$ by

$$L_T = \{(ax, a, x) \mid x, ax \in \mathit{Substr}(y), a \in \Sigma\}.$$

As can be seen in the above definition, each node v of $\mathit{STrie}(y)$ can be identified with the substring of y that is represented by v. Assuming that string y terminates with a unique character that appears nowhere else in y, for each suffix $y[i..|y|] \in \mathit{Suffix}(y)$ there is a unique leaf ℓ_i in $\mathit{STrie}(y)$ such that the suffix $y[i..|y|]$ is spelled out by the path from the root to ℓ_i.

It is well known that $\mathit{STrie}(y)$ requires $O(n^2)$ space. One idea to reduce its space to $O(n)$ is to contract each path consisting only of non-branching edges into a single edge labeled with a non-empty string. This leads to the suffix tree $\mathit{STree}(y)$ of string y. Following conventions from [4, 9], $\mathit{STree}(y)$ is defined as follows.

▶ **Definition 2.** $\mathit{STree}(y)$ for string y is an edge-labeled rooted tree (V_S, E_S) such that

$$\begin{aligned} V_S &= \{\overrightarrow{x} \mid x \in \mathit{Substr}(y)\} \\ E_S &= \{(x, \beta, x\beta) \mid x, x\beta \in V_S, \beta \in \Sigma^+, b = \beta[1], \overrightarrow{xb} = x\beta\}. \end{aligned}$$

The second element β of each edge $(x, \beta, x\beta)$ is the label of the edge. We also define the set L_S of labeled "reversed" edges called the *suffix links* of $\mathit{STree}(y)$ by

$$L_S = \{(ax, a, x) \mid x, ax \in V_S, a \in \Sigma\},$$

and denote the tree (V_S, L_S) of the suffix links by $\mathit{SLT}(y)$.

Observe that each internal node of $\mathit{STree}(y)$ is a branching internal node in $\mathit{STrie}(y)$. Note that for any $x \in \mathit{Substr}(y)$ the leaves in the subtree rooted at $\overrightarrow{x}$ correspond to $\mathit{BegPos}(x)$. By representing each edge label β with a pair of integers (i, j) such that $y[i..j] = \beta$, $\mathit{STree}(y)$ can be represented with $O(n)$ space.

An alternative way to reduce the size of $\mathit{STrie}(y)$ to $O(n)$ is to regard $\mathit{STrie}(y)$ as a partial DFA which recognizes $\mathit{Suffix}(y)$, and to minimize it. This leads to the directed acyclic word graph $\mathit{DAWG}(y)$ of string y. Following conventions from [4, 9], $\mathit{DAWG}(y)$ is defined as follows.

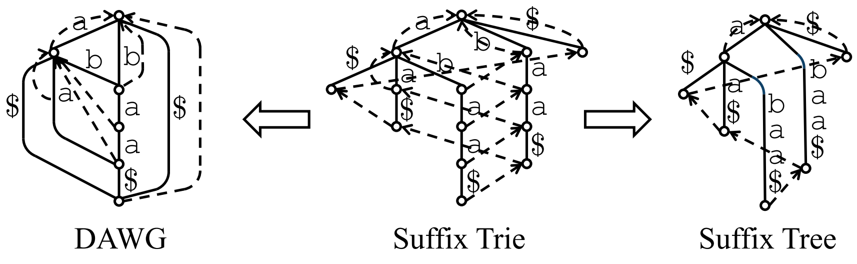


Figure 1 $STrie(y)$, $STree(y)$, and $DAWG(y)$ for string $y = \mathtt{abaa\$}$. The solid arcs represent edges, and the broken arcs represent suffix links.

▶ **Definition 3.** $DAWG(y)$ of string y is an edge-labeled DAG (V_D, E_D) such that

$$\begin{aligned} V_D &= \{[x]_R \mid x \in Substr(y)\} \\ E_D &= \{([x]_R, b, [xb]_R) \mid x, xb \in Substr(y), b \in \Sigma\}. \end{aligned}$$

We also define the set L_D of labeled "reversed" edges called the *suffix links* of $DAWG(y)$ by

$$L_D = \{([ax]_R, a, [x]_R) \mid x, ax \in Substr(y), a \in \Sigma, [ax]_R \neq [x]_R\}.$$

See Figure 1 for examples of $STrie(y)$, $STree(y)$, and $DAWG(y)$.

▶ **Theorem 4** ([3]). *For any string y of length $n > 2$, the number of nodes in $DAWG(y)$ is at most $2n - 1$ and the number of edges in $DAWG(y)$ is at most $3n - 4$.*

Minimization of $STrie(y)$ to $DAWG(y)$ can be done by merging isomorphic subtrees of $STrie(y)$ which are rooted at nodes connected by a chain of suffix links of $STrie(y)$. Since the substrings represented by these merged nodes end at the same positions in y, each node of $DAWG(y)$ forms an equivalence class $[x]_R$. We will make an extensive use of this property in our $O(n)$-time construction algorithm for $DAWG(y)$ over an integer alphabet.

2.3 Minimal Absent Words

A string x is said to be an *absent word* of another string y if $x \notin Substr(y)$. An absent word x of y is said to be a *minimal absent word* (*MAW*) of y if $Substr(x) \setminus \{x\} \subset Substr(y)$. The set of all MAWs of y is denoted by $MAW(y)$. For example, if $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $y = \mathtt{abaab}$, then $MAW(y) = \{\mathtt{aaa}, \mathtt{aaba}, \mathtt{bab}, \mathtt{bb}, \mathtt{c}\}$.

▶ **Lemma 5** ([13]). *For any string $y \in \Sigma^*$, $\sigma \leq |MAW(y)| \leq (\sigma_y - 1)(|y| - 1) + \sigma$, where $\sigma = |\Sigma|$ and σ_y is the number of distinct characters occurring in y. This bound is tight.*

The next lemma follows from the definition of MAWs.

▶ **Lemma 6.** *Let y be any string. For any characters $a, b \in \Sigma$ and string $x \in \Sigma^*$, $axb \in MAW(y)$ iff $axb \notin Substr(y)$, $ax \in Substr(y)$, and $xb \in Substr(y)$.*

By Lemma 6, we can encode each MAW axb of y in $O(1)$ space by (i, j, b), where $ax = y[i..j]$.

3 Constructing DAWGs in $O(n)$ Time for Integer Alphabet

In this section, we present an optimal $O(n)$-time algorithm to construct $DAWG(y)$ with suffix links L_D for a given string y of length n over an integer alphabet. Our algorithm constructs $DAWG(y)$ with suffix links L_D from $STree(y)$ with suffix links L_S. The following result is known.

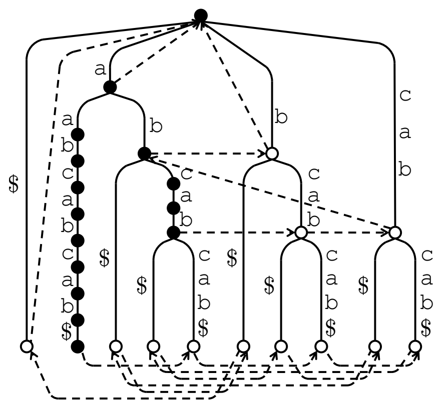


Figure 2 An example of $STree'(y)$ with string $y =$ aabcabcab\$.

▶ **Theorem 7** ([8]). *Given a string y of length n over an integer alphabet, edge-sorted $STree(y)$ with suffix links L_S can be computed in $O(n)$ time.*

Let $\mathcal{L}$ and $\mathcal{R}$ be, respectively, the sets of longest elements of all equivalence classes on y w.r.t. $\equiv_L$ and $\equiv_R$, namely, $\mathcal{L} = \{\overrightarrow{x} \mid x \in Substr(y)\}$ and $\mathcal{R} = \{\overleftarrow{x} \mid x \in Substr(y)\}$. Let $STree'(y) = (V'_S, E'_S)$ be the edge-labeled rooted tree obtained by adding extra nodes for strings in $\mathcal{R}$ to $STree(y)$, namely,

$$\begin{aligned} V'_S &= \{x \mid x \in \mathcal{L} \cup \mathcal{R}\}, \\ E'_S &= \{(x, \beta, x\beta) \mid x, x\beta \in V'_S, \beta \in \Sigma^+, \\ &\quad 1 \leq \forall i < |\beta|, x \cdot \beta[1..i] \notin V'_S\}. \end{aligned}$$

Notice that the size of $STree'(y)$ is $O(n)$, since $|\mathcal{L} \cup \mathcal{R}| \leq |V_S| + |V_D| = O(n)$, where V_S and V_D are respectively the sets of nodes of $STree(y)$ and $DAWG(y)$.

A node $x \in V'_S$ of $STree'(y)$ is called *black* iff $x \in \mathcal{R}$. See Figure 2 for an example of $STree'(y)$.

▶ **Lemma 8.** *For any $x \in Substr(y)$, if x is represented by a black node in $STree'(y)$, then every prefix of x is also represented by a black node in $STree'(y)$.*

Proof. Since x is a black node, $x = \overleftarrow{x}$. Assume on the contrary that there is a proper prefix z of x such that z is not represented by a black node. Let $zu = x$ with $u \in \Sigma^+$. Since $z \equiv_R \overleftarrow{z}$, we have $x = zu \equiv_R \overleftarrow{z}u$. On the other hand, since z is not black, we have $|\overleftarrow{z}| > |z|$. However, this contradicts that x is the longest member $\overleftarrow{x}$ of $[x]_R$. Thus, every prefix of x is also represented by a black node. ◀

▶ **Lemma 9.** *For any string y, let $BT(y)$ be the trie consisting only of the black nodes of $STree'(y)$. Then, every leaf ℓ of $BT(y)$ is a node of the original suffix tree $STree(y)$.*

Proof. Assume on the contrary that some leaf ℓ of $BT(y)$ corresponds to an internal node of $STree'(y)$ that has exactly one child. Since any substring in $\mathcal{L}$ is represented by a node of the original suffix tree $STree(y)$, we have $\ell \in \mathcal{R}$. Since $\ell = \overleftarrow{\ell}$, ℓ is the longest substring of y which has ending positions $EndPos(\ell)$ in y. This implies one of the following situations: (1) occurrences of ℓ in y are immediately preceded by at least two distinct characters $a \neq b$, (2) ℓ occurs as a prefix of y and all the other occurrences of ℓ in y are immediately preceded by a unique character a, or (3) ℓ occurs exactly once in y as its prefix. Let u be the only

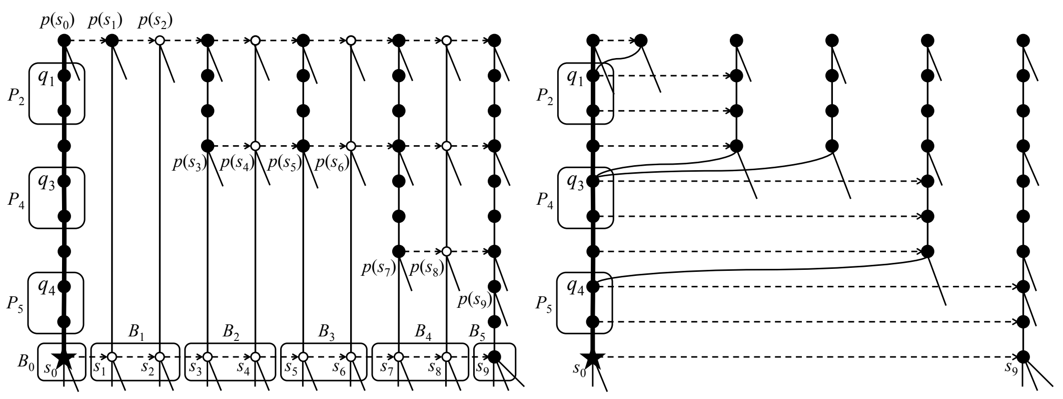


Figure 3 (Left): Illustration for a part of $STree'(y)$, where the branching nodes are those that exist also in the original suffix tree $STree(y)$. Suppose we have just visited node $x = s_0$ (marked by a star) in the post-order traversal on $STree'(y)$. Here, $s_0, \ldots, s_9$ are connected by a chain of the suffix links starting from s_0, and s_9 is the first black node after s_0 in the chain. In the corresponding DAG D, we will add in-coming edges to the black nodes in the path from $p(x)$ to x, and will add suffix links from these black nodes in the path. The sequence $s_0, \ldots, s_m$ of nodes in $STree'(y)$ is partitioned into blocks, such that that the parents of the nodes in the same block belong to the same equivalence class w.r.t. $\equiv_R$. (Right): The in-coming edges and the suffix links have been added to the nodes in the path from $p(x)$ to $x = s_0$.

child of ℓ in $STree'(y)$, and let $\ell z = u$, where $z \in \Sigma^+$. By the definition of ℓ, u is not black. On the other hand, in any of the situations (1)-(3), $u = \ell z$ is the longest substring of y which has ending positions $EndPos(u)$ in y. Hence we have $u = \overleftarrow{u}$ and u must be black, a contradiction. Thus, every leaf ℓ of $BT(y)$ is a node of the original suffix tree $STree(y)$. ◀

▶ **Lemma 10** ([14]). *For any node $x \in V_S$ of the original suffix tree $STree(y)$, its corresponding node in $STree'(y)$ is black iff (1) x is a leaf of the suffix link tree $SLT(y)$, or (2) x is an internal node of $SLT(y)$ and for any character $a \in \Sigma$ such that $ax \in V_S$, $|BegPos(ax)| \neq |BegPos(x)|$.*

Using Lemma 9 and Lemma 10, we can compute all leaves of $BT(y)$ in $O(n)$ time by a standard traversal on the suffix link tree $SLT(y)$. Then, we can compute all internal black nodes of $BT(y)$ in $O(n)$ time using Lemma 8. Now, by Theorem 7, the next lemma holds:

▶ **Lemma 11.** *Given a string y of length n over an integer alphabet, edge-sorted $STree'(y)$ can be constructed in $O(n)$ time.*

We construct $DAWG(y)$ with suffix links L_D from $STree'(y)$, as follows. First, we construct a DAG D, which is initially equivalent to the trie $BT(y)$ consisting only of the black nodes of $STree'(y)$. Our algorithm adds edges and suffix links to D, so that the DAG D will finally become $DAWG(y)$. In so doing, we traverse $STree'(y)$ in post-order. For each black node x of $STree'(y)$ visited in the post-order traversal, which is either an internal node or a leaf of the original suffix tree $STree(y)$, we perform the following: Let $p(x)$ be the parent of x in the *original suffix tree* $STree(y)$. It follows from Lemma 8 that every prefix x' of x with $|p(x)| \leq |x'| \leq |x|$ is represented by a black node. For each black node x' in the path from $p(x)$ to x in the DAG D, we compute the in-coming edges to x' and the suffix link of x'.

Let $s_0, \ldots, s_m$ be the sequence of nodes connected by a chain of suffix links starting from $s_0 = x$, such that $|BegPos(s_i)| = |BegPos(s_0)|$ for all $0 \leq i \leq m-1$ and $|BegPos(s_m)| > |BegPos(s_0)|$ (see the left diagram of Figure 3). In other words, s_m is the first black node after s_0 in the chain of suffix links (this is true by Lemma 10). Since $|s_i| = |s_{i-1}| + 1$ for every $1 \leq i \leq m-1$, $\mathit{EndPos}(s_i) = \mathit{EndPos}(s_0)$. Thus, $s_0, \ldots, s_{m-1}$ form a single equivalence class w.r.t. $\equiv_R$ and are represented by the same node as $x = s_0$ in the DAWG.

For any $0 \leq i \leq m-1$, let $d(s_i) = |s_i| - |p(s_i)|$. Observe that the sequence $d(s_0), \ldots, d(s_m)$ is monotonically non-increasing. We partition the sequence $s_0, \ldots, s_m$ of nodes into blocks so that the parents of all nodes in the same block belong to the same equivalence class w.r.t. $\equiv_R$. Let r be the number of such blocks, and for each $0 \leq k \leq r-1$, let $B_k = s_{i_k}, \ldots, s_{i_{k+1}-1}$ be the kth such block. Note that for each block B_k, $p(s_{i_k})$ is the only black node among the parents $p(s_{i_k}), \ldots, p(s_{i_{k+1}-1})$ of the nodes in B_k, since it is the longest one in its equivalence class $[p(s_{i_k})]_R$. Also, every node in the same block has the same value for function d. Thus, for each block B_k, we add a new edge $(p(s_{i_k}), b_k, q_k)$ to the DAG D, where q_k is the (black) ancestor of x such that $|q_k| = |x| - d(s_{i_k}) + 1$, and b_k is the first character of the label of the edge from $p(s_{i_k})$ to s_{i_k} in $\mathit{STree}'(y)$. Notice that this new edge added to D corresponds to the edges between the nodes in the block B_k and their parents in $\mathit{STree}'(y)$. We also add a suffix link $(p(q_k), a, p(s_{i_k}))$ to D, where $a = s_{i_k - 1}[1]$. See also the right diagram of Figure 3.

For each $2 \leq k \leq r-1$, let P_k be the path from q_{k-1} to g_k, where $g_k = p(p(q_k))$ for $2 \leq k \leq r-2$ and $g_{r-1} = x = s_0$. Each P_k is a sub-path of the path from $p(s_0)$ to s_0, and every node in P_k has not been given their suffix link yet. For each node v in P_k, we add the suffix link from v to the ancestor u of s_{i_k} such that $|s_{i_k}| - |u| = |s_0| - |v|$. See also the right diagram of Figure 3.

Repeating the above procedure for all black nodes of $\mathit{STree}'(y)$ that are either internal nodes or leaves of the original suffix tree $\mathit{STree}(y)$ in post order, the DAG D finally becomes $\mathit{DAWG}(y)$ with suffix links L_D. We remark however that the edges of $\mathit{DAWG}(y)$ might not be sorted, since the edges that exist in $\mathit{STree}'(y)$ were firstly inserted to the DAG D. Still, we can easily sort all the edges of $\mathit{DAWG}(y)$ in $O(n)$ total time after they are constructed: First, extract all edges of $\mathit{DAWG}(y)$ by a standard traversal on $\mathit{DAWG}(y)$, which takes $O(n)$ time. Next, radix sort them by their labels, which takes $O(n)$ time because we assumed an integer alphabet of polynomial size in n. Finally, re-insert the edges to their respective nodes in the sorted order.

▶ **Theorem 12.** *Given a string y of length n over an integer alphabet, we can compute edge-sorted $\mathit{DAWG}(y)$ with suffix links L_D in $O(n)$ time and space.*

Proof. The correctness can easily be seen if one recalls that minimizing $\mathit{STrie}(y)$ based on its suffix links produces $\mathit{DAWG}(y)$. The proposed algorithm simulates this minimization using only the subset of the nodes of $\mathit{STrie}(y)$ that exist in $\mathit{STree}'(y)$. The out-edges of each node of $\mathit{DAWG}(y)$ are sorted in lexicographical order as previously described.

We analyze the time complexity of our algorithm. We can compute $\mathit{STree}'(y)$ in $O(n)$ time by Lemma 11. The initial trie for D can easily be computed in $O(n)$ time from $\mathit{STree}'(y)$. Let x be any node visited in the post-order traversal on $\mathit{STree}'(y)$ that is either an internal node or a leaf of the original suffix tree $\mathit{STree}(y)$. The cost of adding the new in-coming edges to the black nodes in the path from $p(x)$ to $x = s_0$ is linear in the number of nodes in the sequence $s_0, \ldots, s_m$ connected by the chain of suffix links starting from $s_0 = x$. Since s_0 and s_m are the only black nodes in the sequence, it follows from Lemma 10 that the chain of suffix links from s_0 to s_m is a non-branching path of the suffix link tree $\mathit{SLT}(y)$. This implies that the suffix links in this chain are used only for node x during the post-order traversal of $\mathit{STree}'(y)$. Since the number of edges in $\mathit{SLT}(y)$ is $O(n)$, the amortized cost of adding each edge to D is constant. Also, the total cost to sort all edges is $O(n)$, as was previously explained. Now let us consider the cost of adding the suffix links from the nodes in each sub-path P_k. For each node v in P_k, the destination node v can be found in constant time by simply climbing up the path from s_{i_k} in the chain of suffix links. Overall, the total time cost to transform the trie for D to $\mathit{DAWG}(y)$ is $O(n)$.

The working space is clearly $O(n)$. ◀

Figure 4 shows an example of DAWG construction by our algorithm.

In some applications such as bidirectional pattern searches, it is preferable that the in-coming suffix links at each node of $\mathit{DAWG}(y)$ are also sorted in lexicographical order, but our algorithm described above does not sort the suffix links. However, we can sort the suffix links in $O(n)$ time by the same technique applied to the edges of $\mathit{DAWG}(y)$.

4 Constructing Affix Trees in $O(n)$ Time for Integer Alphabet

Let y be the input string of length n over an integer alphabet. Recall the sets $\mathcal{L} = \{\overrightarrow{x} \mid x \in \mathit{Substr}(y)\}$ and $\mathcal{R} = \{\overleftarrow{x} \mid x \in \mathit{Substr}(y)\}$ introduced in Section 3. For any set $S \subseteq \Sigma^* \times \Sigma^*$ of ordered pairs of strings, let $S[1] = \{x_1 \mid (x_1, x_2) \in S$ for some $x_2 \in \Sigma^*\}$ and $S[2] = \{x_2 \mid (x_1, x_2) \in S$ for some $x_1 \in \Sigma^*\}$. For any string x, let $\hat{x}$ denote the reversed string of x.

The affix tree [15] of string y, denoted $\mathit{ATree}(y)$, is a *bidirectional* text indexing structure defined as follows:

▶ **Definition 13.** $\mathit{ATree}(y)$ for string y is an edge-labeled DAG $(V_A, E_A) = (V_A, E_A^F \cup E_A^B)$ which has two mutually distinct sets E_A^F, E_A^B of edges such that

$$\begin{aligned}
V_A &= \{(x, \hat{x}) \mid x \in \mathcal{L} \cup \mathcal{R}\},\\
E_A^F &= \{((x, \hat{x}), \beta, (x\beta, \hat{\beta}\hat{x})) \mid x, x\beta \in V_A[1],\\
&\qquad \beta \in \Sigma^+, 1 \leq \forall i < |\beta|, x \cdot \beta[1..i] \notin V_A[1]\},\\
E_A^B &= \{((x, \hat{x}), \hat{\alpha}, (\alpha x, \hat{x}\hat{\alpha})) \mid \hat{x}, \hat{x}\hat{\alpha} \in V_A[2],\\
&\qquad \alpha \in \Sigma^+, 1 \leq \forall i < |\alpha|, \hat{x} \cdot \hat{\alpha}[1..i] \notin V_A[2]\}.
\end{aligned}$$

E_A^F is the set of forward edges labeled by substrings of y, while E_A^B is the set of backward edges labeled by substrings of $\hat{y}$.

▶ **Theorem 14.** *Given a string y of length n over an integer alphabet, we can compute edge-sorted $\mathit{ATree}(y)$ in $O(n)$ time and space.*

Proof. Clearly, there is a one-to-one correspondence between each node $(x, \hat{x}) \in V_A$ of $\mathit{ATree}(y) = (V_A, E_A^F \cup E_A^B)$ and each node $x \in V_S'$ of $\mathit{STree}'(y) = (V_S', E_S')$ of Section 3 (see also Figure 2 and Figure 5). Moreover, there is a one-to-one correspondence between each forward edge $(x, \beta, x\beta) \in E_A^F$ of $\mathit{ATree}(y)$ and each edge $(x, \beta, x\beta) \in E_S'$ of $\mathit{STree}'(y)$. Hence, what remains is to construct the backward edges in E_A^B for $\mathit{ATree}(y)$. A straightforward modification to our DAWG construction algorithm of Section 3 can construct the backward edges of $\mathit{ATree}(y)$; instead of working on the DAG D, we directly add the suffix links to the black nodes of $\mathit{STree}'(y)$ whose suffix links are not defined yet (namely, those that are neither branching nodes nor leaves of the suffix link tree $\mathit{SLT}(y)$). Since the suffix links are reversed edges, by reversing them we obtain the backward edges of $\mathit{ATree}(y)$. The labels of the backward edges can be easily computed in $O(n)$ time by storing in each node the length of the string it represents. Finally, we can sort the forward and backward edges in lexicographical order in overall $O(n)$ time, using the same idea as in Section 3. ◀

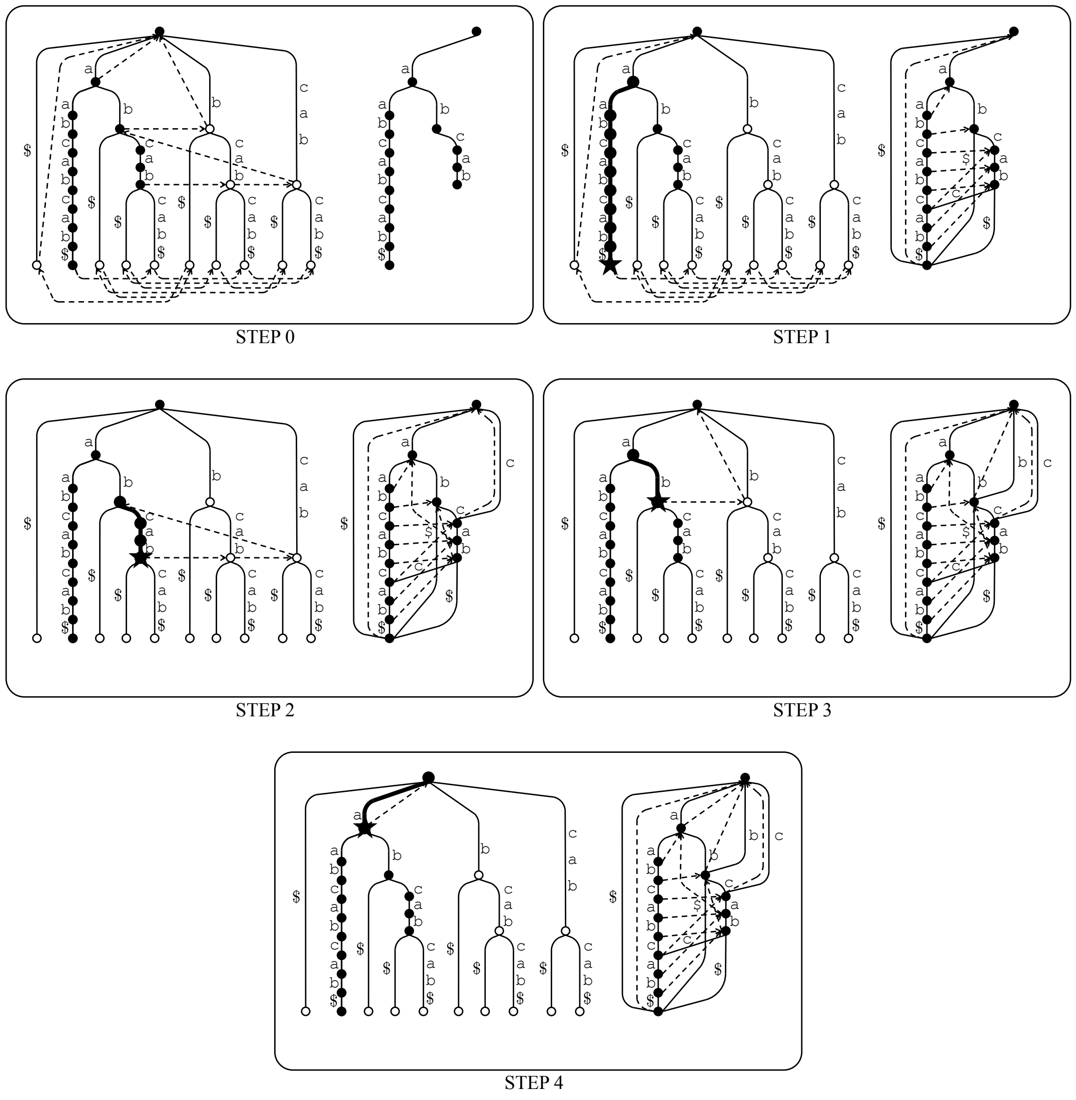


Figure 4 Snapshots during the construction of $DAWG(y)$ for $y = \texttt{aabcabcab\$}$. Step 0: (Left): $STree'(y)$ with suffix links L_S and (Right): the initial trie for D. We traverse $STree'(y)$ in post order. Step 1: We arrived at black leaf node $x_1 = \texttt{aabcabcab\$}$ (indicated by a star). We determine the in-coming edges and suffix links for the black nodes in the path from $p(x_1) = \texttt{a}$ and x_1 (indicated by thick black lines). To the right is the resulting DAG D for this step. Step 2: We arrived at black branching node $x_2 = \texttt{abcab}$ (indicated by a star). We determine the in-coming edges and suffix links for the black nodes in the path from $p(x_2) = \texttt{ab}$ and x_2 (indicated by thick black lines). To the right is the resulting DAG D for this step. Step 3: We arrived at black branching node $x_3 = \texttt{ab}$ (indicated by a star). We determine the in-coming edges and suffix links for the black nodes in the path from $p(x_3) = \texttt{a}$ and x_3 (indicated by thick black lines). To the right is the resulting DAG D for this step. Step 4: We arrived at black branching node $x_4 = \texttt{a}$ (indicated by a star). We determine the in-coming edges and suffix links for the black nodes in the path from $p(x_4) = \varepsilon$ and x_4 (indicated by thick black lines). To the right is the resulting DAG D for this step. Since all branching and leaf black nodes have been processed, the final DAG D is $DAWG(y)$ with suffix links.

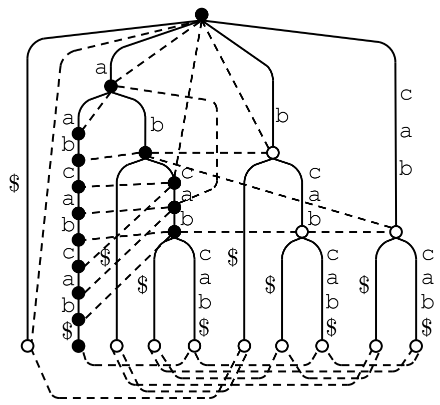


Figure 5 An example of $ATree(y)$ with string $y =$ `aabcabcab$`. The solid arcs represent the forward edges in E_A^F, while the broken arcs represent the backward edges in E_A^B. For simplicity, the labels of backward edges are omitted.

5 Computing Minimal Absent Words in $O(n + |MAW(y)|)$ Time

As an application to our $O(n)$-time DAWG construction algorithm of Section 3, in this section we show an optimal time algorithm to compute the set of all minimal absent words of a given string over an integer alphabet.

Finding minimal absent words of length 1 for a given string y (i.e., the characters not occurring in y) is easy to do in $O(n+\sigma)$ time and $O(1)$ working space for an integer alphabet, where σ is the alphabet size. In what follows, we concentrate on finding minimal absent words of y of length at least 2.

Crochemore et al. [7] proposed a $\Theta(\sigma n)$-time algorithm to compute $MAW(y)$ for a given string y of length n. The following two lemmas, which show tight connections between $DAWG(y)$ and $MAW(y)$, are implicitly used in their algorithm but under a somewhat different formulation. Since our $O(n + |MAW(y)|)$-time solution is built on the lemmas, we give a proof for completeness.

▶ **Lemma 15.** *Let $a, b \in \Sigma$ and $x \in \Sigma^*$. If $axb \in MAW(y)$, then $x = \overleftarrow{x}$, namely, x is the longest string represented by node $[x]_R \in V_D$ of $DAWG(y)$.*

Proof. Assume on the contrary that $x \neq \overleftarrow{x}$. Since x is not the longest string of $[x]_R$, there exists a character $c \in \Sigma$ such that $cx \in Substr(y)$ and $[x]_R = [cx]_R$. Since $axb \in MAW(y)$, it follows from Lemma 6 that $xb \in Substr(y)$. Since $[x]_R = [cx]_R$, c always immediately precedes x in y. Thus we have $cxb \in Substr(y)$.

Since $axb \in MAW(y)$, $c \neq a$. On the other hand, it follows from Lemma 6 that $ax \in Substr(y)$. However, this contradicts that c always immediately precedes x in y and $c \neq a$. Consequently, if $axb \in MAW(y)$, then $x = \overleftarrow{x}$. ◀

For any node $v \in V_D$ of $DAWG(y)$ and character $b \in \Sigma$, we write $\delta_D(v, b) = u$ if $(v, b, u) \in E_D$ for some $u \in V_D$, and write $\delta_D(v, b) = nil$ otherwise. For any suffix link $(u, a, v) \in L_D$ of $DAWG(y)$, we write $sl_D(u) = v$. Since there is exactly one suffix link coming out from each node $u \in V_D$ of $DAWG(y)$, the character a is unique for each node u.

▶ **Lemma 16.** *Let $a, b \in \Sigma$ and $x \in \Sigma^*$. Then, $axb \in MAW(y)$ iff $x = \overleftarrow{x}$, $\delta_D([x]_R, b) = [xb]_R$, $sl_D([ax]_R) = [x]_R$, and $\delta_D([ax]_R, b) = nil$.*

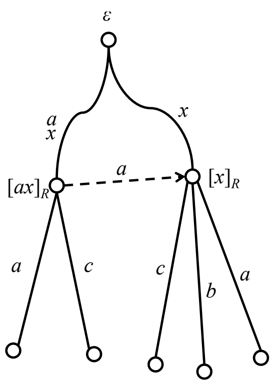


Figure 6 Computing minimal absent words from a DAWG. In this case, axb is a MAW since it does not occur in the string while ax and xb do.

Algorithm 1: $\Theta(n\sigma)$-time algorithm (MF-TRIE) by Crochemore et al. [7]

```
Input: String $y$ of length $n$
Output: All minimal absent words for $y$
1 $MAW \leftarrow \emptyset$;
2 Construct $DAWG(y)$ augmented with suffix links $L_D$;
3 for each non-source node $u$ of $DAWG(y)$ do
4     for each character $b \in \Sigma$ do
5         if $\delta_D(u, b) = nil$ and $\delta_D(sl_D(u), b) \neq nil$ then
6             $MAW \leftarrow MAW \cup \{axb\}$ ;        // $(u, a, sl_D(u)) \in L_D$, $x = \text{long}(sl_D(u))$
7 Output $MAW$;
```

Proof. ($\Rightarrow$) From Lemma 15, $x = \overleftarrow{x}$. From Lemma 6, $axb \notin Substr(y)$. However, $ax, xb \in Substr(y)$, and thus we have $\delta_D([ax]_R, b) = nil$, $\delta_D([x]_R, b) = [xb]_R$, and $sl_D([ax]_R) = [x]_R$, where the last suffix link exists since $x = \overleftarrow{x}$.

($\Leftarrow$) Since $\delta_D([x]_R, b) = [xb]_R$ and $sl_D([ax]_R) = [x]_R$, we have that $xb, ax \in Substr(y)$. Since $ax \in Substr(y)$ and $\delta_D([ax]_R, b) = nil$, we have that $axb \notin Substr(y)$ Thus from Lemma 6, $axb \in MAW(y)$. ◀

From Lemma 16 all MAWs of y can be computed by traversing all the states of $DAWG(y)$ and comparing all out-going edges between nodes connected by suffix links. A pseudo-code of the algorithm MF-TRIE by Crochemore et al. [7], which is based on this idea, is shown in Algorithm 1. Since all characters in the alphabet Σ are tested at each node, the total time complexity becomes $\Theta(n\sigma)$. The working space is $O(n)$, since only the DAWG and its suffix links are needed.

Next we show that with a simple modification in the for loops of the algorithm and with a careful examination of the total cost, the set $MAW(y)$ of all MAWs of the input string y can be computed in $O(n + |MAW(y)|)$ time and $O(n)$ working space. Basically, the only change is to move the "$\delta_D(sl_D(u), b) \neq nil$" condition in Line 5 to the for loop of Line 4. Namely, when we focus on node u of $DAWG(y)$, we test only the characters which label the out-edges from node $sl_D(u)$. A pseudo-code of the modified version is shown in Algorithm 2.

▶ **Theorem 17.** *Given a string y of length n over an integer alphabet, we compute $MAW(y)$ in optimal $O(n + |MAW(y)|)$ time with $O(n)$ working space.*

Algorithm 2: Proposed $O(n + |MAW(y)|)$-time algorithm

```
Input: String $y$ of length $n$
Output: All minimal absent words for $y$
1 $MAW \leftarrow \emptyset$;
2 Construct edge-sorted $DAWG(y)$ augmented with suffix links $L_D$;
3 for each non-source node $u$ of $DAWG(y)$ do
4     for each character $b$ such that $\delta_D(sl_D(u), b) \neq nil$ do
5         if $\delta_D(u, b) = nil$ then
6             $MAW \leftarrow MAW \cup \{axb\}$ ;          // $(u, a, sl_D(u)) \in L_D$, $x = \mathrm{long}(sl_D(u))$
7 Output $MAW$;
```

Proof. First, we show the correctness of our algorithm. For any node u of $DAWG(y)$, $EndPos(sl_D(u)) \supset EndPos(u)$ holds since every string in $sl_D(u)$ is a suffix of the strings in u. Thus, if there is an out-edge of u labeled c, then there is an out-edge of $sl_D(u)$ labeled c. Hence, the task is to find every character b such that there is an out-edge of $sl_D(u)$ labeled b but there is no out-edge of u labeled b. The for loop of Line 4 of Algorithm 2 tests all such characters and only those. Hence, Algorithm 2 computes $MAW(y)$ correctly.

Second, we analyze the efficiency of our algorithm. As was mentioned above, minimal absent words of length 1 for y can be found in $O(n + \sigma)$ time and $O(1)$ working space. By Lemma 5, $\sigma \leq |MAW(y)|$ and hence the σ-term is dominated by the output size $|MAW(y)|$. Now we consider the cost of finding minimal absent words of length at least 2 by Algorithm 2. Let b be any character such that there is an out-edge e of $sl_D(u)$ labeled b. There are two cases: (1) If there is no out-edge of u labeled b, then we output an MAW, so we can charge the cost to check e to an output. (2) If there is an out-edge e' of u labeled b, then the trick is that we can charge the cost to check e to e'. Since each node u has exactly one suffix link going out from it, each out-edge of u is charged only once in Case (2). Since the out-edges of every node u and those of $sl_D(u)$ are both sorted, we can compute their difference for every node u in $DAWG(y)$, in overall $O(n)$ time. Edge-sorted $DAWG(y)$ with suffix links can be constructed in $O(n)$ time for an integer alphabet as in Section 3. Overall, Algorithm 2 runs in $O(n + |MAW(y)|)$ time. The space requirement is clearly $O(n)$. ◀

References

1 Golnaz Badkobeh, Maxime Crochemore, and Chalita Toopsuwan. Computing the maximal-exponent repeats of an overlap-free string in linear time. In *SPIRE 2012*, pages 61–72, 2012.

2 Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Versatile succinct representations of the bidirectional burrows-wheeler transform. In *Proc. ESA 2013*, pages 133–144, 2013.

3 Anselm Blumer, J. Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel I. Seiferas. The smallest automaton recognizing the subwords of a text. *Theor. Comput. Sci.*, 40:31–55, 1985. `doi:10.1016/0304-3975(85)90157-4`.

4 Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987. `doi:10.1145/28869.28873`.

5 Maxime Crochemore. Transducers and repetitions. *Theor. Comput. Sci.*, 45(1):63–86, 1986.

6 Maxime Crochemore, Gabriele Fici, Robert Mercas, and Solon P. Pissis. Linear-time sequence comparison using minimal absent words & applications. In *LATIN 2016*, pages 334–346, 2016.

7 Maxime Crochemore, Filippo Mignosi, and Antonio Restivo. Automata and forbidden words. *Inf. Process. Lett.*, 67(3):111–117, 1998. doi:10.1016/S0020-0190(98)00104-5.

8 Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000.

9 Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi. On-line construction of compact directed acyclic word graphs. *Discrete Applied Mathematics*, 146(2):156–179, 2005.

10 Moritz G. Maaß. Linear bidirectional on-line construction of affix trees. *Algorithmica*, 37(1):43–74, 2003.

11 Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.

12 Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976. doi:10.1145/321941.321946.

13 Filippo Mignosi, Antonio Restivo, and Marinella Sciortino. Words and forbidden factors. *Theor. Comput. Sci.*, 273(1-2):99–117, 2002.

14 Kazuyuki Narisawa, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Efficient computation of substring equivalence classes with suffix arrays. In *CPM 2007*, pages 340–351, 2007.

15 Jens Stoye. Affix trees. Technical Report Report 2000-04, Universität Bielefeld, 2000. URL: https://www.techfak.uni-bielefeld.de/~stoye/dropbox/report00-04.pdf.

16 Shiho Sugimoto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Finding absent words from grammar compressed strings. In the Festschrift for Bořivoj Melichar, 2012.

17 Yuka Tanimura, Yuta Fujishige, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. A faster algorithm for computing maximal α-gapped repeats in a string. In *SPIRE 2015*, pages 124–136, 2015.

18 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

19 Jun-ichi Yamamoto, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. Faster compact on-line Lempel-Ziv factorization. In *STACS 2014*, pages 675–686, 2014.

20 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, 1977.

On Planar Valued CSPs[*]

Peter Fulla[1] and Stanislav Živný[2]

1 **Department of Computer Science, University of Oxford, UK**
`peter.fulla@cs.ox.ac.uk`
2 **Department of Computer Science, University of Oxford, UK**
`standa.zivny@cs.ox.ac.uk`

Abstract

We study the computational complexity of planar valued constraint satisfaction problems (VCSPs). First, we show that intractable *Boolean* VCSPs have to be *self-complementary* to be tractable in the planar setting, thus extending a corresponding result of Dvořák and Kupec [ICALP'15] from CSPs to VCSPs. Second, we give a complete complexity classification of *conservative* planar VCSPs on arbitrary finite domains. As it turns out, in this case planarity does not lead to any new tractable cases, and thus our classification is a sharpening of the classification of conservative VCSPs by Kolmogorov and Živný [JACM'13].

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases constraint satisfaction, valued constraint satisfaction, planarity, polymorphisms, multimorphisms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.39

1 Introduction

The valued constraint satisfaction problem (VCSP) is a far-reaching generalisation of many natural satisfiability, colouring, minimum-cost homomorphism, and min-cut problems [18, 21]. It is naturally parametrised by its domain and a valued constraint language. A *domain* D is an arbitrary finite set. A *valued constraint language*, or just a language, Γ is a (usually finite) set of weighted relations; each weighted relation $\gamma \in \Gamma$ is a function $\gamma : D^{\mathrm{ar}(\gamma)} \to \overline{\mathbb{Q}}$, where $\mathrm{ar}(\gamma) \in \mathbb{N}^{+}$ is the *arity* of γ and $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ is the set of extended rationals.

An *instance* $I = (V, D, C)$ of the VCSP on domain D is given by a finite set of n variables $V = \{x_1, \dots, x_n\}$ and an objective function $C : D^n \to \overline{\mathbb{Q}}$ expressed as a weighted sum of *valued constraints* over V, i.e. $C(x_1, \dots, x_n) = \sum_{i=1}^{q} w_i \cdot \gamma_i(\mathbf{x}_i)$, where γ_i is a weighted relation, $w_i \in \mathbb{Q}_{\geq 0}$ is the *weight* and $\mathbf{x}_i \in V^{\mathrm{ar}(\gamma_i)}$ the *scope* of the ith valued constraint. Given an instance I, the goal is to find an assignment $s : V \to D$ of domain labels to the variables that *minimises* C. Given a language Γ, we denote by VCSP(Γ) the class of all instances I that use only weighted relations from Γ in their objective function.

We now provide a few examples of languages on $D = \{0, 1\}$. If $\Gamma_{\mathsf{nae}} = \{\rho\}$ with $\rho(x, y, z) = \infty$ if $x = y = z$ and $\rho(x, y, z) = 0$ otherwise, then VCSP(Γ_{nae}) captures precisely the NAE-3-Sat (Not-All-Equal 3-Satisfiability) problem. If $\Gamma_{\mathsf{cut}} = \{\gamma\}$ with $\gamma(x, y) = 1$ if $x = y$ and $\gamma(x, y) = 0$ otherwise, then VCSP(Γ_{cut}) captures precisely the Min-UnCut problem. If $\Gamma_{\mathsf{is}} = \{\rho, \gamma\}$ with $\rho(x, y) = \infty$ if $x = y = 1$ and $\rho(x, y) = 0$ otherwise, and $\gamma(x) = 1 - x$, then VCSP(Γ_{is}) captures precisely the Maximum Independent Set problem.

[*] The authors were supported by a Royal Society Research Grant. The work was done while the authors were visiting the Simons Institute for the Theory of Computing at UC Berkeley. Stanislav Živný was supported by a Royal Society University Research Fellowship.

Minimisation of bounded-arity submodular functions (or equivalently, submodular pseudo-Boolean polynomials of bounded degree) corresponds to VCSP(Γ_{sub}) for Γ_{sub} consisting of all weighted relations γ that satisfy $\gamma(\min(\mathbf{x}, \mathbf{y})) + \gamma(\max(\mathbf{x}, \mathbf{y})) \leq \gamma(\mathbf{x}) + \gamma(\mathbf{y})$, where min and max are applied componentwise.

We will be concerned with *exact* solvability of VCSPs. A language Γ is called *tractable* if VCSP(Γ') can be solved (to optimality) in polynomial time for every finite subset $\Gamma' \subseteq \Gamma$, and Γ is called *intractable* if VCSP(Γ') is NP-hard for some finite $\Gamma' \subseteq \Gamma$. For instance, Γ_{sub} is tractable [8] whereas Γ_{nae}, Γ_{cut}, Γ_{is} are intractable [15].

1.1 Contribution

Languages on a two-element domain are called *Boolean.* The complexity of Boolean valued constraint languages is well understood and eight tractable cases have been identified [8]. Suppose that a Boolean language Γ is intractable. We are interested in restrictions that can be imposed on input instances of VCSP(Γ) that make the problem tractable. A natural way is to restrict the *incidence graph* of the instance (the precise definition is given in Section 2). In this paper we initiate the study of the *planar* variant of the VCSP.

We denote by $\mathrm{VCSP}_{\mathrm{p}}(\Gamma)$ the class of instances I of VCSP(Γ) with planar incidence graph (with an additional requirement that leads to a finer classification, as discussed in detail in Section 2). Language Γ is called *planarly-tractable* if $\mathrm{VCSP}_{\mathrm{p}}(\Gamma')$ can be solved (to optimality) in polynomial time for every finite subset $\Gamma' \subseteq \Gamma$, and it is called *planarly-intractable* if $\mathrm{VCSP}_{\mathrm{p}}(\Gamma')$ is NP-hard for some finite $\Gamma' \subseteq \Gamma$. For instance, while Γ_{nae}, Γ_{cut}, and Γ_{is} are intractable, it is known that Γ_{nae} and Γ_{cut} are planarly-tractable [28, 17] whereas Γ_{is} is planarly-intractable [14]. The problem of classifying all intractable languages as planarly-tractable and planarly-intractable is challenging and open even for Boolean valued constraint languages.

A Boolean valued constraint language Γ is called *self-complementary* if every $\gamma \in \Gamma$ satisfies $\gamma(\mathbf{x}) = \gamma(\overline{\mathbf{x}})$ for every $\mathbf{x} \in D^{\mathrm{ar}(\gamma)}$, where $\overline{\mathbf{x}} = (1 - x_1, \ldots, 1 - x_{\mathrm{ar}(\gamma)})$ for $\mathbf{x} = (x_1, \ldots, x_{\mathrm{ar}(\gamma)})$. As our first contribution, we show in Section 3 that intractable Boolean valued constraint languages that are *not* self-complementary are planarly-intractable. We prove this by carefully constructing planar NP-hardness gadgets for any intractable Boolean valued constraint language that is not self-complementary, relying on the fact that all tractable Boolean valued constraint languages are known [8]. Our result subsumes the analogous result obtained for $\{0, \infty\}$-valued languages [10]. We remark that focusing on Boolean languages is natural as it avoids a number of difficulties intrinsic to the planar setting. Let $\Gamma_{\mathsf{col}} = \{\gamma\}$ with $\gamma(x, y) = 0$ if $x \neq y$ and $\gamma(x, y) = \infty$ otherwise. Then Γ_{col} on domain D with $|D| = 3$ is planarly intractable (since $\mathrm{VCSP}_{\mathrm{p}}(\Gamma_{\mathsf{col}})$ captures precisely the 3-Colouring problem on planar graphs) [15] but is tractable on D with $|D| = 4$ for highly nontrivial reasons, namely the Four Colour Theorem.

A valued constraint language Γ on D is called *conservative* if Γ contains all $\{0, 1\}$-valued unary weighted relations. The complexity of conservative valued constraint languages is well understood: a complete complexity classification has been obtained in [26], with a recent simplification of both the algorithmic and the hardness part [35]. As our second contribution, we give a complete complexity classification of conservative valued constraint languages on arbitrary finite domains with respect to planar-tractability. In particular, we show that every intractable conservative valued constraint language is also planarly-intractable. Hence there are no new tractable cases in the conservative planar setting. This may seem unsurprising but the proof is not trivial.

Note that for Boolean valued constraint languages that are conservative the claim follows immediately from our first result: any intractable Boolean language containing both $\gamma_0(x) = x$

and $\gamma_1(x) = 1 - x$ (guaranteed by the conservativity assumption) is not self-complementary, and thus is planarly-intractable. This shows that $\Gamma = \Gamma_{\mathsf{cut}} \cup \{\gamma_0, \gamma_1\}$ is intractable, a result originally obtained in [1] since $\mathrm{VCSP_p}(\Gamma)$ captures precisely the planar Min-UnCut problem with unary weights. (In fact, the same argument shows that both $\Gamma_{\mathsf{cut}} \cup \{\gamma_0\}$ and $\Gamma_{\mathsf{cut}} \cup \{\gamma_1\}$ are planarly-intractable.)

As it is common in the world of CSPs, dealing with non-Boolean domains is considerably more difficult than the case of Boolean domains. For valued constraint languages we have a Galois connection with certain algebraic objects [6, 13] but no Galois connection is known for valued constraint languages in the planar setting. Moreover, it is unclear how to use the recent relatively simple proof of the complexity classification of conservative valued constraint languages [35] and make it work in the planar setting since the proof depends on linear programming duality. (This is related to the lack of a Galois connection in the planar setting. In particular, [35, Lemma 2], which relates (non-planar) expressibility and operator Opt, is proved via LP duality, and it is unclear how to prove it in the planar setting.)

Our approach is to follow the original proof of the classification of conservative valued constraint languages [26]. In order to adapt the proof for the planar setting, we significantly simplify it and generalise necessary parts. Details on proof differences as well as challenges that we needed to overcome to make the proof work are outlined in Section 4. We believe that our proof techniques, and in particular the now simplified and generalised technique from [26], will be useful in future work on planar (V)CSPs.

1.2 Related work

VCSPs with $\{0, \infty\}$-valued weighted relations are just (ordinary) decision CSPs [11]. There has been a lot of work on decision CSPs, see [5] for a recent survey. Most results have been obtained for CSPs parametrised by a constraint language, see [2] for a recent survey. Some of the algebraic methods developed for CSPs [3] have been extended to VCSPs [6, 34, 13, 27] and successfully used in classifying various fragments of VCSPs [20, 25, 33, 23, 35]. However, it is unclear how to use algebraic methods for instance-restricted classes of VCSPs (sometimes called *hybrid* [5]), even though there are some recent investigations in this direction [24, 32].

Planar restrictions have been studied for Boolean (decision) CSPs [10], for Boolean symmetric counting CSPs with real [4] and complex [16] weights, and also for Boolean CSPs with respect to polynomial-time approximation schemes [22, 9].

2 Preliminaries

2.1 Planar VCSPs

Let I be a VCSP instance with variables V and valued constraints S. The *incidence graph* of I is the bipartite multigraph with vertex set $S \cup V$ and edges (γ, x_i) for every $\gamma(x_1, \ldots, x_{\mathrm{ar}(\gamma)}) \in S$ and $1 \leq i \leq \mathrm{ar}(\gamma)$.

We are interested in VCSP instances with *planar* incidence graphs. Following [10], we additionally require the order of edges around constraint vertices in the plane drawing of the incidence graph respect the order of arguments of the corresponding constraint. Note that the variant without this additional restriction can be easily modelled by replacing each weighted relation γ in a language by all weighted relations obtained from γ by permuting the order of its inputs. Hence, this choice leads to a finer classification.

Following [10], rather than working with the incidence graph, we equivalently define the problem in terms of a related plane graph where variables correspond to vertices and valued constraints to faces. We note that our graphs are allowed to have loops, possibly several at a single vertex, and parallel edges.

For a connected plane graph G, we denote by $F(G)$ the set of its faces. For any face $f \in F(G)$, let $b(f)$ denote a closed walk bounding f, enumerated in the clockwise order around f.

▶ **Definition 1.** A *plane* VCSP *instance* (I, G, ϕ) is given by a VCSP instance I with variables V and objective function C with q valued constraints, a connected plane graph G over vertices V, and an injective mapping $\phi : \{1, \dots, q\} \to F(G)$ such that for every valued constraint $\gamma_i(x_1, x_2, \dots, x_{\mathrm{ar}(\gamma_i)})$ it holds $b(\phi(i)) = x_1 x_2 \dots x_{\mathrm{ar}(\gamma_i)} x_1$.

We note that the definition of a *planar* VCSP *instance*, in which case the graph G and mapping ϕ are not given, is equivalent to Definition 1. This is because, as mentioned in [10], checking whether a VCSP instance I has a planar representation, and if so then finding (I, G, ϕ), can be done in polynomial time [19]. For simplicity of presentation, we will assume that graph G and mapping ϕ are given.

We denote by $\mathrm{VCSP}_\mathrm{p}(\Gamma)$ the class of plane VCSP instances over the language Γ.

2.2 Planar Weighted Relational Clones

In this section, we define planar weighted relational clones, which are closures of valued constraint languages that do not change the tractability of corresponding planar VCSPs.

Relations can be seen as a special case of weighted relations with range $\{0, \infty\}$ (also called *crisp*). For a weighted relation $\gamma : D^r \to \overline{\mathbb{Q}}$, we denote by $\mathrm{Feas}(\gamma) = \{\mathbf{x} \in D^r \mid \gamma(\mathbf{x}) < \infty\}$ the underlying *feasibility relation*, and by $\mathrm{Opt}(\gamma) = \{\mathbf{x} \in \mathrm{Feas}(\gamma) \mid \gamma(\mathbf{x}) \le \gamma(\mathbf{y})$ for every $\mathbf{y} \in D^r\}$ the relation of minimal-value (or *optimal*) tuples. We also write $\mathrm{Feas}(\gamma) = 0 \cdot \gamma$ and see the Feas operator as scaling a weighted relation by zero, where we define $0 \cdot \infty = \infty$.

An assignment $s : V \to D$ for a VCSP instance (V, D, C) with $V = \{x_1, \dots, x_n\}$ is called *feasible* if $C(s(x_1), \dots, s(x_n)) < \infty$.

▶ **Definition 2.** Let (I, G, ϕ) be a plane VCSP instance such that ϕ does not map any i to the outer face f_o of G, and let $\mathbf{v} = (v_1, \dots, v_r)$ be an r-tuple of variables from V such that $b(f_o) = v_r v_{r-1} \dots v_1 v_r$. We denote by $\pi_\mathbf{v}(I)$ the r-ary weighted relation mapping any $\mathbf{x} \in D^r$ to the minimum objective value obtained by feasible assignments s of I with $s(\mathbf{v}) = \mathbf{x}$, or ∞ if no such feasible assignment exists.

An r-ary weighted relation γ is *planarly expressible* from a valued constraint language Γ if there exists a plane instance I over Γ and an r-tuple $\mathbf{v}$ of its variables such that $\pi_\mathbf{v}(I) = \gamma$.

▶ **Definition 3.** A *planar weighted relational clone* is a non-empty set of weighted relations over the same domain that is closed under planar expressibility, scaling by non-negative rational constants, addition of rational constants, and operator Opt. We will denote the smallest planar weighted relational clone containing a valued constraint language Γ by $\mathrm{wClone}_\mathrm{p}(\Gamma)$.

An analogous notion of weighted relational clones closed under *general* (i.e. not necessarily planar) expressibility [6, 13] has been used to study the complexity of VCSPs.

▶ **Lemma 4.** *For any domain D and language Γ on D, the binary equality relation $\rho_=$ on D belongs to* $\mathrm{wClone}_\mathrm{p}(\Gamma)$.

Proof. Relation $\rho_=$ is planarly expressible by a plane instance consisting of a single variable x with two self-loops, and $\mathbf{v} = (x, x)$. ◀

▶ **Theorem 5.** *For any valued constraint language* Γ, Γ *is planarly-tractable if, and only if,* $\mathrm{wClone_p}(\Gamma)$ *is planarly-tractable, and* Γ *is planarly-intractable if, and only if,* $\mathrm{wClone_p}(\Gamma)$ *is planarly-intractable.*

Proof. We show that $\mathrm{VCSP_p}(\mathrm{wClone_p}(\Gamma))$ is polynomial-time reducible to $\mathrm{VCSP_p}(\Gamma)$. Given an instance I over $\mathrm{wClone_p}(\Gamma)$, we replace in it all weighted relations planarly expressible from Γ by their plane instances. Scaling, which includes Feas, can be achieved by adjusting the weights of the valued constraints. Adding a constant to a weighted relation affects the value of every feasible assignment by the same amount, and therefore can be ignored.

Relation $\mathrm{Opt}(\gamma)$ can be simulated by scaling γ by a sufficiently large constant. Let W equal an upper bound on the maximum objective value of a feasible assignment of I. Without loss of generality, we may assume that no weighted relation of I assigns a negative value and that the smallest value assigned by γ is 0. Let d equal the second smallest value assigned by γ. We replace $\mathrm{Opt}(\gamma)$ with $(W/d+1)\cdot\gamma$, so that any assignment of I that would incur an infinite value from $\mathrm{Opt}(\gamma)$ has now objective value exceeding W. ◀

2.3 Algebraic Properties

For any r-tuple $\mathbf{x} \in D^r$, we write x_i for its ith component. We apply a k-ary operation $f : D^k \to D$ to k r-tuples componentwise; that is, if $\mathbf{x}^1 = (x^1_1,\dots,x^1_r), \mathbf{x}^2 = (x^2_1,\dots,x^2_r),\dots,\mathbf{x}^k = (x^k_1,\dots,x^k_r)$, then

$$f(\mathbf{x}^1,\dots,\mathbf{x}^k) \;=\; (f(x^1_1,x^2_1,\dots,x^k_1), f(x^1_2,x^2_2,\dots,x^k_2),\dots,f(x^1_r,x^2_r,\dots,x^k_r))\,.$$

The following notion is at the heart of the algebraic approach to decision CSPs [3].

▶ **Definition 6.** Let γ be a weighted relation on D. A k-ary operation $f : D^k \to D$ is a *polymorphism* of γ (and γ is *invariant under* or *admits* f) if, for every $\mathbf{x}^1,\dots,\mathbf{x}^k \in \mathrm{Feas}(\gamma)$, we have $f(\mathbf{x}^1,\dots,\mathbf{x}^k) \in \mathrm{Feas}(\gamma)$. We say that f is a polymorphism of a language Γ if it is a polymorphism of every $\gamma \in \Gamma$. We denote by $\mathrm{Pol}(\Gamma)$ the set of all polymorphisms of Γ.

A k-ary *projection* is an operation of the form $\pi^{(k)}_i(x_1,\dots,x_k) = x_i$ for some $1 \le i \le k$. Projections are (trivial) polymorphisms of all valued constraint languages.

The following notion, which involves a collection of k k-ary polymorphisms, played an important role in the complexity classification of Boolean valued constraint languages [8].

▶ **Definition 7.** Let γ be a weighted relation on D. A list $\langle f_1,\dots,f_k\rangle$ of k-ary polymorphisms of γ is a k-ary *multimorphism* of γ (and γ *admits* $\langle f_1,\dots,f_k\rangle$) if, for every $\mathbf{x}^1,\dots,\mathbf{x}^k \in \mathrm{Feas}(\gamma)$, we have

$$\sum_{i=1}^{k} \gamma(f_i(\mathbf{x}^1,\dots,\mathbf{x}^k)) \;\le\; \sum_{i=1}^{k} \gamma(\mathbf{x}^i)\,. \tag{1}$$

We say that $\langle f_1,\dots,f_k\rangle$ is a multimorphism of a language Γ if it is a multimorphism of every $\gamma \in \Gamma$.

It is known that weighted relational clones preserve polymorphisms and multimorphisms [6] and thus planar weighted relational clones do as well.

▶ **Example 8.** The class of submodular functions on $D = \{0,1\}$ [30] can be defined as the valued constraint language Γ_{sub} that admits $\langle \min,\max\rangle$ as a multimorphism; that is, for every $\gamma \in \Gamma_{\mathsf{sub}}$, we have $\gamma(\min(\mathbf{x},\mathbf{y})) + \gamma(\max(\mathbf{x},\mathbf{y})) \le \gamma(\mathbf{x}) + \gamma(\mathbf{y})$.

3 Boolean Valued CSPs

In this section, we will consider only languages on a Boolean domain $D = \{0, 1\}$. Our first result is that self-complementarity is necessary for planar-tractability of intractable Boolean languages.

▶ **Theorem 9.** *Let Γ be a Boolean valued constraint language that is intractable. If Γ is not self-complementary then it is planarly-intractable.*

We start with some notation for important operations on D. For any $a \in D$, c_a is the constant unary operation such that $c_a(x) = a$ for all $x \in D$. Operation $\neg$ is the unary negation, i.e. $\neg(0) = 1$ and $\neg(1) = 0$. Binary operation min (max) is the minimum (maximum) operation with respect to the order $0 < 1$. Ternary operation Mn (Mj) is the unique minority (majority) operation.

Next we define some useful relations. For any $a \in D$, we denote by ρ_a the unary constant relation $\{(a)\}$. Relation $\rho_{\neq}$ is the binary disequality relation, i.e. $\rho_{\neq} = \{(0, 1), (1, 0)\}$. Ternary relation $\rho_{\text{1-in-3}}$ corresponds to the 1-in-3 Positive 3-Sat problem, i.e. $\rho_{\text{1-in-3}} = \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$. Weighted relations $\gamma_0, \gamma_1, \gamma_{\neq}$ are defined as soft-constraint variants of $\rho_0, \rho_1, \rho_{\neq}$ assigning value 0 to allowed tuples and 1 to disallowed tuples.

Note that Γ is self-complementary if, and only if, Γ admits multimorphism $\langle\neg\rangle$. The proof of Theorem 9 is based on the following four lemmas.

▶ **Lemma 10.** *Let Γ be a valued constraint language that admits neither of the multimorphisms $\langle c_0\rangle$, $\langle c_1\rangle$. Then $\rho_0, \rho_1 \in \mathrm{wClone_p}(\Gamma)$ or $\rho_{\neq} \in \mathrm{wClone_p}(\Gamma)$.*

▶ **Lemma 11.** *Let Γ be a valued constraint language that admits neither of the multimorphisms $\langle \min, \min\rangle$, $\langle \max, \max\rangle$, $\langle \min, \max\rangle$. If $\rho_0, \rho_1 \in \mathrm{wClone_p}(\Gamma)$, then $\rho_{\neq} \in \mathrm{wClone_p}(\Gamma)$.*

▶ **Lemma 12.** *Let Γ be a valued constraint language that does not admit multimorphism $\langle\neg\rangle$. If $\rho_{\neq} \in \mathrm{wClone_p}(\Gamma)$, then $\rho_0, \rho_1 \in \mathrm{wClone_p}(\Gamma)$.*

▶ **Lemma 13.** *Let Γ be a valued constraint language that admits neither of the multimorphisms $\langle \mathrm{Mn}, \mathrm{Mn}, \mathrm{Mn}\rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mj}\rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mn}\rangle$. If $\rho_0, \rho_1, \rho_{\neq} \in \mathrm{wClone_p}(\Gamma)$, then $\rho_{\text{1-in-3}} \in \mathrm{wClone_p}(\Gamma)$.*

Proof (of Theorem 9). Since Γ is intractable we know, by [8, Theorem 7.1], that Γ admits neither of the multimorphisms $\langle c_0\rangle$, $\langle c_1\rangle$, $\langle \min, \min\rangle$, $\langle \max, \max\rangle$, $\langle \min, \max\rangle$, $\langle \mathrm{Mn}, \mathrm{Mn}, \mathrm{Mn}\rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mj}\rangle$, $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mn}\rangle$. By assumption, Γ is not self-complementary and hence does not admit the $\langle\neg\rangle$ multimorphism.

By Lemmas 10, 11, and 12, we have $\rho_0, \rho_1, \rho_{\neq} \in \mathrm{wClone_p}(\Gamma)$ and hence by Lemma 13 $\rho_{\text{1-in-3}} \in \mathrm{wClone_p}(\Gamma)$. Planar 1-in-3 Positive 3-Sat problem is NP-complete [29], and therefore Γ is planarly-intractable by Theorem 5. ◀

4 Conservative Valued CSPs

A valued constraint language Γ is called *conservative* if Γ includes all $\{0, 1\}$-valued unary weighted relations. As our second result, we prove that planarity does not add any tractable cases for conservative valued constraint languages.

▶ **Theorem 14.** *Let Γ be an intractable conservative valued constraint language. Then Γ is planarly-intractable.*

Consequently, we obtain a complexity classification of all conservative valued constraint languages in the planar setting, thus sharpening the classification of conservative valued constraint languages [26, 35]. As mentioned in Section 1, for Boolean domains Theorem 14 follows from Theorem 9. Thus, the only tractable Boolean conservative languages in the planar setting are given by the multimorphisms $\langle \min, \max \rangle$ and $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mn} \rangle$ [8].

We now define certain special types of multimorphisms.

A k-ary operation $f : D^k \to D$ if called *conservative* if $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$ for every $x_1, \ldots, x_k \in D$. A multimorphism $\langle f_1, \ldots, f_k \rangle$ is called *conservative* if applying $\langle f_1, \ldots, f_k \rangle$ to $(x_1, \ldots, x_k)$ returns a permutation of $(x_1, \ldots, x_k)$.

▶ **Definition 15.** A binary multimorphism $\langle f, g \rangle$ of Γ is called a *symmetric tournament pair* (STP) if it is conservative and both f and g are commutative operations.

It was shown in [7] that languages admitting an STP multimorphism are tractable.

A ternary operation $f : D^3 \to D$ is called a *majority* operation if $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$ for all $x, y \in D$, and a *minority* operation if $f(x, x, y) = f(x, y, x) = f(y, x, x) = y$ for all $x, y \in D$.

▶ **Definition 16.** A ternary multimorphism $\langle f, g, h \rangle$ is called an MJN if f and g are (possibly equal) majority operations and g is a minority operation.

It was shown in [26] that languages admitting an MJN multimorphism are tractable.

▶ **Theorem 17** ([26]). *Let Γ be a conservative valued constraint language on D. Then either Γ admits a conservative binary multimorphism $\langle f, g \rangle$ and a conservative ternary multimorphism $\langle f', g', h' \rangle$ and there is a family M of 2-element subsets of D, such that*

- *for every $\{a, b\} \in M$, $\langle f, g \rangle$ restricted to $\{a, b\}$ is a symmetric tournament pair, and*
- *for every $\{a, b\} \notin M$, $\langle f', g', h' \rangle$ restricted to $\{a, b\}$ is an MJN multimorphism,*

in which case Γ is tractable, or else Γ is intractable.

The idea of the proof of Theorem 17 is as follows: given a conservative valued constraint language Γ, we define a certain graph G_Γ whose vertices are pairs of different labels from D and an edge $(a, b) - (c, d)$ is present if there is a binary weighted relation $\gamma \in \mathrm{wClone}(\Gamma)$ that is "non-submodular with respect to the order $a < b$ and $c < d$". The edges of G_Γ are then classified as soft and hard. It is shown that a soft self-loop implies intractability of Γ. Otherwise, the vertices of G_Γ are partitioned into $M \cup \overline{M}$, where M denotes the set of loopless vertices and $\overline{M}$ denotes the rest (i.e. vertices with hard loops). It is then shown that G_Γ restricted to M is bipartite, which is in turn used to construct a binary multimorphism and a ternary multimorphism of Γ such that the binary multimorphism is an STP on M and the ternary multimorphism is an MJN on $\overline{M}$. (Proving that the constructed objects are multimorphisms of Γ is the most technical part of the proof.) Any such language is then tractable via an involved algorithm from [26] that relies on [7], or by an LP relaxation [35].

Our approach is to follow the above-described proof and adapt it to the planar setting. It is natural to replace $\mathrm{wClone}(\Gamma)$ by $\mathrm{wClone}_{\mathrm{p}}(\Gamma)$ in the definition of G_Γ. But this simple change does not immediately yield the desired result. There are two main obstacles. First, the proof of Theorem 17 from [26] heavily relies on [31], which guarantees the existence of a majority polymorphism. However, this is proved in [31] using (functional) clones and depends on the Galois connection between clones and relational co-clones; such a connection is not known for planar expressibility! Second, some of the gadgets (and in particular the "i-expansion" from [26, Section 6.4]) are not necessarily planar.

To avoid these obstacles, we modify, significantly simplify, and generalise the proof so that it works in the planar setting. The key changes are the following. (i) We use a closure of a language, denoted Γ^* below, that is a subset of the planar weighted relational clone of a conservative language. (ii) We do *not* rely on Takhanov's result on the existence of a majority polymorphism [31] but instead prove directly without using [31] that (the closure of) Γ is 2-decomposable. (iii) We define different STP and MJN multimorphisms that allow us to simplify the proof that these are indeed multimorphisms of Γ. In particular, we will prove modularity of weighted relations on $\overline{M}$ and show that the ternary multimorphism satisfies Inequality (1) with equality, thus obtaining a better structural understanding of tractable languages. The main simplification is that we define MJN as close to projection operations as possible, and in particular not depending on the STP multimorphism as in [26].

We now define a few operations on weighted relations.

▶ **Definition 18.** Let γ be an r-ary weighted relation on D. A *domain restriction* of γ to $D' \subseteq D$ at coordinate i is the r-ary weighted relation defined as $\gamma'(x_1, \dots, x_r) = \gamma(x_1, \dots, x_r) + \rho_{D'}(x_i)$, where $\rho_{D'}(x) = 0$ if $x \in D'$ and $\rho_{D'}(x) = \infty$ otherwise. A *pinning* of γ to $a \in D$ at coordinate i is the $(r-1)$-ary weighted relation defined as $\gamma'(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_r) = \min_{x_i \in D} \gamma(x_1, \dots, x_r) + \rho_{\{a\}}(x_i)$. A *minimisation* of γ at coordinate i is the $(r-1)$-ary weighted relation defined as $\gamma'(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_r) = \min_{x_i \in D} \gamma(x_1, \dots, x_r)$.

A *join* of two binary weighted relations γ_1 and γ_2 is the weighted relation $\gamma(x, y) = \min_{z \in D} \gamma_1(z, x) + \gamma_2(z, y)$.

We will make use only of a limited subset of $\mathrm{wClone_p}(\Gamma)$, which is defined below.

▶ **Definition 19.** For a conservative valued constraint language Γ on D, we define Γ^* to be the smallest set containing Γ, all unary weighted relations and the binary equality relation on D, and closed under operators Feas and Opt, addition of unary weighted relations to weighted relations of arbitrary arity, minimisation, and join.

Note that $\Gamma^* \subseteq \mathrm{wClone_p}(\Gamma)$, as any unary weighted relation can be obtained from the set of all $\{0, 1\}$-valued unary weighted relations by addition of unary weighted relations, scaling, addition of constants, and operator Opt. It is easy to show that addition of unary weighted relations, minimisation, and join are planarly-expressible. Set Γ^* is also closed under domain restriction and pinning, as these operations can be achieved by adding unary weighted relations and minimisation. By Theorem 5, Γ^* has the same complexity as Γ.

▶ **Definition 20.** Let Γ be a conservative language. We define an undirected graph G_Γ on vertices (a, b) for all $a, b \in D, a \neq b$. For any vertex $v = (a, b)$, we will denote by $\overline{v}$ vertex (b, a). Graph G_Γ is allowed to have self-loops. It contains edge $(a_1, b_1) - (a_2, b_2)$ if there is a binary weighted relation $\gamma \in \Gamma^*$ such that $(a_1, b_2), (b_1, a_2) \in \mathrm{Feas}(\gamma)$ and

$$\gamma(a_1, b_2) + \gamma(b_1, a_2) < \gamma(a_1, a_2) + \gamma(b_1, b_2)\,. \tag{2}$$

If there exists such a weighted relation γ with at least one of $(a_1, a_2), (b_1, b_2)$ belonging to $\mathrm{Feas}(\gamma)$, we will call the edge *soft*, otherwise the edge is *hard*. We denote by $\overline{M}$ and M the set of vertices with and without self-loops respectively.

We will show in Theorem 25, proved in Section 5, that if G_Γ has a soft self-loop then Γ is planarly-intractable. Our goal, assuming G_Γ has no soft self-loops, is to prove the following.

▶ **Theorem 21.** *If G_Γ has no soft self-loop, then Γ admits a binary multimorphism $\langle \sqcap, \sqcup \rangle$ that is an STP on M, and a ternary multimorphism $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ that is an MJN on $\overline{M}$.*

5 Proof of Theorem 21

We will need the following definition.

▶ **Definition 22.** Let ρ be an r-ary relation. For any $i, j \in \{1, \ldots, r\}$, we will denote by $\mathrm{Pr}_{i,j}(\rho)$ the projection of ρ on coordinates i and j, i.e. the binary relation defined as

$$(a_i, a_j) \in \mathrm{Pr}_{i,j}(\rho) \iff (\exists \mathbf{x} \in \rho)\ x_i = a_i \wedge x_j = a_j\,. \tag{3}$$

Relation ρ is *2-decomposable* if

$$\mathbf{x} \in \rho \iff \bigwedge_{1 \le i,j \le r} (x_i, x_j) \in \mathrm{Pr}_{i,j}(\rho)\,. \tag{4}$$

The following lemma will be useful in proving results about both Boolean and conservative valued constraint languages. For any r-tuple $\mathbf{z}$ and a subset of coordinates $I \subseteq \{1, \ldots, r\}$, we denote by $\mathbf{z}_I$ the projection of $\mathbf{z}$ onto I. For any partition of coordinates $I, J \subseteq \{1, \ldots, r\}$, we then write $\cdot$ for the inverse operation, i.e. $\mathbf{z}_I \cdot \mathbf{z}_J = \mathbf{z}$.

▶ **Lemma 23.** *Let γ be an r-ary weighted relation and $I, J \subseteq \{1, \ldots, r\}$ a partition of its coordinates. If $\mathbf{x}, \mathbf{y} \in \mathrm{Feas}(\gamma)$ and*

$$\gamma(\mathbf{x}) + \gamma(\mathbf{y}) < \gamma(\mathbf{x}_I \cdot \mathbf{y}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{x}_J)\,, \tag{5}$$

then there exist coordinates $i \in I, j \in J$ and a binary weighted relation $\gamma_{i,j} \in \{\gamma\}^*$ such that $(x_i, x_j), (y_i, y_j) \in \mathrm{Feas}(\gamma_{i,j})$ and

$$\gamma_{i,j}(x_i, x_j) + \gamma_{i,j}(y_i, y_j) < \gamma_{i,j}(x_i, y_j) + \gamma_{i,j}(y_i, x_j)\,. \tag{6}$$

Moreover, if every relation in $\{\gamma\}^*$ is 2-decomposable, then $\mathbf{x}_I \cdot \mathbf{y}_J \in \mathrm{Feas}(\gamma)$ implies $(x_i, y_j) \in \mathrm{Feas}(\gamma_{i,j})$ and $\mathbf{y}_I \cdot \mathbf{x}_J \in \mathrm{Feas}(\gamma)$ implies $(y_i, x_j) \in \mathrm{Feas}(\gamma_{i,j})$.

The following lemma gives a useful alternative characterisation of an edge in G_Γ.

▶ **Lemma 24.** *Graph G_Γ contains edge $(a_1, b_1) - (a_2, b_2)$ if, and only if, binary relation $\{(a_1, b_2), (b_1, a_2)\}$ belongs to Γ^*. The edge is soft if, and only if, at least one of binary relations $\{(a_1, a_2), (a_1, b_2), (b_1, a_2)\}$, $\{(b_1, b_2), (a_1, b_2), (b_1, a_2)\}$ belongs to Γ^*.*

▶ **Theorem 25.** *If G_Γ has a soft self-loop, language Γ is planarly-intractable.*

Proof. Let (a, b) be a vertex of G_Γ with a soft self-loop. Without loss of generality, we have $\rho = \{(a, a), (a, b), (b, a)\} \in \Gamma^*$ by Lemma 24. We denote by γ_a, γ_b the unary weighted relations defined as $\gamma_a(a) = \gamma_b(b) = 0$, $\gamma_a(b) = \gamma_b(a) = 1$, and $\gamma_a(x) = \gamma_b(x) = \infty$ for $x \notin \{a, b\}$. Set $\Gamma' = \{\rho, \gamma_a, \gamma_b\} \subseteq \Gamma^*$ can be viewed as a conservative language over a Boolean domain $\{a, b\}$. Observe that Γ' is intractable (via checking that it does not fall into either of the two tractable cases for Boolean conservative valued constraint languages [8] corresponding to the $\langle \min, \max \rangle$ and $\langle \mathrm{Mj}, \mathrm{Mj}, \mathrm{Mn} \rangle$ multimorphisms) and not self-complementary (neither of its weighted relations is self-complementary), and hence planarly-intractable by Theorem 9. Alternatively, just take the obvious encoding of the planar Maximum Independent Set problem as discussed in Section 1. ◀

In order to prove Theorem 21, we now introduce several lemmas. From now on we will assume that G_Γ has no soft self-loop.

▶ **Lemma 26.** *For any vertex v, graph G_Γ contains edge $v - \overline{v}$. There is no edge between M and $\overline{M}$, no odd cycle in M, and no soft edge in $\overline{M}$.*

Proof. As the binary equality relation belongs to Γ^*, we have edge $v - \overline{v}$ for all vertices v.

Consider any sequence of vertices v_1, v_2, v_3, v_4 such that there is an edge between every two consecutive ones, and denote $v_i = (a_i, b_i)$. By Lemma 24, there exist binary relations $\rho_i = \{(a_i, b_{i+1}), (b_i, a_{i+1})\} \in \Gamma^*$ for $i \in \{1, 2, 3\}$. Their join equals $\{(a_1, b_4), (b_1, a_4)\} \in \Gamma^*$, and hence G_Γ contains edge $v_1 - v_4$. If any of edges $v_1 - v_2, v_2 - v_3, v_3 - v_4$ is soft, we can replace the corresponding relation ρ_i with $\{(a_i, a_{i+1}), (a_i, b_{i+1}), (b_i, a_{i+1})\}$ or $\{(b_i, b_{i+1}), (a_i, b_{i+1}), (b_i, a_{i+1})\}$ to show that $v_1 - v_4$ is also soft.

Suppose that there is an edge between $s \in M$ and $t \in \overline{M}$. Then we have edges $s - t, t - t, t - s$, and hence also self-loop $s - s$, which is a contradiction.

If there is an odd cycle in M, let us choose a shortest one and denote its vertices $v_1, \ldots, v_k$ ($k \geq 3$). We have a sequence of adjacent vertices v_k, v_1, v_2, v_3, and hence v_3 and v_k are also adjacent. But that means there is a shorter odd cycle (or a self-loop) $v_3, \ldots, v_k$; a contradiction.

Finally, suppose that $s, t \in \overline{M}$ and edge $s - t$ is soft. Then we have edges $s - t, t - t, t - s$, and hence a soft self-loop at s, which is a contradiction. ◀

▶ **Lemma 27.** *Every relation in Γ^* is 2-decomposable.*

Proof. Let $\rho \in \Gamma^*$ be an r-ary relation. By definition, $\mathbf{x} \in \rho$ implies $\bigwedge_{1 \leq i,j \leq r} (x_i, x_j) \in \mathrm{Pr}_{i,j}(\rho)$ for every relation. We prove the converse implication by induction on r. If $r \leq 2$, relation ρ is trivially 2-decomposable. Let $r = 3$. Suppose for the sake of contradiction that $(x_1, x_2, x_3) \notin \rho$ even though $(y_1, x_2, x_3), (x_1, y_2, x_3), (x_1, x_2, y_3) \in \rho$ for some $y_1, y_2, y_3 \in D$. Let $\rho_1 \in \Gamma^*$ be the binary relation obtained from ρ by pinning it at the first coordinate to label x_1; we have $(x_2, y_3), (y_2, x_3) \in \rho_1$, $(x_2, x_3) \notin \rho_1$, and thus graph G_Γ contains edge $(x_2, y_2) - (x_3, y_3)$. Analogously, the graph contains edges $(x_3, y_3) - (x_1, y_1)$ and $(x_1, y_1) - (x_2, y_2)$. This is an odd cycle, so it must hold $(x_1, y_1), (x_2, y_2), (x_3, y_3) \in \overline{M}$. Let γ be a unary weighted relation with $\gamma(x_1) = 0, \gamma(y_1) = 1$ and $\gamma(z) = \infty$ for all $z \in D \setminus \{x_1, y_1\}$. By adding γ to ρ at the first coordinate and then minimising over it we show that edge $(x_2, y_2) - (x_3, y_3)$ is soft, which is a contradiction.

It remains to prove the lemma for $r \geq 4$. Let $\rho_1 \in \Gamma^*$ be the relation obtained from ρ by minimisation over the first coordinate. Relation ρ_1 is 2-decomposable by the induction hypothesis, so $(x_2, \ldots, x_r) \in \rho_1$, and hence $(y_1, x_2, \ldots, x_r) \in \rho$ for some $y_1 \in D$. Analogously, we have $(x_1, y_2, x_3, \ldots, x_r), (x_1, x_2, y_3, x_4, \ldots, x_r) \in \rho$ for some $y_2, y_3 \in D$. Pinning ρ at every coordinate $k \geq 4$ to its respective label x_k gives a ternary 2-decomposable relation ρ' such that $(x_i, x_j) \in \mathrm{Pr}_{i,j}(\rho')$ for all $i, j \in \{1, 2, 3\}$. Therefore, $(x_1, x_2, x_3) \in \rho'$ and $\mathbf{x} \in \rho$. ◀

The following lemma involves a generalisation of the definition of an edge in G_Γ to non-binary weighted relations.

▶ **Lemma 28.** *Let $\gamma \in \Gamma^*$ be an r-ary weighted relation and $I, J \subseteq \{1, \ldots, r\}$ a partition of its coordinates. If $\mathbf{x}, \mathbf{y} \in \mathrm{Feas}(\gamma)$ and*

$$\gamma(\mathbf{x}) + \gamma(\mathbf{y}) < \gamma(\mathbf{x}_I \cdot \mathbf{y}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{x}_J), \tag{7}$$

then graph G_Γ contains edge $(x_i, y_i) - (y_j, x_j)$ for some $i \in I, j \in J$. If at least one of $\mathbf{x}_I \cdot \mathbf{y}_J, \mathbf{y}_I \cdot \mathbf{x}_J$ belongs to $\mathrm{Feas}(\gamma)$, the edge is soft.

Proof. By Lemma 23, there are coordinates $i \in I, j \in J$ and a binary weighted relation $\gamma_{i,j} \in \Gamma^*$ such that $(x_i, x_j), (y_i, y_j) \in \mathrm{Feas}(\gamma_{i,j})$ and $\gamma_{i,j}(x_i, x_j) + \gamma_{i,j}(y_i, y_j) < \gamma_{i,j}(x_i, y_j) +$

$\gamma_{i,j}(y_i, x_j)$, so graph G_Γ contains edge $(x_i, y_i) - (y_j, x_j)$. If $\mathbf{x}_I \cdot \mathbf{y}_J$ or $\mathbf{y}_I \cdot \mathbf{x}_J$ belongs to Feas(γ), then (x_i, y_j) or (y_i, x_j) belongs to Feas($\gamma_{i,j}$) (as Feas(γ) is 2-decomposable by Lemma 27), and hence the edge is soft. ◀

▶ **Lemma 29.** *Let $\gamma \in \Gamma^*$ be an r-ary weighted relation and $I, J \subseteq \{1, \dots, r\}$ a partition of its coordinates. If $\mathbf{x}, \mathbf{y}, \mathbf{x}_I \cdot \mathbf{y}_J, \mathbf{y}_I \cdot \mathbf{x}_J \in$ Feas(γ) and, for all $i \in I$, $(x_i, y_i) \in \overline{M}$, then*

$$\gamma(\mathbf{x}) + \gamma(\mathbf{y}) = \gamma(\mathbf{x}_I \cdot \mathbf{y}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{x}_J)\,. \tag{8}$$

Proof. Suppose for the sake of contradiction that the equality does not hold. Without loss of generality, we may assume that $\gamma(\mathbf{x}) + \gamma(\mathbf{y}) < \gamma(\mathbf{x}_I \cdot \mathbf{y}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{x}_J)$. By Lemma 28, graph G_Γ contains a soft edge incident to (x_i, y_i) for some $i \in I$, which contradicts Lemma 26. ◀

Graph G_Γ does not have any odd cycle on vertices M. Therefore, there is a partition of M into two independent sets M_1, M_2. (In fact, it can be shown that every connected component of G_Γ restricted to M is a complete bipartite graph but we do not need this fact here.) Note that $(a, b) \in M_1$ if, and only if, $(b, a) \in M_2$, as every vertex $v \in M$ is adjacent to $\overline{v}$. We define multimorphism $\langle \sqcap, \sqcup \rangle$ as follows:

$$\langle \sqcap, \sqcup \rangle(x, y) = \begin{cases} (x, y) & \text{if } (x, y) \in M_1, & (9a)\\ (y, x) & \text{if } (x, y) \in M_2, & (9b)\\ (x, y) & \text{otherwise.} & (9c) \end{cases}$$

By definition, $\langle \sqcap, \sqcup \rangle$ is commutative on M.

▶ **Theorem 30.** *$\langle \sqcap, \sqcup \rangle$ is a multimorphism of Γ.*

Proof. Let $\gamma \in \Gamma$ be an r-ary weighted relation and $\mathbf{x}, \mathbf{y} \in$ Feas(γ). Suppose for the sake of contradiction that (1) does not hold. We partition set $\{1, \dots, r\}$ into I and J: Set J consists of all coordinates j such that case (9b) applies to (x_j, y_j); set I covers the other two cases. For any $i \in I$, either $x_i = y_i$ or $(x_i, y_i) \in M_1 \cup \overline{M}$. For any $j \in J$, $(x_j, y_j) \in M_2$ and hence $(y_j, x_j) \in M_1$. $\langle \sqcap, \sqcup \rangle$ maps $\mathbf{x}, \mathbf{y}$ to $\mathbf{x}_I \cdot \mathbf{y}_J, \mathbf{y}_I \cdot \mathbf{x}_J$, so we have $\gamma(\mathbf{x}) + \gamma(\mathbf{y}) < \gamma(\mathbf{x}_I \cdot \mathbf{y}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{x}_J)$. By Lemma 28, graph G_Γ contains edge $(x_i, y_i) - (y_j, x_j)$ for some $i \in I, j \in J$, which contradicts Lemma 26. ◀

The following definition corresponds to the "μ function" from [26, Section 6].

▶ **Definition 31.** For any $a, b, c \in D$, we say that $ab|c$ holds if a, b, c are all different labels and there exist $(s, t) \in \overline{M}$ such that binary relation $\{(a, s), (b, s), (c, t)\}$ belongs to Γ^*.

The intuition is that if $ab|c$ holds, then any minority operation on $\overline{M}$ must map any permutation of $\{a, b, c\}$ to c.

▶ **Lemma 32.** *For any $a, b, c \in D$, at most one of $ab|c$, $ca|b$, $bc|a$ holds. If $ab|c$, then $(a, c), (b, c) \in \overline{M}$.*

Proof. Suppose that both $ca|b$ and $bc|a$ hold. Then there are $(s_1, t_1), (s_2, t_2) \in \overline{M}$ and binary relations $\rho_1, \rho_2 \in \Gamma^*$ such that $\rho_1 = \{(c, s_1), (a, s_1), (b, t_1)\}$, $\rho_2 = \{(b, s_2), (c, s_2), (a, t_2)\}$. We construct binary relation ρ as $\rho(x, y) = \min_{z \in D} \rho_1(z, x) + \rho_2(z, y)$. We have $\rho \in \Gamma^*$ and $\rho = \{(s_1, s_2), (s_1, t_2), (t_1, s_2)\}$, which implies a soft edge in $\overline{M}$ and hence a contradiction.

If $ab|c$, then there are $(s, t) \in \overline{M}$ such that $\{(a, s), (b, s), (c, t)\} \in \Gamma^*$. By restricting this relation at the first coordinate to labels $\{a, c\}$ we get edge $(a, c) - (t, s)$ and thus $(a, c) \in \overline{M}$; analogously by restricting to $\{b, c\}$ we get $(b, c) \in \overline{M}$. ◀

We define multimorphism $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ as follows:

$$\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle(x,y,z) = \begin{cases} (x,y,z) & \text{if } x = y \wedge (y,z) \in \overline{M} \text{ or } xy|z, & \text{(10a)} \\ (z,x,y) & \text{if } z = x \wedge (x,y) \in \overline{M} \text{ or } zx|y, & \text{(10b)} \\ (y,z,x) & \text{if } y = z \wedge (z,x) \in \overline{M} \text{ or } yz|x, & \text{(10c)} \\ (x,y,z) & \text{otherwise.} & \text{(10d)} \end{cases}$$

Note that the operations of $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ are majorities and a minority on $\overline{M}$. Also note that in the subcase $x = y \wedge (y,z) \in \overline{M}$ of case (10a), the output has to be (x,y,z) for $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ to be an MJN multimorphism of Γ on $\overline{M}$ (and similarly for the first subcase of case (10b) and case (10c)). It is the other cases where there is some freedom and where we differ from [26].

▶ **Theorem 33.** $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ *is a multimorphism of* Γ.

We will actually prove that (1) holds with *equality.*

Proof. Suppose for the sake of contradiction this is not true for some r-ary weighted relation $\gamma \in \Gamma^*$ and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathrm{Feas}(\gamma)$; we choose γ so that it has the minimum arity among such counterexamples. We denote the r-tuples to which $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ maps $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ by $(\mathbf{f}, \mathbf{g}, \mathbf{h})$.

First we show that case (10b) does not occur. Let I be the set of coordinates i such that case (10b) applies to (x_i, y_i, z_i) and let J cover the remaining cases. Suppose that I is non-empty, and note that $\mathbf{f}_I = \mathbf{z}_I, \mathbf{g}_I = \mathbf{x}_I, \mathbf{h}_I = \mathbf{y}_I$. For every $i \in I$, it holds $(x_i, y_i), (z_i, y_i) \in \overline{M}$ (directly or by Lemma 32), and either $z_i = x_i$ or $z_i x_i | y_i$.

We claim that $\{x_i, y_i, z_i\} \times \{x_j, y_j, z_j\} \subseteq \mathrm{Pr}_{i,j}(\mathrm{Feas}(\gamma))$ for all $i \in I, j \in J$. (A detailed proof of the claim is given in the full version of this paper [12].)

Because $\mathrm{Feas}(\gamma)$ is 2-decomposable by Lemma 27, we have $\mathbf{u}_I \cdot \mathbf{v}_J \in \mathrm{Feas}(\gamma)$ for any $\mathbf{u}, \mathbf{v} \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$. It must hold

$$\gamma(\mathbf{y}_I \cdot \mathbf{x}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{y}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{z}_J) = \gamma(\mathbf{y}_I \cdot \mathbf{f}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{g}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{h}_J)\,, \tag{11}$$

otherwise we would obtain a smaller counterexample by pinning γ at every coordinate $i \in I$ to its respective label y_i. This gives $\mathbf{y}_I \cdot \mathbf{f}_J, \mathbf{y}_I \cdot \mathbf{g}_J, \mathbf{y}_I \cdot \mathbf{h}_J \in \mathrm{Feas}(\gamma)$ and hence $\mathbf{u}_I \cdot \mathbf{v}_J \in \mathrm{Feas}(\gamma)$ for any $\mathbf{u}, \mathbf{v} \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{f}, \mathbf{g}, \mathbf{h}\}$. By Lemma 29, it holds

$$\gamma(\mathbf{x}_I \cdot \mathbf{x}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{g}_J) = \gamma(\mathbf{x}_I \cdot \mathbf{g}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{x}_J)\,, \tag{12}$$
$$\gamma(\mathbf{z}_I \cdot \mathbf{z}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{f}_J) = \gamma(\mathbf{z}_I \cdot \mathbf{f}_J) + \gamma(\mathbf{y}_I \cdot \mathbf{z}_J)\,. \tag{13}$$

Adding (11), (12), and (13) shows that (1) holds as equality, which is a contradiction. Therefore, case (10b) does not apply at any coordinate.

Suppose that case (10c) applies at some coordinate i. $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ maps $(\mathbf{y}, \mathbf{x}, \mathbf{z})$ to $(\mathbf{g}, \mathbf{f}, \mathbf{h})$, which gives us another smallest counterexample to the theorem. However, at coordinate i is now applied case (10b), which was proved impossible.

Finally, we have that only cases (10a) and (10d) may occur in a smallest counterexample. But then $\langle \mathrm{Mj}_1, \mathrm{Mj}_2, \mathrm{Mn}_3 \rangle$ maps $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ to $(\mathbf{x}, \mathbf{y}, \mathbf{z})$, and hence the stated equality holds. ◀

References

1 Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241–3253, 1982. doi:10.1088/0305-4470/15/10/028.

2 Libor Barto. Constraint satisfaction problem and universal algebra. *ACM SIGLOG News*, 1(2):14–24, 2014. `doi:10.1145/2677161.2677165`.

3 Andrei Bulatov, Andrei Krokhin, and Peter Jeavons. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005. `doi:10.1137/S0097539700376676`.

4 Jin-yi Cai, Pinyan Lu, and Mingji Xia. Holographic algorithms with matchgates capture precisely tractable planar #CSP. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 427–436. IEEE Computer Society, 2010.

5 Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, 2016. `doi:10.1007/s10601-015-9198-6`.

6 David A. Cohen, Martin C. Cooper, Páidí Creed, Peter Jeavons, and Stanislav Živný. An algebraic theory of complexity for discrete optimisation. *SIAM Journal on Computing*, 42(5):915–1939, 2013. URL: `http://zivny.cz/publications/cccjz13sicomp-preprint.pdf`, `doi:10.1137/130906398`.

7 David A. Cohen, Martin C. Cooper, and Peter G. Jeavons. Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theoretical Computer Science*, 401(1-3):36–51, 2008. `doi:10.1016/j.tcs.2008.03.015`.

8 David A. Cohen, Martin C. Cooper, Peter G. Jeavons, and Andrei A. Krokhin. The Complexity of Soft Constraint Satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006. `doi:10.1016/j.artint.2006.04.002`.

9 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classification of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2001.

10 Zdeněk Dvořák and Martin Kupec. On Planar Boolean CSP. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2015.

11 Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. `doi:10.1137/S0097539794266766`.

12 Peter Fulla and Stanislav Živný. On Planar Valued CSPs. *CoRR*, abs/1602.06323, 2016. URL: `http://arxiv.org/abs/1602.06323`.

13 Peter Fulla and Stanislav Živný. A Galois Connection for Valued Constraint Languages of Infinite Size. *ACM Transactions on Computation Theory*, 8(3), 2016. Article No. 9. URL: `http://www.cs.ox.ac.uk/Stanislav.Zivny/homepage/publications/fz16toct-preprint.pdf`, `doi:10.1145/2898438`.

14 M. R. Garey and David S. Johnson. The rectilinear steiner tree problem in NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

15 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

16 Heng Guo and Tyson Williams. The complexity of planar Boolean #CSP with complex weights. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP'13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 516–527. Springer, 2013.

17 F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975. `doi:10.1137/0204019`.

18 Pavol Hell and Jaroslav Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008. `doi:10.1016/j.cosrev.2008.10.003`.

19 John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

20 Anna Huber, Andrei Krokhin, and Robert Powell. Skew bisubmodularity and valued CSPs. *SIAM Journal on Computing*, 43(3):1064–1084, 2014. doi:10.1137/120893549.

21 Peter Jeavons, Andrei Krokhin, and Stanislav Živný. The complexity of valued constraint satisfaction. *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 113:21–55, 2014. URL: http://zivny.cz/publications/jkz14.pdf.

22 Sanjeev Khanna and Rajeev Motwani. Towards a syntactic characterization of PTAS. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC'96)*, pages 329–337, 1996. doi:10.1145/237814.237979.

23 Vladimir Kolmogorov, Andrei A. Krokhin, and Michal Rolínek. The complexity of general-valued CSPs. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*. IEEE Computer Society, 2015.

24 Vladimir Kolmogorov, Michal Rolínek, and Rustem Takhanov. Effectiveness of Structural Restrictions for Hybrid CSPs. *CoRR*, abs/1504.07067, 2015. URL: http://arxiv.org/abs/1504.07067.

25 Vladimir Kolmogorov, Johan Thapper, and Stanislav Živný. The power of linear programming for general-valued CSPs. *SIAM Journal on Computing*, 44(1):1–36, 2015. doi:10.1137/130945648.

26 Vladimir Kolmogorov and Stanislav Živný. The complexity of conservative valued CSPs. *Journal of the ACM*, 60(2), 2013. Article No. 10. URL: http://zivny.cz/publications/kz13jacm-preprint.pdf, doi:10.1145/2450142.2450146.

27 Marcin Kozik and Joanna Ochremiak. Algebraic properties of valued constraint satisfaction problem. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 846–858. Springer, 2015. doi:10.1007/978-3-662-47672-7_69.

28 Bernard M. E. Moret. Planar NAE3SAT is in P. *SIGACT News*, 19(2):51–54, 1988.

29 Wolfgang Mulzer and Günter Rote. Minimum-weight Triangulation is NP-hard. *Journal of the ACM*, 55(2):11:1–11:29, 1998. doi:10.1145/1346330.1346336.

30 Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

31 Rustem Takhanov. A Dichotomy Theorem for the General Minimum Cost Homomorphism Problem. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, pages 657–668, 2010. doi:10.4230/LIPIcs.STACS.2010.2493.

32 Rustem Takhanov. Hybrid (V)CSPs and algebraic reductions. *CoRR*, abs/1506.06540, 2015. URL: http://arxiv.org/abs/1506.06540.

33 Johan Thapper and Stanislav Živný. The complexity of finite-valued CSPs. *Journal of the ACM*. To appear. URL: http://arxiv.org/abs/1210.2977v3.

34 Johan Thapper and Stanislav Živný. Necessary Conditions on Tractability of Valued Constraint Languages. *SIAM Journal on Discrete Mathematics*, 29(4):2361–2384, 2015. URL: http://zivny.cz/publications/tz15sidma-preprint.pdf, doi:10.1137/140990346.

35 Johan Thapper and Stanislav Živný. Sherali-Adams relaxations for valued CSPs. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming (ICALP'15)*, volume 9134 of *Lecture Notes in Computer Science*, pages 1058–1069. Springer, 2015. URL: http://zivny.cz/publications/tz15icalp-preprint.pdf, doi:10.1007/978-3-662-47672-7_86.

Determining Sets of Quasiperiods of Infinite Words

Guilhem Gamard[1] and Gwenaël Richomme[2]

1 **LIRMM (CNRS, Univ. Montpellier)**
UMR 5506 - CC 477, 161 rue Ada, 34095, Montpellier Cedex 5, France

2 **LIRMM (CNRS, Univ. Montpellier)**
UMR 5506 - CC 477, 161 rue Ada, 34095, Montpellier Cedex 5, France and
Univ. Paul-Valéry Montpellier 3,
Dpt MIAp, Route de Mende, 34199 Montpellier Cedex 5, France

Abstract

A word is quasiperiodic if it can be obtained by concatenations and overlaps of a smaller word, called a quasiperiod. Based on links between quasiperiods, right special factors and square factors, we introduce a method to determine the set of quasiperiods of a given right infinite word. Then we study the structure of the sets of quasiperiods of right infinite words and, using our method, we provide examples of right infinite words with extremal sets of quasiperiods (no quasiperiod is quasiperiodic, all quasiperiods except one are quasiperiodic, . . .). Our method is also used to provide a short proof of a recent characterization of quasiperiods of the Fibonacci word. Finally we extend this result to a new characterization of standard Sturmian words using a property of their sets of quasiperiods.

1998 ACM Subject Classification F.4.0 Mathematical logic and formal languages, G.2.1 Combinatorics

Keywords and phrases Combinatorics on Words; Quasiperiodicity; Sturmian words

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.40

1 Introduction

Many studies around words focus on the task of measuring regularities of strings. Various notions were introduced to that end, the strongest one being *periodicity*. Recall that an infinite word is *periodic* if it is obtained by infinite concatenation of occurrences of a word u, called a *period*.

In the context of text algorithms, Apostolico and Ehrenfeucht introduced [1] the notion of *quasiperiodicity*, which is a generalization of periodicity for finite words. A word $w = w_1 w_2 \dots w_n$ is *quasiperiodic* if there exists another word $q \neq w$ (called a *cover* or a *quasiperiod* of w) such that w is covered with occurrences of q. More precisely, for all $i \in \{1, \dots, n\}$, there exists $k \in \{0, \dots, |q| - 1\}$ such that $w_{i-k} w_{i-k+1} \dots w_{i-k+|q|-1}$ is an occurrence of q. For instance, the string "*ababa abababa ababababa ababa*" has quasiperiods *aba* and *ababa*, but it is not periodic.

This definition generalizes immediately to right infinite words (see [18]). As finite words may have several quasiperiods, infinite words may have infinitely many quasiperiods. Words with infinitely many quasiperiods are called *multi-scale quasiperiodic* (see [19]). In Section 2 we state a characterization of the set of quasiperiods of an infinite aperiodic word showing links between some extremal quasiperiods and some square factors and right special factors. An important consequence of this result is to provide a general method to determine the set of quasiperiods of any infinite word. Next we describe several examples of uses of this method.

In [6], Christou, Crochemore and Iliopoulos provide characterizations of quasiperiods of Fibonacci strings. One of their motivations was that “Fibonacci strings are important in many concepts [3] and are often cited as a worst case example for many string algorithms.” However, Fibonacci strings are not always the best words for this purpose. For example, Groult and Richomme [11] proved that the algorithms provided by Brodal and Pedersen [5] and by Iliopoulos and Mouchard [12] to compute all the quasiperiods of a word do not reach their worst case on Fibonacci strings. They proved that those algorithms were optimal, and provided a family of strings reaching the worst case. Nevertheless, the study of finite Fibonacci strings is indeed of great interest: see, e.g., references in [6].

Some of the results from [6] were recently reformulated by Mousavi, Schaeffer and Shallit as a new characterization of quasiperiods of the infinite Fibonacci word [21] (another one was given in [14]). They use this result, among many others, to show how to build automated proofs of some results about the Fibonacci word. Using the method to determine the set of quasiperiods of any infinite word described in Section 2, we provide a short proof of the above mentioned characterization (Section 4.1).

The infinite Fibonacci word is a special case of Sturmian words (and therefore Fibonacci strings are special cases of factors of Sturmian words). A natural question is whether the previous characterization of quasiperiods of the Fibonacci word can be extended to other Sturmian words. Unfortunately, some Sturmian words are not quasiperiodic [15]; more precisely, a Sturmian word is quasiperiodic if and only if it is not a Lyndon word. However, we can still extend our characterization to *standard* Sturmian words, i.e. Sturmian words having all their left special factors as prefixes (Section 4.3).

Sturmian words are not necessarily quasiperiodic, but their bi-infinite counterparts are always multi-scale quasiperiodic. This result can be extended to subshifts, i.e. topological spaces generated from languages by the shift operation. A subshift is *quasiperiodic* (resp. *multi-scale quasiperiodic*) if and only if it is generated by a word which is quasiperiodic (resp. multi-scale quasiperiodic). Monteil and Marcus proved [19] that all Sturmian subshifts are multi-scale quasiperiodic. They also proved that multi-scale quasiperiodic shifts have zero topological entropy, are minimal (their words are uniformly recurrent), and that all of their factors have frequencies (see [9] for a generalization to two-dimensional words).

The main tool of [19] is a so-called *derivation* operation, which takes the inverse image of a word by a well-chosen morphism. The derivative of a quasiperiodic word $\mathbf{w}$ is another word which describes the lengths of the overlaps in $\mathbf{w}$ between each two consecutive occurrences of its quasiperiods. While reading [19], one naturally asks whether the derivation operation preserves multi-scale quasiperiodicity. In other terms, given a multi-scale quasiperiodic word, does its derivative still has infinitely many quasiperiods? In Section 3, we show this is not the case. We provide a right infinite multi-scale quasiperiodic word whose derivative is non-quasiperiodic. While discussing properties of the derivation operation, we also provide a word such that each quasiperiod has the previous (in terms of length) one as a quasiperiod. This nested effect can be avoided; we provide a multi-scale quasiperiodic word with only non-quasiperiodic quasiperiods. The proof of properties of our examples all involve our general method.

Let us summarize the main parts of our paper. In Section 2, we present general properties of quasiperiods of right infinite words and a general method to determine them. In Section 3, we present our results around the derivation operation. In Section 4, we provide our proof of the characterization of the quasiperiods of the Fibonacci word and its generalization to the new characterization of standard Sturmian words.

We assume readers are aware of general results and definitions in Combinatorics on Words (see for instance [16, 17]). Let us recall basic definitions. All infinite words we consider are right infinite words over finite alphabet. Given an alphabet A, A^* (resp. A^ω) denotes the set of finite (resp. infinite) words. The empty word is denoted by ε. The length of a finite word u is denoted $|u|$. A word u is a *factor* of a word v if $v = pus$ for some words p and s. When p is empty (resp. s is empty), u is a *prefix* (resp. a *suffix*) of v. When p and s are not empty, u is an *internal factor* of v. If there exists at least two different letters α and β such that $u\alpha$ and $u\beta$ are factors of v, u is a *right special factor* of v. When αu and βu are factors of v, u is a *left special factor* of v. A *bispecial factor* is a factor which is both left and right special.

2 Basic properties of sets of quasiperiods

Here we provide materials that help to determine quasiperiods of an infinite word poiting first the two main results of this section. Our first result characterizes periodic infinite words using their sets of quasiperiods. We give its proof later, as a consequence of Proposition 2.3, an intermediate result to show Theorem 2.2.

▶ **Proposition 2.1.** *An infinite word is periodic if and only if all its sufficiently long prefixes belong to its set of quasiperiods.*

Consider an infinite word $\mathbf{w}$. As any quasiperiod of $\mathbf{w}$ is one of its prefixes, $\mathbf{w}$ has at most one quasiperiod of length n for each integer n. Thus the knowledge of the lengths of quasiperiods is equivalent to the knowledge of the quasiperiods themselves. After Proposition 2.1, it is clear that lengths of quasiperiods of an aperiodic infinite word, if there are any, are distributed into intervals of integers which are finite and disjoint (possibly singletons). The next theorem characterizes these intervals.

▶ **Theorem 2.2.** *Let $\mathbf{w}$ be an infinite aperiodic word and, for any integer i, let p_i denote the prefix of length i of $\mathbf{w}$. The set of lengths of quasiperiods of $\mathbf{w}$ is an union (possibly empty) of disjoint intervals of integers. If $[i, j]$ is such an interval, then there are no quasiperiods of lengths $i - 1$ nor $j + 1$. Moreover,*

- *p_ip_i is a factor of $\mathbf{w}$;*
- *p_{i-1} is not an internal factor of p_ip_{i-1};*
- *p_j is a right special factor of $\mathbf{w}$;*
- *for all k such that $i \le k < j$, the word p_k is not a right special factor of $\mathbf{w}$.*

This theorem induces a method to determine the quasiperiods of an aperiodic infinite word $\mathbf{w}$. This method consists in determining first among all prefixes p such that pp is a factor of $\mathbf{w}$ or such that p is right special, those such that $\mathbf{w}$ is p-quasiperiodic. With this information, using Theorem 2.2, we can deduce that:

- for any prefix p of $\mathbf{w}$ such that $\mathbf{w}$ is p-quasiperiodic and pp is a factor of $\mathbf{w}$, if q is the smallest right special prefix of $\mathbf{w}$ longer than p, then all prefixes π such that $|p| \le |\pi| \le |q|$ are quasiperiods of $\mathbf{w}$;
- for any prefix q of $\mathbf{w}$ which is a quasiperiod and right special, if p is the smallest prefix such that $|p| > |q|$ and pp is a factor of $\mathbf{w}$, then all prefixes π such that $|q| < |\pi| < |p|$ are *not* quasiperiods of $\mathbf{w}$ (observe that when $\mathbf{w}$ has finitely many quasiperiods, p does not exist).

Moreover, any multi-scale quasiperiodic word which is not periodic has infinitely many quasiperiods which are right special factors. If an infinite word has finitely many quasiperiods, then the longest one is right special. The converse is not true: some multi-scale quasiperiodic words have infinitely many right special factors which are not quasiperiods. Construction of such counter-examples is left to the readers.

In Theorem 2.2, the existence of disjoint intervals is ensured by aperiodicity and Proposition 2.1. The first two items are consequence of Proposition 2.4 below, and the last two items are a consequence of Proposition 2.3. The end of this section is dedicated to the proof of these propositions.

▶ **Proposition 2.3.** *Let* $\mathbf{w}$ *be an infinite word with a quasiperiod* q*. Let* a *be the letter such that* qa *is a prefix of* $\mathbf{w}$*. The word* qa *is a quasiperiod of* $\mathbf{w}$ *if and only if* q *is not right special.*

Proof. If q is not a right special factor of $\mathbf{w}$ then each of its occurrences is followed by the letter a and so, since q is a quasiperiod of $\mathbf{w}$, qa is also a quasiperiod of $\mathbf{w}$.

Conversely assume that qa is a quasiperiod of $\mathbf{w}$ and let b be a letter such that qb is a factor of $\mathbf{w}$. Then qb is a factor of a qa-quasiperiodic factor u of $\mathbf{w}$. For a length reason, we can assume that u is the overlap of two occurrences of qa. Therefore there exist words p_1, p_2, s_1 and s_2 such that $u = p_1\, qb\, s_1 = qa\, s_2 = p_2\, qa$ with $0 < |p_2| \leq |qa|$ and $0 < |s_2| \leq |qa|$. If either p_1 or s_1 is the empty word then $a = b$, so assume these words are not empty. By a classical result (see for instance [16, Prop. 1.3.4]), the equation $p_2\, qa = qa\, s_2$ implies that there exist words x and y and an integer k such that $p_2 = xy$ and $qa = (xy)^k x$ and $s_2 = yx$. In particular, qa is a factor of the periodic word $(xy)^\omega$. Equation $p_1\, qb\, s_1 = p_2\, qa$ implies that qb is also a factor of $(xy)^\omega$. As $|qa| = |qb| \geq |xy|$ (recall that $xy = p_2$ and $|qa| \geq |p_2|$), we conclude that $a = b$, so the word q is not right special factor of $\mathbf{w}$. ◀

As shown below, Proposition 2.1 is a corollary of Proposition 2.3.

Proof of Proposition 2.1. Let $\mathbf{w}$ be an infinite word. If $\mathbf{w}$ is periodic with period of length n, then any prefix of $\mathbf{w}$ with length at least n is a quasiperiod of $\mathbf{w}$.

Conversely assume that, for an integer n, all prefixes with length at least n are quasiperiods of $\mathbf{w}$. For $i \geq 0$, let p_i be the prefix of length i. As $\mathbf{w}$ is p_n-quasiperiodic, $\mathbf{w}$ has at least two occurrences. There exists a word u such that both p_n and up_n are prefixes of $\mathbf{w}$. By hypothesis and Proposition 2.3, p_n is not right special. Each of its occurrences extend to p_{n+1}. Hence both p_{n+1} and up_{n+1} are prefixes of $\mathbf{w}$. Iterating this argument, for all $i \geq n$, both p_i and up_i are prefixes of $\mathbf{w}$. Thus $\mathbf{w} = u\mathbf{w}$: $\mathbf{w}$ is periodic with period u. ◀

Proposition 2.3 provides a first piece of information on some extremal quasiperiods of a word. The next result provides further information. It generalizes an observation made in [14] for the smallest quasiperiod of a word.

▶ **Proposition 2.4.** *Assume that* qa *is a quasiperiod of an infinite word* $\mathbf{w}$ *for some word* q *and some letter* a*. The word* q *is not a quasiperiod of* $\mathbf{w}$ *if and only if the word* qaq *is a factor of* $\mathbf{w}$ *and* q *is not an internal factor of* qaq*.*

Proof. If qaq is not a factor of $\mathbf{w}$, then each occurrence of qa is properly overlapped by the next occurrence of qa. This implies that each occurrence of q is overlapped by or concatenated to the next occurrence of q, that is $\mathbf{w}$ is q-quasiperiodic.

The converse is immediately true. If a word $\mathbf{w}$ contains qaq as a factor and if q is not an internal factor of qaq, then q cannot be a quasiperiod of $\mathbf{w}$. ◀

It should be observed for understanding Theorem 2.2 that under the hypotheses "qa is a quasiperiod of $\mathbf{w}$" and "q is not an internal factor of qaq", the word qaq is a factor of $\mathbf{w}$ if and only if $qaqa$ is a factor of $\mathbf{w}$.

3 On multiscale properties

As explained in the introduction, the goal of this section is threefold. First we provide examples to illustrate the usage of Theorem 2.2. Second we want to show that the derivation operation introduced in [19] does not preserve multiscale quasiperiodicity. Third we aim to study the structure of the relation "is a quasiperiod of" for multiscale quasiperidic words.

A finite word u is said to be *superprimitive* if it is not quasiperiodic. It can be seen (for instance as a consequence of the study made in [14]) that the Fibonacci word has both infinitely many superprimitive quasiperiods and infinitely many non-superprimitive quasiperiods. Moreover, due to its morphic structure, the Fibonacci word has an infinite sequence of *nested quasiperiods*. In other terms, it has a sequence of quasiperiods $(q_n)_{n\geq 0}$ such that, for each $n \geq 0$, q_{n+1} is q_n-quasiperiodic. There are many possible structures for the relation "is a quasiperiod of" inside multiscale quasiperiodic words. We provide several extremal examples throughout this section.

3.1 A multiscale quasiperiodic word with only one superprimitive quasiperiod

▶ **Proposition 3.1.** *There exists a multiscale quasiperiodic word having only one superprimitive quasiperiod.*

This proposition is a direct corollary of the next lemma.

Recall that a morphism between two sets of words A^* and B^* (with A, B finite alphabets) is an application which commutes with concatenation. A morphism is entirely defined by the images of the letters. A morphism is called *non-erasing* if no image of letters is the empty word. As usual, for a word u and a morphism h, we denote $h^\omega(u)$ the word $\lim_{n\to\infty} h^n(u)$ when it exists.

▶ **Lemma 3.2.** *Let h be the morphism from $\{a,b\}^*$ to $\{a,b\}^*$ defined by $h(a) = abaaba$ and $h(b) = bababa$. The quasiperiods of $h^\omega(a)$ are exactly the words $h^n(aba)$ for $n \geq 0$, and all these words are aba-quasiperiodic.*

Proof. This lemma is a direct consequence of Theorem 2.2 and the following four steps. Indeed Steps 1 to 3 determine the prefixes p of $h^\omega(a)$ such that pp is a factor of $h^\omega(a)$ and $h^\omega(a)$ is p-quasiperiodic. Step 4 and Theorem 2.2 allow to conclude that words $h^n(aba)$ are the only quasiperiods of $h^\omega(a)$.

Step 1: The prefixes p such that pp is a factor of $h^\omega(a)$ are $h^n(a)$, $h^n(ab)$ and $h^n(aba)$.

Let p be such a word. If $|p| \leq 6$, it can be checked that $p \in \{a, ab, aba, h(a)\}$. Assume $|p| > 6$. The word p has $abaabab$ as a prefix. Observe that if $\pi abaabab$ is a prefix of $h^\omega(a)$ for a word π, then necessarily $\pi \in h(A^*)$. As pp is a factor of $h^\omega(a)$, there exist words π and p' such that $h(\pi)pp$ is a prefix of $h^\omega(a)$ and $p = h(p')$. Hence $h(\pi p'p')$ is a prefix of $h^\omega(a)$. As $h(a)$ and $h(b)$ are not prefixes of one another, $\pi p'p'$ is a prefix of $h^\omega(a)$. As $|\pi p'p'| < |h(\pi p'p')|$, the proof of this step ends by induction.

Step 2: Words $h^n(a)$ and $h^n(ab)$ are not quasiperiods of $h^\omega(a)$.

Assume $h^n(a)$ is a quasiperiod of $h^\omega(a)$ for a smallest integer n. Observe $n \neq 0$ and $n \neq 1$. Let $(\pi_k)_{k\geq 0}$ be a sequence of words such that $\pi_k h^n(a)$ is a prefix of $h^\omega(a)$ and, for $k \geq 0$, $|\pi_{k+1}| - |\pi_k| \leq |h^n(a)|$. Since $abaabab$ is a prefix of $h^n(a)$ (as $n \geq 2$), for each $k \geq 0$, $\pi_k = h(\pi'_k)$ for a word π'_k. Observe that $(\pi'_k h^{n-1}(a))_{k\geq 0}$ is a sequence of prefixes of $h^\omega(a)$ and $|\pi'_{k+1}| - |\pi'_k| = \frac{|\pi_{k+1}|-|\pi_k|}{6} \leq \frac{|h^n(a)|}{6} = |h^{n-1}(a)|$. Hence $h^{n-1}(a)$ is a quasiperiod of $h^\omega(a)$. This contradicts the choice of n.

Similarly one can prove that words $h^n(ab)$ are not quasiperiods of $h^\omega(a)$.

Step 3: Words $h^n(aba)$ are quasiperiods of $h^\omega(a)$.

This is a direct consequence of the fact that aba is a quasiperiod of any word in $h(a\{a,b\}^\omega)$ and so of $h^\omega(a)$.

Step 4: Words $h^n(aba)$ are right special factors of $h^\omega(a)$.

This is direct consequence of the fact that aba is a right special factor of $h^\omega(a)$ and $h(a)$ and $h(b)$ begin with different letters.

By the previous steps and Theorem 2.2, factors $h^n(aba)$ are both beginnings and endings of intervals of quasiperiods. Therefore, they are the only quasiperiods of $h^\omega(a)$. ◀

Let us observe that, as aba is a quasiperiod of $h(aba)$, for any $n \geq 1$, $h^n(aba)$ is $h^{n-1}(aba)$-quasiperiodic. Thus not only $h^\omega(a)$ has a unique superprimitive quasiperiod but its sequence of quasiperiods (sorted by increasing length) is a sequence of nested quasiperiods.

3.2 About normal form and derivation

In [20], Mouchard introduced two normal forms to decompose a quasiperiodic (finite) word. A *border* of a nonempty word u is a factor different from u which is both a prefix and a suffix of u. Let $\mathcal{B}(q)$ be the set of borders of q; let $\mathcal{L}(q)$ be the set of words u such that $q = uv$ with $v \in \mathcal{B}(q)$; and let $\mathcal{R}(q)$ be the set of words u such that $q = vu$ with $v \in \mathcal{B}(q)$. Note that the empty word belongs to $\mathcal{B}(q)$ and q belongs to $\mathcal{L}(q) \cap \mathcal{R}(q)$. Any q-quasiperiodic finite word can be decomposed as a concatenation of elements of $\mathcal{L}(q)$ (or as a concatenation of elements of $\mathcal{R}(q)$). Mouchard proved that if q is superprimitive then the decomposition over $\mathcal{L}(q)$ (resp. over $\mathcal{R}(q)$) is unique. This decomposition is called the *left normal form* (resp. *right normal form*) of the word.

As observed by Marcus and Monteil [19] this result extends naturally to infinite words. They introduced a derivation operation. Observe that any word has at most one element of $\mathcal{L}(q)$ of each length. If $\mathbf{w}$ is decomposed over $\mathcal{L}(q)$ and if $(\ell_n)_{n\geq 0}$ is the decomposition, then the *left derivated word* is the word $(|q| - |\ell_n|)_{n\geq 0}$ (written over the alphabet $\{0, \ldots, |q| - 1\}$). Marcus and Monteil showed that this derivation operation is a desubstitution operation, that is, the inverse operation of taking the image of an infinite word under a morphism. This morphism, that we called (following the idea of [19]) the *left integrating morphism*, is defined from $\{0, \ldots, |q| - 1\}^*$ to A^* by mapping i on the prefix of length $|q| - i$ of q.

For instance, consider the *Fibonacci word* $\mathbf{F}$, that is the fixed point of the *Fibonacci morphism* φ defined by $\varphi(a) = ab$ and $\varphi(b) = a$. We know it is aba-quasiperiodic (as $\mathbf{F}$ does not contain aaa and bb as factors and starts with ab - see also [14]). With this quasiperiod $q = aba$, the morphism used to derivate any q-quasiperiodic word is the morphism defined by $0 \mapsto aba$, $1 \mapsto ab$ which, up to a renaming of letters, is φ^2. Hence $\mathbf{F}$ is its own derivative word, and therefore can be derivated arbitrarily many times.

Because of this terminology of "derivation", one could expect that, if a word is multiscale quasiperiodic, then it could be derivated infinitely many times. The next result, combined with Lemma 3.2, disproves this intuition.

▶ **Proposition 3.3.** *Let $\mathbf{w}$ be a quasiperiodic word such that for all quasiperiods q there is no quasiperiod of length $|q| + 1$. Given any quasiperiod q of $\mathbf{w}$, the corresponding left derivated word is not quasiperiodic.*

Proof. Let q be any quasiperiod of $\mathbf{w}$ and let $\mathbf{x}$ be the corresponding left derivated word. We denote by ν the morphism underlying the derivation: $\mathbf{w} = \nu(\mathbf{x})$. By construction of ν, for all letters α in $\{0, \ldots, |q| - 1\}$, $\nu(\alpha)$ begins with the first letter, say a, of $\mathbf{w}$. Assume that Q is a quasiperiod of $\mathbf{x}$. Then $\nu(Q)$ is a quasiperiod of $\mathbf{w}$ (as ν is a non-erasing morphism - basic fact mentioned in [15]). As $\nu(Q)$ is always followed by the letter a: $\nu(Q)a$ is a quasiperiod of $\mathbf{w}$, a contradiction with the hypothesis. ◀

As the derivation operation is associated to the left normal form, one can ask whether a similar definition associated to the right normal form could give a better behavior. To any q-quasiperiodic word, we call *right derivated word*, the word $(|q| - |r_n|)_{n\geq 0}$ where $(r_n)_{n\geq 0}$ is the decomposition of $\mathbf{w}$ over $\mathcal{R}(q)$. We call *right integrating morphism*, the morphism defined by mapping i on the suffix of length $|q| - i$ of q. The example of the Fibonacci word, developed below, shows that right derivation does not preserve multiscale quasiperiodicity.

The smallest quasiperiod of the Fibonacci word is aba. The corresponding right integrating morphism is the morphism μ defined by $\mu(a) = aba$; $\mu(b) = ba$. One can observe that $\varphi^2(a)aba = aba\mu(a)$ and $\varphi^2(b)aba = aba\mu(b)$. Thus $\varphi^2(u)aba = aba\mu(u)$ for any word u. Applying the previous formula for arbitrary large prefixes of $\mathbf{F}$, we get $\varphi^2(\mathbf{F}) = aba\mu(\mathbf{F})$, that is $\mathbf{F} = \mu(a\mathbf{F})$. The right derivated word of $\mathbf{F}$ is the word $a\mathbf{F}$. This word is a Lyndon infinite word and consequently it is not quasiperiodic (see [15]).

We end this section with an example of word for which both left and right derivation does not provide a multiscale quasiperiodic word. The proof of these properties is omitted (but still can be done using our general method). Let us consider the following four morphisms f, g, λ, χ, and the word $\mathbf{w}_{fg}$ defined by $\mathbf{w}_{fg} = f(g^\omega(a))$ and

$$f : \begin{cases} a \mapsto aba \\ b \mapsto ba \end{cases} \qquad g : \begin{cases} a \mapsto aba \\ b \mapsto bba \end{cases} \qquad \lambda : \begin{cases} a \mapsto aba \\ b \mapsto ab \end{cases} \qquad \chi : \begin{cases} a \mapsto baa \\ b \mapsto bab \end{cases}$$

▶ **Lemma 3.4.** *The word $\mathbf{w}_{fg}$ is equal to $\lambda(\chi^\omega(b))$. It is multiscale quasiperiodic and its quasiperiods are the words $f(g^n(a)) = \lambda(\chi^n(a))$. For any quasiperiod q of $\mathbf{w}_{fg}$, the right derivated word of $\mathbf{w}_{fg}$ is $g^\omega(a)$ and its left derivated word is $\chi^\omega(b)$. Both words $g^\omega(a)$ and $\chi^\omega(b)$ are not quasiperiodic.*

By lack of place, the proof of this lemma is omitted.

One can verify that $\mathbf{w}_{fg}$ is a fixed point of the morphism defined by $h(a) = a$ and $h(b) = babaab$ (this property is a consequence of $h(f(u)) = f(g(u))$ for all words u). This opens a new question: can any multiscale quasiperiodic word $\mathbf{w}$ be desubstituted into another multiscale quasiperiodic word?

3.3 A multiscale quasiperiodic word with all quasiperiods superprimitive

Let $q = abbababba$ and consider morphism ψ defined by:

$$\begin{cases} \psi(a) = (abbab)^7 = abbababbababbababbababbababbababbab \\ \psi(b) = bababba(q)^2(abbab)^2 = bababbaabbababbaabbababbaabbababbab \end{cases}$$

▶ **Proposition 3.5.** *The quasiperiods of the infinite word $\psi^\omega(a)$ are the words $\psi^n(q)$ with $n \geq 0$. Moreover each of these quasiperiods is superprimitive.*

This proposition is a synthesis of the next three lemmas.

▶ **Lemma 3.6.** *The word $\psi^\omega(a)$ is $\psi^n(q)$-quasiperiodic for each $n \geq 0$.*

Proof. As already recalled in the proof of Proposition 3.3, for any non-erasing morphism f and any infinite word $\mathbf{w}$, if $\mathbf{w}$ is q-quasiperiodic then $f(\mathbf{w})$ is $f(q)$-quasiperiodic. Hence to prove the lemma, we just need to prove that $\psi^\omega(a)$ is q-quasiperiodic.

As both words obtained from $\psi(a)$ and $ab\psi(b)$ removing their last b are q-quasiperiodic, for any infinite word $\mathbf{w}$, $\psi(a\mathbf{w})$ is q-quasiperiodic. In particular $\psi^\omega(a)$ is q-quasiperiodic. ◀

▶ **Lemma 3.7.** *For any $n \geq 0$, the word $\psi^n(q)$ is superprimitive.*

Proof. Assume by contradiction that n is the least integer such that $\psi^n(q)$ is quasiperiodic, and let Q be one of its quasiperiods. Necessarily $n \geq 1$. The word $\psi(a)ba$ is a prefix of $\psi^n(q)$. An exhaustive verification shows that a prefix of $\psi(a)ba$ is a border of $\psi^n(q)$ if and only if this prefix is of the form $(abbab)^\ell$ with $\ell \in [1;7]$ when $n = 1$ and $\ell \in \{1, 2\}$ when $n \geq 2$. As any $abbab$-quasiperiodic word cannot contain the word aa as a factor, no prefix of $\psi(a)ba$ can be a quasiperiod of $\psi^n(q)$. It follows that $\psi(a)ba$ must be a prefix of Q.

Observe that if $\psi(a)ba$ is a factor of the image by ψ of a word (finite or infinite) u, then any occurrence of $\psi(a)ba$ in $\psi(u)$ corresponds to a prefix of the image of a suffix of u. Consequently, considering the last occurrence of Q in $\psi^n(q)$, we then deduce that $Q = \psi(Q')$ for some word Q'. That Q is a quasiperiod of $\psi^n(q)$ means there exists a double sequence of words $(p_i, s_i)_{1 \leq i \leq k}$ such that $\psi^n(q) = p_i Q s_i$ for each i in $[1; k]$, $p_1 = \varepsilon = s_k$ and, for each i in $[1; k-1]$, $|p_i Q| \geq |p_{i+1}| > |p_i|$. The observation at the beginning of the paragraph implies that, for each i in $[1; k]$, $p_i = \psi(p'_i)$ for some word p'_i. As $Q = \psi(Q')$ and as images of letters by ψ have all the same length, for each i in $[1; k]$ $s_i = \psi(s'_i)$ for some word s'_i. Injectivity of ψ implies that for each i in $[1; k]$, $\psi^{n-1}(q) = p'_i Q' s'_i$. Moreover $p'_1 = \varepsilon = s'_k$. Observe for each i in $[1; k]$, $|p_i| = 35|p'_i|$ and $|q| = 35|Q'|$. Hence for each i in $[1; k-1]$ $|p'_i Q| \geq |p'_{i+1}| > |p'_i|$. Hence $\psi^{n-1}(a)$ is Q'-quasiperiodic. This contradicts the minimality in the choice of n. ◀

▶ **Lemma 3.8.** *If Q is a quasiperiod of $\psi^\omega(a)$ then $Q = \psi^n(q)$ for some integer $n \geq 0$.*

Proof. Observe that q is right special in $\psi^\omega(a)$, and so, as $\psi(a)$ and $\psi(b)$ begin with different letters, for all $n \geq 0$, the word $\psi^n(q)$ is right special. Thus by Theorem 2.2, we just have to prove that, if Q is a quasiperiod of $\psi^\omega(a)$ and QQ is a factor of $\psi^\omega(a)$, then $Q = \psi^n(q)$ for some integer $n \geq 0$. We use arguments similar to those used in the proof of Lemma 3.7.

The word $\psi(a)ba$ is a prefix of $\psi^\omega(a)$. An exhaustive verification shows that among all prefixes of this word, only q is a quasiperiod of $\psi^\omega(a)$. Let us assume that Q is a quasiperiod of $\psi^\omega(a)$ with $|Q| \geq |\psi(a)ba|$ and QQ a factor of $\psi^\omega(a)$. As in the proof of Lemma 3.7, we observe that if $u\psi(a)ba$ is a prefix of $\psi^\omega(a)$ then $u = \psi(v)$ for some word v. Thus this also holds if uQ is a prefix of $\psi^\omega(a)$. From the fact that QQ is a factor of $\psi^\omega(a)$, we deduce $Q = \psi(Q')$ for a word Q'. Moreover, possibly acting more precisely as in the proof of Lemma 3.7, we can see that Q' must be a quasiperiod of $\psi^\omega(a)$ with $Q'Q'$ a factor of $\psi^\omega(a)$. Hence by induction on $|Q|$, we can deduce that $Q = \psi^n(q)$ for some integer $n \geq 0$. ◀

To end this section, we emphasize the interest of the previous examples by mentioning that when there are arbitrarily large intervals of lengths of quasiperiods, then there exists arbitrarily large quasiperiods that are not superprimitive.

▶ **Lemma 3.9.** *Let $\mathbf{w}$ be an aperiodic multiscale quasiperiodic word for which there exist arbitrary large intervals $[i, j]$ of lengths of quasiperiods. This word $\mathbf{w}$ admits an infinite sequence of nested quasiperiods.*

Proof. Let q_0 be any quasiperiod of $\mathbf{w}$. By hypothesis, there exists an interval $[i, j]$ with $j - i \geq |q_0|$ such that for all integers k in $[i, j]$, the prefix of length k of $\mathbf{w}$ is one of its quasiperiods. As $j - i \geq |q_0|$, there exists an integer k in $[i, j]$ such that the prefix of length k of $\mathbf{w}$ is q_0-quasiperiodic. By iterating that reasoning, we can construct a sequence of nested quasiperiods of $\mathbf{w}$. ◀

4 Quasiperiods of standard Sturmian words

The starting result of this section is the recent characterization of the Fibonacci word of [21] mentioned in the introduction. We provide a short proof using the general method of Section 2. We also reformulate this result in such a way it could be generalized to all standard Sturmian words. This is done in Section 4.2 before showing in Section 4.3 this is a characteristic property of this family of words.

4.1 Fibonacci example

We denote by $(F_n)_{n\geq 0}$ the sequence of Fibonacci integers ($F_0 = 1$, $F_1 = 1$, $F_{n+2} = F_{n+1} + F_n$ for $n \geq 0$) and by $(f_n)_{n\geq 1}$ the sequence of finite Fibonacci words ($f_1 = a$, $f_2 = ab$, $f_{n+2} = f_{n+1}f_n$ for $n \geq 1$). It is well-known that the infinite Fibonacci word $\mathbf{F} = \lim_{n\to\infty} f_n$ is also the fixed point of the morphism φ defined by $\varphi(a) = ab$ and $\varphi(b) = a$.

▶ **Lemma 4.1** (see [21]). *For all $n \geq 0$, the prefix of length n of $\mathbf{F}$ is a quasiperiod of $\mathbf{F}$ if and only if $n \notin \{F_p - 1 \mid p \geq 0\}$.*

Proof. It is well-known that left special factors of $\mathbf{F}$ coincide with its prefixes (see for instance [4, Prop. 4.10.3]). Thus determining prefixes of $\mathbf{F}$ that are right special is equivalent to determining the bispecial factors of $\mathbf{F}$. Let us denote by $(g_n)_{n\geq 2}$ the sequence of prefixes of $\mathbf{F}$ of length $(F_{n+1} - 2)_{n\geq 2}$. These words are exactly the bispecial factors of $\mathbf{F}$ ($\mathbf{F}$ is a standard Sturmian word; by [7] palindromic prefixes of standard Sturmian word are its bispecial factors; lengths of palindromic prefixes of $\mathbf{F}$ are computed in [8]).

For any $n \geq 3$, f_nf_n is a prefix of $\mathbf{F}$ (indeed $abaaba = (\varphi^2(a))^2$ is a prefix of $\mathbf{F}$ which is the fixed point of φ). Moreover as $\mathbf{F}$ is $\varphi^2(a)$-quasiperiodic, $\mathbf{F}$ is f_n-quasiperiodic.

By Theorem 2.2, for all $n \geq 3$, for each prefix π of $\mathbf{F}$ with $F_n \leq |\pi| \leq |g_n| = F_{n+1} - 2$, π is a quasiperiod of $\mathbf{F}$. Moreover the prefix of length $|g_n| + 1 = F_{n+1} - 1$ is not a quasiperiod of $\mathbf{w}$. Finally prefixes of $\mathbf{F}$ of length $F_0 - 1 = F_1 - 1 = 0$, $F_2 - 1 = 1$ or $F_3 - 1 = 2$ are not quasiperiods of $\mathbf{F}$. ◀

As mentioned in the previous proof, bispecial factors of the Fibonacci word are its prefixes of length $F_{n+1} - 2$ for $n \geq 2$.

▶ **Corollary 4.2.** *For all $n \geq 0$, the prefix of length $n + 1$ of $\mathbf{F}$ is a quasiperiod of $\mathbf{F}$ if and only if the prefix of length n of $\mathbf{F}$ is not bispecial.*

4.2 Quasiperiods of standard Sturmian words

The study of quasiperiods in Sturmian words dates back to an original question of Marcus [18]: "Is every Sturmian word quasiperiodic?" This question was completely answered in [15]: a Sturmian word is quasiperiodic if and only if it is not a Lyndon word. In other words, in any Sturmian shift, all but two words are quasiperiodic. Episturmian words, a family of words that include Sturmian words, were also considered and a characterization of all quasiperiods of any episturmian word was provided (see [10, Th. 4.19]). This characterization is quite elaborate and uses the so-called directive word of the studied episturmian word. With a bit of work, Lemma 4.1 could be deduced from this characterization. This is also the case of the next result, which generalizes Corollary 4.2.

Let us recall that an infinite word is *Sturmian* if and only if it has exactly $n + 1$ factors of length n for all n. By the well-known Morse-Hedlund theorem, Sturmian words are the aperiodic words with the least possible number of factors. These words have exactly one left special factor and one right special factor of each length. A Sturmian word is called *standard Sturmian* if its left special factors coincide with its prefixes. By [15], they are multiscale quasiperiodic.

▶ **Proposition 4.3.** *Let* $\mathbf{w}$ *be a standard Sturmian word and* n *a positive integer. Then the prefix of length* n *of* $\mathbf{w}$ *is a quasiperiod if and only if its prefix of length* $n-1$ *is* not *bispecial.*

This proposition could be proved using [10, Th. 4.19]. We rather provide another argument, which may be reused in other contexts. We work with *graphs of words* and *return words*, for which we recall the definitions.

Let n be an integer and $\mathbf{w}$ be an infinite word. The *n-th order graph of words of* $\mathbf{w}$, denoted by $\mathcal{G}_{\mathbf{w}}(n)$, is the directed graph whose vertices are the factors of length n of $\mathbf{w}$, such that there is an edge between two vertices x and y if and only if $\mathbf{w}$ has a factor of length $n+1$ which has x as a prefix and y as a suffix. Observe that a factor v of $\mathbf{w}$ is right special if and only if its vertex in $\mathcal{G}_{\mathbf{w}}(|v|)$ has at least two outgoing edges. Therefore the graph of words allows to visualize right special factors, so it can help searching for quasiperiods using Proposition 2.3.

Let u be a factor of $\mathbf{w}$. A word v is a *return word* for u in $\mathbf{w}$ if and only if uv is a factor of $\mathbf{w}$ which has exactly two occurrences of u, one as a prefix and one as a suffix. A factor of $\mathbf{w}$ is *recurrent* if and only if it occurs infinitely many times in $\mathbf{w}$. Each return word v of u in $\mathbf{w}$ corresponds to a path of length $|v|$ starting from u in $\mathcal{G}_{\mathbf{w}}(|u|)$ (but not all such paths induce return words). The introduction of return words to study quasiperiodicity stems from the following lemma.

▶ **Lemma 4.4.** *[10, Lem. 4.3] A finite word* v *is a quasiperiod of an infinite word* $\mathbf{w}$ *if and only if* v *is a recurrent prefix of* $\mathbf{w}$ *such that any return to* v *in* $\mathbf{w}$ *has length at most* $|v|$.

The graphs of words of Sturmian words are well-known since works from Arnoux and Rauzy [2]. We exploit this information to characterize quasiperiods of Sturmian words.

Let $\mathbf{w}$ be a Sturmian word. It has exactly one left special factor and one right special factor of each length. Let $\ell_n(\mathbf{w})$ and $r_n(\mathbf{w})$ denote respectively the left and right special factors of length n of $\mathbf{w}$. Since $\mathbf{w}$ is on a binary alphabet, $\ell_n(\mathbf{w})$ has exactly two incoming edges and all other vertices have only one incoming edge. Likewise, $r_n(\mathbf{w})$ has exactly two outgoing edges and all other vertices have only one outgoing edge. There are only two possible shapes for such a graph. If $r_n(\mathbf{w}) = \ell_n(\mathbf{w})$ then $G_n(\mathbf{w})$ is the union of two edge-disjoint paths who only share one vertex, $r_n(\mathbf{w})$. Otherwise, $G_n(\mathbf{w})$ is the union of three edge-disjoint paths, one from $\ell_n(\mathbf{w})$ to $r_n(\mathbf{w})$ and two from $r_n(\mathbf{w})$ to $\ell_n(\mathbf{w})$. These paths do not share vertices other than $\ell_n(\mathbf{w})$ and $r_n(\mathbf{w})$.

The path going from $\ell_n(\mathbf{w})$ to $r_n(\mathbf{w})$, and which might be empty if these two vertices are equal, is called the *special path.* The other two paths are called the *short path* and the *long path*, according to their respective lengths. If both are of the same length, we arbitrarily choose which one is the short path (this does not matter). Although the special path might be of length 0, the short and the long path have always at least 1 edge.

The length of a return word to $\ell_n(\mathbf{w})$ is the sum of the length of the special path and of one of the short or long paths. As a Sturmian word has $n+2$ factors of length $n+1$, the graph $G_n(\mathbf{w})$ has $n+2$ edges (recall that each edge corresponds to a factor of length $n+1$). Lemma 4.4 implies that $\ell_n(\mathbf{w})$ is a quasiperiod of $\mathbf{w}$ if and only if the short path of $G_n(\mathbf{w})$ is not of length 1 and the special path is not empty.

This situation is well-known; see for instance the description of evolution of graphs of words in [2]. It occurs exactly for integers n such that $r_{n-1}(\mathbf{w}) = \ell_{n-1}(\mathbf{w})$. This ends the proof of Proposition 4.3.

4.3 Standard Sturmian words : a new characterization

The converse of Proposition 4.3 holds and allows to provide the following new characterization of standard Sturmian words.

▶ **Theorem 4.5.** *Let* $\mathbf{w}$ *be an aperiodic word. The word* $\mathbf{w}$ *is standard Sturmian if and only if it is multiscale quasiperiodic and satisfies the following condition: for each positive integer* n*, the prefix of length* n *of* $\mathbf{w}$ *is a quasiperiod if and only if the prefix of length* $n-1$ *is not right special.*

Proof. The “only if” part corresponds to Proposition 4.3. Let us prove the “if” part. Let $\mathbf{w}$ be an aperiodic word such that, for each $n > 0$, the prefix of length n of $\mathbf{w}$ is a quasiperiod if and only if the prefix of length $n-1$ is not right special. Let a be the first letter of $\mathbf{w}$ and let $B = \mathrm{alph}(\mathbf{w}) \setminus \{a\}$ with $\mathrm{alph}(\mathbf{w})$ the set of letters occurring in $\mathbf{w}$. The size of $\mathrm{alph}(\mathbf{w})$ may be arbitrary, but is at least two (so B is not empty) since $\mathbf{w}$ is not periodic.

Step 1: The word $\mathbf{w}$ has no factor in B^* of length at least 2.

First, observe that factors of $\mathbf{w}$ belonging to B^* have bounded length. Indeed, $\mathbf{w}$ is quasiperiodic and any quasiperiod contains occurrences of the letter a and of letters from B. Hence lengths of factors belonging to B^* are bounded by the length of the smallest quasiperiod. Let x be a factor of $\mathbf{w}$ of maximal length among all factors belonging to B^*. As B is not empty, $|x| \geq 1$. Let p be the smallest prefix of $\mathbf{w}$ ending with x. By maximality in the definition of x, p is not right special: it is always followed by the letter a. Thus by hypothesis on $\mathbf{w}$, pa is a quasiperiod of $\mathbf{w}$.

By definition, p begins with a and ends with x. By maximality in the definition of x, p ends with ax and, by construction, does not contain any other occurrence of x. It follows that borders of pa are the words ε and a. Thus $\mathbf{w} \in p\{p, ap\}^\omega$.

Let π be the prefix of $\mathbf{w}$ of length $|p|-1$ and let b be the last letter of p. We have $p = \pi b$ and $\mathbf{w} \in \pi b\{\pi b, a\pi b\}^\omega$. As x has only one occurrence in p as a suffix, it has no occurrence in π. Moreover a is the first letter of π. Assume that π is not right special. By hypothesis, it follows that p is a quasiperiod of $\mathbf{w}$. By the choice on x and definition of p, p cannot be an internal factor of pap. Thus $\mathbf{w} = p^\omega$: a contradiction with aperiodicity of $\mathbf{w}$. Thus π is right special. There exists a letter c different from b such that πc is a factor of $\mathbf{w}$. This word πc occurs in a factor $a\pi$. Hence $a\pi = \pi c$ which implies $a = c$ and π is a power of a. Thus $|x| = 1$.

First corollary of Step 1: There exists an aperiodic word $\mathbf{w}'$ such that $\mathbf{w} = L_a(\mathbf{w}')$, where L_a is the morphism defined by $L_a(a) = a$, $L_a(x) = ax$ for any letter $x \neq a$.

The existence of $\mathbf{w}'$ is a reformulation of the result of Step 1. Aperiodicity of $\mathbf{w}'$ is a consequence of aperiodicity of $\mathbf{w}$.

Second corollary of Step 1: $\mathbf{w}$ is a binary word.

Indeed assume that $\mathbf{w}$ contains at least three different letters a (its first letter), b and c, with the first occurrence of b occurring before the first occurrence of c. Any quasiperiod of $\mathbf{w}$ must contain b and c. By Step 1, b is always followed by the letter a. Let π be smallest prefix of $\mathbf{w}$ ending with b. The word πa is a prefix of $\mathbf{w}$. As it does not contains c, πa cannot be a quasiperiod of $\mathbf{w}$. By the properties of $\mathbf{w}$, π is right special. This contradicts the fact that b is not right special.

Step 2: The smallest quasiperiod of $\mathbf{w}$ is its prefix $a^k ba$ ($k \geq 1$).

Indeed by Step 1 (and its second corollary), each occurrence of b is followed by the letter a. In particular the prefix $a^k b$ is not right special. By the properties of $\mathbf{w}$, $a^k ba$ is a quasiperiod of $\mathbf{w}$. As $\mathbf{w}$ is aperiodic, $a^k b$ is not a quasiperiod of $\mathbf{w}$. Clearly $\mathbf{w}$ has no quasiperiod that are powers of the letter a. Hence $a^k ba$ is the smallest quasiperiod of $\mathbf{w}$.

Step 3: For each integer n, the prefix of $\mathbf{w}'$ of length $n+1$ is a quasiperiod of $\mathbf{w}'$ if and only if the prefix of $\mathbf{w}'$ of length n is not right special.

Let p be a prefix of $\mathbf{w}'$ and c be the letter such that pc is a prefix of $\mathbf{w}'$. We have to prove that pc is a quasiperiod of $\mathbf{w}'$ if and only if p is not right special in $\mathbf{w}'$. This is a direct consequence of the next four properties (the third one is an hypothesis on $\mathbf{w}$, the proof of the others are omitted by lack of place):

1. Let $\mathbf{x} \in \{a, b\}^\omega$. A word q is a quasiperiod of $\mathbf{x}$ if and only if both words $L_a(q)$ and $L_a(q)a$ are quasiperiods of $L_a(\mathbf{x})$.
2. $L_a(p)ac$ is a quasiperiod of $\mathbf{w}$ if and only if $L_a(pc)$ and $L_a(pc)a$ are quasiperiods of $\mathbf{w}$.
3. $L_a(p)ac$ is a quasiperiod of $\mathbf{w}$ if and only if $L_a(p)a$ is not right special in $\mathbf{w}$.
4. Let $\mathbf{x} \in \{a, b\}^\omega$. A word u is right special in $\mathbf{x}$ if and only if $L_a(u)a$ is right special in $L_a(\mathbf{x})$.

Step 4: $\mathbf{w}'$ is multi-scale quasiperiodic.

By Step 2, a^kba is a quasiperiod of $\mathbf{w}$. Hence $\mathbf{w} \in a^kb\{a^kb, a^{k+1}b\}^\omega$. As $\mathbf{w}$ is aperiodic, there exists an integer $i \geq 1$ such that $(a^kb)^ia^{k+1}b$ is a prefix of $\mathbf{w}$. Let j be the greatest integer such that of $((a^kb)^ia^{k+1}b)^j$ is a factor of $\mathbf{w}$ (aperiodicity of $\mathbf{w}$ implies the existence of j). Let p be any prefix of $\mathbf{w}$ beginning with $(a^kb)^ia^{k+1}b$ and ending with $((a^kb)^ia^{k+1}b)^{j-1}(a^kb)^ia^{k+1}$ (multiscale quasiperiodicity of $\mathbf{w}$ implies there exist infinitely many such p). As a^{k+2} is not a factor of $\mathbf{w}$, each occurrence of p in $\mathbf{w}$ is always followed by the letter b. Hence p is not right special and pb is a quasiperiod of $\mathbf{w}$. By maximality of j, $pbpb$ is not a factor of $\mathbf{w}$. Thus two consecutive occurrences of pb must overlap by a factor at least as long as $(a^kb)^ia^{k+1}b$. Let p' be the unique (by the properties of L_a) word such that $L_a(p') = pb$. The word p' is a quasiperiod of $\mathbf{w}'$. As there are infinitely many possible words p, and so p', $\mathbf{w}'$ is multiscale quasiperiodic.

Conclusion. We have proven that:

- $\mathbf{w}$ is a binary word (let $\{a, b\}$ be the alphabet of $\mathbf{w}$);
- for an aperiodic multi-scale quasiperiodic word $\mathbf{w}'$ and a letter x, $\mathbf{w} = L_x(\mathbf{w}')$;
- the word $\mathbf{w}'$ satisfies the condition which links its quasiperiods and its right special factors, like $\mathbf{w}$.

Hence we can iterate this argument on $\mathbf{w}'$ and so on. Thus $\mathbf{w}$ is $\{L_a, L_b\}$-adic, that is, there exists an infinite sequence $(s_i)_{i\geq 0}$ and an infinite sequence of letters $(\alpha_i)_{i\geq 1}$ such that $s_0 = \mathbf{w}$ and, for $i \geq 1$, $s_{i-1} = L_{\alpha_i}(s_i)$. By [13, Cor. 2.7], $\mathbf{w}$ is standard episturmian. As it is a binary word, $\mathbf{w}$ is Sturmian. ◀

5 Conclusion

Proposition 2.1 and Theorem 4.5 show that multiscale quasiperiodicity is an interesting combinatorial notion as it allows to characterize some families of right infinite words. These characterizations can be extended to biinfinite words. For instance a biinfinite word is Sturmian if and only if it is multiscale quasiperiodic and satisfies: for each positive integer n, $\mathbf{w}$ has a quasiperiod of length n if and only if $\mathbf{w}$ has no bispecial factor of length $n-1$. Nevertheless, the structure of sets of quasiperiods of biinfinite words still needs to be studied, because it is more complex as there may exist several quasiperiods having the same length.

Another important problem is left open at the end of Section 3.2. What is the exact link between desubstitution and multiscale quasiperiodicity? Can any multiscale quasiperiodic word $\mathbf{w}$ be desubstituted into another multiscale quasiperiodic word? If the answer is negative, what additional conditions does this property imply?

References

1 A. Apostolico and A. Ehrenfeucht. Efficient detection of quasiperiodicities in strings. *Theor. Comput. Sci.*, 119:247–265, 1993.

2 P. Arnoux and G. Rauzy. Représentation géométrique de suites de complexité $2n+1$. *Bull. Soc. Math. France*, 119:199–215, 1991.

3 J. Berstel. Fibonacci words - a survey. In *The book of L*. Springer-Verlag, 1986.

4 V. Berthé and M. Rigo, editors. *Combinatorics, Automata and Number Theory.* Number 135 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2010.

5 G. S. Brodal and C. N. S. Pedersen. Finding maximal quasiperiodicities in strings. In *Combinatorial Pattern Matching (CPM'2000), 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-23, 2000*, volume 1848 of *Lect. Notes in Comput. Science*, pages 397–411, 2000.

6 M. Christou, M. Crochemore, and I. Costas S. Quasiperiodicities in Fibonacci strings. Technical Report 1201.6162, ArXiv, 2002 (To appear in Ars Combinatoria).

7 A. de Luca. Sturmian words: structure, combinatorics, and their arithmetics. *Theor. Comput. Sci.*, 183:45–82, 1997.

8 S. Fischler. Palindromic prefixes and episturmian words. *J. Combin. Theory Ser. A*, 113(7):1281–1304, 2006.

9 G. Gamard and G. Richomme. Coverability in two dimensions. In A. Horia Dediu, E. Formenti, C. Martín-Vide, and B. Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lect. Notes in Comput. Science*, pages 402–413. Springer, 2015.

10 A. Glen, F. Levé, and G. Richomme. Quasiperiodic and Lyndon episturmian words. *Theor. Comput. Sci.*, 409(3):578–600, 2008.

11 R. Groult and G. Richomme. Optimality of some algorithms to detect quasiperiodicities. *Theoretical Computer Science*, 411:3110 – 3122, 2010.

12 C. S. Iliopoulos and L. Mouchard. An $o(n \log n)$ algorithm for computing all maximal quasiperiodicities in strings. In C. S. Calude and M. J. Dinneen, editors, *Combinatorics, Computation and Logic. Proceedings of DMTCS'99 and CATS'99*, Lect. Notes in Comput. Science, pages 262–272, Auckland, New-Zealand, 1999. Springer.

13 J. Justin and G. Pirillo. Episturmian words and episturmian morphisms. *Theor. Comput. Sci.*, 276(1-2):281–313, 2002.

14 F. Levé and G. Richomme. Quasiperiodic infinite words: some answers. *Bull. Europ. Assoc. Theoret. Comput. Sci.*, 84:128–238, 2004.

15 F. Levé and G. Richomme. Quasiperiodic Sturmian words and morphisms. *Theor. Comput. Sci.*, 372(1):15–25, 2007.

16 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.

17 M. Lothaire. *Algebraic Combinatorics on Words*, volume 90 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 2002.

18 S. Marcus. Quasiperiodic infinite words. *Bull. Eur. Assoc. Theor. Comput. Sci.*, 82:170–174, 2004.

19 T. Monteil and S. Marcus. Quasiperiodic infinite words: multi-scale case and dynamical properties. arXiv:math/0603354v1, 2006.

20 L. Mouchard. Normal forms of quasiperiodic strings. *Theor. Comput. Sci.*, 249:313–324, 2000.

21 H. Mousavi, L. Schaeffer, and J. Shallit. Decision algorithms for Fibonacci-automatic words, I: basic results. *RAIRO Theor. Inform. Appl.*, To appear.

On the Complexity Landscape of Connected f-Factor Problems[*]

Robert Ganian[1], N. S. Narayanaswamy[2], Sebastian Ordyniak[3], C. S. Rahul[4], and M. S. Ramanujan[5]

1 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
`rganian@gmail.com`
2 **Indian Institute of Technology Madras, Chennai, India**
`swamy@cse.iitm.ac.in`
3 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
`sordyniak@gmail.com`
4 **Indian Institute of Technology Madras, Chennai, India**
`rahulcs@cse.iitm.ac.in`
5 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
`ramanujan@ac.tuwien.ac.at`

Abstract

Given an n-vertex graph G and a function $f : V(G) \to \{0, \dots, n-1\}$, an f-factor is a subgraph H of G such that $deg_H(v) = f(v)$ for every vertex $v \in V(G)$; we say that H is a *connected f-factor* if, in addition, the subgraph H is connected. A classical result of Tutte (1954) is the polynomial time algorithm to check whether a given graph has a specified f-factor. However, checking for the presence of a *connected* f-factor is easily seen to generalize Hamiltonian Cycle and hence is NP-complete. In fact, the Connected f-Factor problem remains NP-complete even when $f(v)$ is at least n^ϵ for each vertex v and $\epsilon < 1$; on the other side of the spectrum, the problem was known to be polynomial-time solvable when $f(v)$ is at least $\frac{n}{3}$ for every vertex v.

In this paper, we extend this line of work and obtain new complexity results based on restricting the function f. In particular, we show that when $f(v)$ is required to be at least $\frac{n}{(\log n)^c}$, the problem can be solved in quasi-polynomial time in general and in randomized polynomial time if $c \leq 1$. We also show that when $c > 1$, the problem is NP-intermediate.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.2 Graph Theory

Keywords and phrases f-factors, connected f-factors, quasi-polynomial time algorithms, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.41

1 Introduction

f-factors are a fundamental concept of graph theory dating back to the 19th century, specifically to the work of Petersen [13]. In modern terminology, an f-factor is defined as a spanning subgraph which satisfies degree constraints (given in terms of the degree function f) placed on each vertex of the graph [22]. Some of the most fundamental results on f-factors were obtained by Tutte, who gave sufficient and necessary conditions for the existence of f-factors [19]. Tutte also developed a method for reducing the existence of an f-factor

[*] The authors acknowledge support by the Austrian Science Fund (FWF, project P26696). Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.

41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016).
Editors: Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier; Article No. 41; pp. 41:1–41:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 The table depicting the known as well as new results in the complexity landscape of theConnected f-Factor problem.

$f(v) \geq$	Complexity Class
$n^\epsilon, \forall \epsilon > 0$	NPC [4]
$\frac{n}{\texttt{polylog}n}$	QP (Theorem 2)
$\frac{n}{\log n}$	RP (Theorem 3)
$\frac{n}{c}, \forall c \geq 3$	P (Theorem 1)

to the existence of a perfect matching [18], which gives a straightforward polynomial time algorithm for the problem of deciding whether an f-factor exists. There are also several detailed surveys on f-factors of graphs, for instance by Chung and Graham [3], Akiyama and Kano [1].

Aside from work on general f-factors, substantial attention has been devoted to the variant of f-factors where we require the subgraph to be connected (see for instance the survey articles by Kouider and Vestergaard [20] and Plummer [16]). Unlike the general variant, deciding the existence of a connected f-factor (the Connected f-Factor problem) is an NP-complete problem [6, 2]. It is easy to see that Connected f-Factor generalizes Hamiltonian Cycle (set $f(v) = 2$ for every vertex v), and even the existence of a deterministic single-exponential (in the number of vertices) algorithm is open for the problem [15].

The NP-completeness of this problem has motivated several authors to study the Connected f-Factor problem for various restrictions of the function f. Cornelissen et al. [4] showed that Connected f-Factor remains NP-complete even when $f(v)$ is at least n^ϵ for each vertex v and any constant ϵ between 0 and 1. At the other end of the spectrum, it has been shown that the problem is polynomial-time solvable when $f(v)$ is at least $\frac{n}{3}$ [12] for every vertex v. Aside from these two fairly extreme cases, the complexity landscape of this problem based on lower bounds on the function f has largely been left uncharted.

Our results and techniques. In this paper, we provide new results for solving the Connected f-Factor problem based on lower bounds on the range of f, both positive and negative. Since we will study the complexity landscape of Connected f-Factor through the lens of the function f, it will be useful to formally capture bounds on the function f via an additional "bounding" function g. To this end, we introduce the Connected g-Bounded f-Factor problem below:

Connected g-Bounded f-Factor
Instance: An n-vertex graph G and a mapping $f : V \to \mathbb{N}$ such that $f(v) \geq \frac{n}{g(n)}$.
Task: Find a connected subgraph H of G such that each vertex $v \in V(G)$ satisfies $d_H(v) = f(v)$.

First, we obtain a polynomial time algorithm for Connected f-Factor when $f(v)$ is at least $\frac{n}{c}$ for every vertex v and any constant $c \geq 1$. This result improves upon the previously known polynomial-time algorithm for the case when $f(v)$ is at least $\frac{n}{3}$. This is achieved thanks to a novel approach for the problem, which introduces a natural way of converting one f-factor to another by exchanging a set of edges. Here we formalize this idea using the notion of *Alternating Circuits*. These allow us to focus on a simpler version of the problem, where we merely need to ensure connectedness across a coarse partition of the vertex set. Furthermore, we extend this approach to obtain a quasi-polynomial time algorithm for the CONNECTED f-FACTOR problem when $f(v)$ is at least $\frac{n}{polylog(n)}$. To be precise, we prove the following two theorems (see the next page for an explanation of the function g used in the formal statements).

▶ **Theorem 1.** *For every function $g(n) = \mathcal{O}(1)$,* CONNECTED g-BOUNDED f-FACTOR *can be solved in polynomial time.*

▶ **Theorem 2.** *For every $c > 0$ and function $g(n) = \mathcal{O}((\log n)^c)$,* CONNECTED g-BOUNDED f-FACTOR *can be solved in time $n^{(\log n)^{\mathcal{O}(1)}}$.*

Second, we build upon these new techniques to obtain a *randomized polynomial-time* algorithm which solves CONNECTED f-FACTOR in the more general case when $f(v)$ is lower-bounded by $\frac{n}{\mathcal{O}(\log n)}$ for every vertex v. For this, we also require algebraic techniques that have found several applications in the design of fixed-parameter and exact algorithms for similar problems [5, 21, 7, 14]. Precisely, we prove the following theorem.

▶ **Theorem 3.** *For every function $g(n) = \mathcal{O}(\log n)$,* CONNECTED g-BOUNDED f-FACTOR *can be solved in polynomial time with constant error probability.*

We remark that the randomized algorithm in the above theorem has one-sided error with 'Yes' answers always being correct. Finally, we also obtain a lower bound result for CONNECTED f-FACTOR when $f(n)$ is at least $\frac{n}{(\log n)^c}$ for $c > 1$. Specifically, in this case we show that the problem is in fact NP-intermediate, assuming the Exponential Time Hypothesis [8] holds. Formally speaking, we prove the following theorem.

▶ **Theorem 4.** *For every $c > 1$, for every $g(n) = \Theta((\log n)^c)$,* CONNECTED g-BOUNDED f-FACTOR *is neither in* P *nor* NP*-hard unless the Exponential Time Hypothesis fails.*

We detail the known as well as new results on the complexity landscape of CONNECTED f-FACTOR in Table 1.

Organization of the paper. After presenting required definitions and preliminaries in Section 2, we proceed to the key technique and framework used for our algorithmic results, which forms the main part of Section 3. In the next Section 3.2, we obtain both of our deterministic algorithms, which are formally given as Theorem 1 (for the polynomial-time algorithm) and Theorem 2 (for the quasipolynomial-time algorithm). Section 4 then concentrates on our randomized polynomial-time algorithm, presented in Theorem 3. Finally, Section 5 focuses on ruling out (under established complexity assumptions) both NP-completeness and inclusion in P for all polylogarithmic functions g where we do not already have a polynomial-time algorithm.

2 Preliminaries

2.1 Basic Definitions and Graphs

We use standard definitions and notations from West [22]. $d_G(v)$ denotes the **degree** of a vertex v in a graph G. A **component** in a graph is a maximal subgraph that is connected. Note that the set of components in a graph uniquely determines a partition of the vertex set. A **circuit** in a graph is a cyclic sequence $v_0, e_1, v_1, \cdots, e_k, v_k = v_0$ where each e_i is of the form $\{v_{i-1}, v_i\}$ and occurs at most once in the sequence. An **Eulerian circuit** in a graph is a circuit in which each edge in the graph appears exactly once.

Let V' be a subset of the vertices in the graph G. The **induced subgraph** $G[V']$ is the graph over vertex set V' containing all the edges in G whose endpoints are both in V'.

Given a partition $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_r\}$ of the vertex set of G, the graph $G/\mathcal{Q}$ is constructed as follows: The vertex set of $G/\mathcal{Q}$ is $\mathcal{Q}$. Corresponding to each edge (u, v) in G where u in Q_i, v in Q_j, $i \neq j$, there exists an edge (Q_i, Q_j) in $G/\mathcal{Q}$. Thus, $G/\mathcal{Q}$ is a multigraph without loops. For a subgraph G' of G, we say G' **connects** a partition $\mathcal{Q}$ if $G'/\mathcal{Q}$ is connected. A **refinement** $\mathcal{Q}'$ of a partition $\mathcal{Q}$ is a partition of V where each part Q' in $\mathcal{Q}'$ is a subset of some part Q in $\mathcal{Q}$. This notion of partition refinement was used, e.g., by Kaiser [9]. A **spanning tree of** $G/\mathcal{Q}$ refers to a subgraph T of G with $|\mathcal{Q}|$-1 edges that connects $\mathcal{Q}$.

2.2 Colored Graphs, (Minimal) Alternating Circuits, and f-Factors

Recall the definition of the Connected g-Bounded f-Factor problem. Given an instance (G, f) of Connected g-Bounded f-Factor, a subgraph H of G is an *f-factor* if $d_H(v) = f(v)$ for each $v \in V(G)$; sometimes we also use the term f-factor to refer to $E(H)$.

A colored graph G is one in which each edge is assigned a color from the set $\{red, blue\}$. In a colored graph G, we use R and B to denote subgraphs of G whose edges are the set of red edges ($E(R)$) and blue edges ($E(B)$) of G, respectively, and $V(R) = V(B) = V(G)$. We will use this coloring in our algorithm to distinguish between edge sets of two distinct f-factors of the same graph G. A crucial computational step in our algorithms is to consider the symmetric difference between edge sets of two distinct f-factors and perform a sequence of edge exchanges preserving the degree of each vertex. The following definition is used extensively in our algorithms.

▶ **Definition 5.** A colored graph A is called an **alternating circuit** if there exists an Eulerian circuit in A where every pair of consecutive edges are of different colors.

Clearly, an alternating circuit has an even number of edges and is connected. Furthermore, $d_R(v) = d_B(v)$ for each v in A, where $d_R(v)$ and $d_B(v)$ denote the number of red and blue edges incident to v, respectively. A **minimal alternating circuit** M is an alternating circuit where each vertex v in M has at most two red edges incident to v. By definition, an empty graph is an alternating circuit.

We will also use the following lemma.

▶ **Lemma 6** ([12, Lemma 5]). *Let $S \subseteq E(G)$. An f-factor H containing all the edges in S, if one exists, can be computed in polynomial time.*

3 A Generic Algorithm for Finding Connected g-Bounded f-Factors

Our goal in this section is to present a generic algorithm for Connected g-Bounded f-Factor. In particular, we will in a certain sense reduce the question of solving Connected g-Bounded f-Factor to solving a related problem which we call Partition Connector. This can be viewed as a relaxed version of the original problem, since instead of a connected f-factor it merely asks for an f-factor which connects a specified partitioning of the vertex set. A formal definition is provided below.

PARTITION CONNECTOR
Instance: An n-vertex graph G, $f : V \to \mathbb{N}$, and a partition $\mathcal{Q}$ of $V(G)$.
Task: Find an f-factor of G that connects $\mathcal{Q}$.

The algorithms for solving PARTITION CONNECTOR will then be presented in the later parts of this article—specifically, a deterministic algorithm that runs in quasipolynomial time whenever $g(n) = \mathcal{O}(\texttt{polylog}(n))$ (Section 3.2) and a randomized polynomial-time algorithm for the case when $g(n) = \mathcal{O}(\log n)$ (Section 4).

The majority of this section is devoted to proving the key Theorem 7 stated below, which establishes the link between PARTITION CONNECTOR and CONNECTED g-BOUNDED f-FACTOR.

▶ **Theorem 7.** **(a)** *Let $g(n) = \mathcal{O}(\texttt{polylog}(n))$. If there is a deterministic algorithm running in time $\mathcal{O}(n^{2(|\mathcal{Q}|-1)})$ for* PARTITION CONNECTOR*, then there is a deterministic quasi-polynomial time algorithm for* CONNECTED g-BOUNDED f-FACTOR *with running time $\mathcal{O}(g(n) \cdot n^{2g(n)})$.*
(b) *Let $g(n) = \mathcal{O}(\log n)$. If there is a randomized algorithm running in time $\mathcal{O}(2^{|\mathcal{Q}|}n^{\mathcal{O}(1)})$ with error probability $\mathcal{O}(|\mathcal{Q}|^2/n^2)$ for* PARTITION CONNECTOR*, then there is a randomized polynomial-time algorithm for* CONNECTED g-BOUNDED f-FACTOR *that has a constant error probability.*

3.1 A generic algorithm for Connected g-Bounded f-Factor

The starting point of our generic algorithm is the following observation.

▶ **Observation 8.** Let G be an undirected graph and f be a function $f : V \to \mathbb{N}$. The graph G has a connected f-factor if and only if for each partition $\mathcal{Q}$ of the vertex set V, there exists an f-factor H of G that connects $\mathcal{Q}$.

We remark that for the running time analysis for our generic algorithm we will assume that we are only dealing with instances of CONNECTED g-BOUNDED f-FACTOR, where the number of vertices exceeds $6g(n)^4$. As $g(n)$ is in $\mathcal{O}(\texttt{polylog}(n))$, this does not reduce the applicability of our algorithms, since there is a constant n_0 such that $n \geq 6g(n)^4$ for every $n \geq n_0$; because $g(n)$ is part of the problem description, n_0 does not depend on the input instance. Consequently, we can solve instances of CONNECTED g-BOUNDED f-FACTOR where $n < n_0$ by brute-force in constant time. We will therefore assume without loss of generality in the following that $n \geq n_0$ and hence $n \geq 6g(n)^4$.

Our algorithm constructs a *maximal* sequence of pairs $(H_0, \mathcal{Q}_0), \ldots, (H_k, \mathcal{Q}_k)$ satisfying the following properties:

(M1) Each $\mathcal{Q}_i, 0 \leq i \leq k$ is a partition of the vertex set V, and $\mathcal{Q}_0 = \{V(G)\}$.
(M2) Each $H_i, 0 \leq i \leq k$ is an f-factor of G, and H_i connects $\mathcal{Q}_i$.
(M3) For each $1 \leq i \leq k$, $\mathcal{Q}_i$ is a refinement of $\mathcal{Q}_{i-1}$ satisfying the following:
  (a) Each part Y in $\mathcal{Q}_i$ induces a component $H_{i-1}[Y]$ in $H_{i-1}[Q]$, for some Q in $\mathcal{Q}_{i-1}$.
  (b) $\mathcal{Q}_i \neq \mathcal{Q}_{i-1}$.

The following lemma links the existence of a connected f-factor to the properties of maximal sequences satisfying (M1)–(M3).

▶ **Lemma 9.** *Let (G, f) be an instance of* CONNECTED g-BOUNDED f-FACTOR *and let $(H_0, \mathcal{Q}_0), \ldots, (H_k, \mathcal{Q}_k)$ be any maximal sequence satisfying Properties (M1)–(M3). Then G has a connected f-factor if and only if H_k is a connected f-factor of G.*

The above lemma shows that if we had an algorithm for constructing a maximal sequence satisfying (M1)–(M3), then we could solve the f-factor problem by testing whether the last f-factor in that sequence is connected. However, if the number of parts of the partitions $\mathcal{Q}_i$ is allowed to grow to n, then such an algorithm would eventually have to solve the connected f-factor problem. Hence, to employ the idea for an efficient algorithm, we first need to establish an upper bound on the number of parts in any partition $\mathcal{Q}_i$ of a maximal sequence. The following lemma, which shows a lower bound on the minimum degree and hence a lower bound on the size of any part in $\mathcal{Q}_i$, is crucial for estabilishing such an upper bound.

▶ **Lemma 10.** *Let $(H, \mathcal{Q})$, $(H', \mathcal{Q}')$ be two consequitive pairs occuring in a sequence satisfying properties (M1)–(M3). Then there is an f-factor H'' of G connecting $\mathcal{Q}'$ such that $|N_{H''}(v) \cap Q'| \geq |N_H(v) \cap Q'| - 2(|\mathcal{Q}'| - 1)$ for every $Q' \in \mathcal{Q}'$ and $v \in Q'$. Moreover, H'' can be computed from $\mathcal{Q}'$, H, and H' in polynomial time.*

Our algorithm will employ the above lemma to construct a maximal sequence $(H_0, \mathcal{Q}_0)$, $\dots, (H_k, \mathcal{Q}_k)$ that satisfies the following additional property:

(M4) For every $1 \leq i \leq k$, every $Q \in \mathcal{Q}_i$ and $v \in Q$ it holds that $|N_{H_i}(v) \cap Q| \geq |N_{H_{i-1}}(v) \cap Q| - 2(|\mathcal{Q}_i| - 1)$.

This property will be key for the analysis of our algorithm because it allows us to bound the number of parts in each partition $\mathcal{Q}_i$. Towards this aim we require the following auxirilliary lemma.

▶ **Lemma 11.** *Let $\mathcal{S} = (H_0, \mathcal{Q}_0), \dots, (H_k, \mathcal{Q}_k)$ be a sequence satisfying Properties (M1)–(M4). Then $|N_{H_i}(v) \cap Q| \geq f(v) - \sum_{1 \leq j \leq i} 2(|\mathcal{Q}_j| - 1)$ for every i with $1 \leq i \leq k$, $Q \in \mathcal{Q}_i$ and $v \in Q$.*

We are now ready to show that the number of parts in any partition $\mathcal{Q}_i$ in a maximal sequence does not exceed $g(n) + 1$.

▶ **Lemma 12.** *Let $\mathcal{S} = (H_0, \mathcal{Q}_0), \dots, (H_k, \mathcal{Q}_k)$ be a maximal sequence satisfying Properties (M1)–(M4). Then $|\mathcal{Q}_i| \leq g(n) + 1$ for every i with $0 \leq i \leq k$. Moreover, the length of $\mathcal{S}$ is at most $g(n) + 1$.*

We are now ready to prove the main theorem of this section.

Sketch of Proof of Theorem 7. Let (G, f) be an instance of CONNECTED g-BOUNDED f-FACTOR. The algorithm constructs a maximal sequence satisfying Properties (M1)–(M4). The first pair $(H_0, \mathcal{Q}_0)$ is obtained by computing an arbitrary f-factor of G and setting $\mathcal{Q}_0 = \{V(G)\}$. The crucial ingredient of the algorithm is then a procedure that given the i-th pair $(H_i, \mathcal{Q}_i)$ of a maximal sequence computes a $(i+1)$-th pair $(H_{i+1}, \mathcal{Q}_{i+1})$ that can be appended to the sequence without violating any of the Properties (M1)–(M4). Informally, this is achieved by first setting $\mathcal{Q}_{i+1}$ to be the refinement of $\mathcal{Q}_i$ containing one part for every part $Q \in \mathcal{Q}_i$ and every component of $H_i[Q]$. Then an f-factor connecting $\mathcal{Q}_{i+1}$ is computed using the given algorithm for PARTITION CONNECTOR. If no such f-factor exists, the algorithm returns failure (which is correct due to Observation 8). If such an f-factor exists, the procedure uses the given f-factor and Lemma 10 to compute an f-factor H_{i+1} connecting $\mathcal{Q}_{i+1}$ such that the pair $(H_{i+1}, \mathcal{Q}_{i+1})$ satisfies Property (M4). Clearly, if any of the computed f-factors are connected f-factors of G the procedure returns the corresponding f-factor as a solution. Otherwise, the procedure now tries to find a successor of $(H_{i+1}, \mathcal{Q}_{i+1})$ in the already computed sequence satisfying (M1)–(M4). ◀

3.2 A Quasipolynomial Time Algorithm for Polylogarithmic Bounds

In this section, we prove Theorem 1 and Theorem 2. In fact, we prove a more general result, from which both theorems directly follow.

▶ **Theorem 13.** *For every $c > 0$ and function $g(n) = \mathcal{O}((\log n)^c)$, the* Connected g-Bounded f-factor *problem can be solved in $\tilde{\mathcal{O}}(n^{g(n)})$ time.*

We will make use of the following simple lemma.

▶ **Lemma 14.** *Let G be a graph having a connected f-factor. Let $\mathcal{Q}$ be a partition of the vertex set V. There exists a spanning tree T of $G/\mathcal{Q}$ such that for some f-factor H of G, $E(T) \subseteq E(H)$. Furthermore, H can be computed from T in polynomial time.*

Proof. Let G' be a connected f-factor of G. For any partition $\mathcal{Q}$ of the vertex set, it follows from Theorem 8 that $G'/\mathcal{Q}$ is connected. Consider a spanning tree T of $G'/\mathcal{Q}$. Clearly, there exists at least one f-factor H containing $E(T)$ and hence $H/\mathcal{Q}$ is connected. Once we have $E(T)$, H can be computed in polynomial time using Lemma 6. ◀

In light of Theorem 7, it now suffices to prove the following Lemma 15, from which Theorem 13 immediately follows.

▶ **Lemma 15.** Partition Connector *can be solved in time $\mathcal{O}(n^{2(|\mathcal{Q}|-1)})$.*

Proof. It follows from Lemma 14 that we can solve Partition Connector by going over all spanning trees T of $G/\mathcal{Q}$ and checking for each of them whether there is an f-factor of G containing the edges of T. The lemma now follows because the number of spanning trees of $G/\mathcal{Q}$ is at most $\binom{|E(G)|}{|\mathcal{Q}|-1}$, which is upper bounded by $\mathcal{O}(n^{2(|\mathcal{Q}|-1)})$, and for every such tree T we can check the existence of an f-factor containing T in polynomial time. ◀

4 A Randomized Polynomial Time Algorithm for Logarithmic Bounds

In this section we prove Theorem 3. Due to Theorem 7, it is sufficient for us to provide a randomized algorithm for Partition Connector with running time $\mathcal{O}(2^{|\mathcal{Q}|}n^{\mathcal{O}(1)})$ and error probability $\mathcal{O}(g(n)^2/n^2)$. This is precisely what we do in the rest of this section (Lemma 28). As a first step, we will design an algorithm for the "existential version" of the problem which we call ∃-Partition Connector and define as follows.

∃-Partition Connector
Input: A graph G with n vertices, $f : V \to \mathbb{N}$, and a partition $\mathcal{Q}$ of $V(G)$.
Question: Is there an f-factor of G that connects $\mathcal{Q}$?

We will then describe how to use our algorithm for this problem as a subroutine in our algorithm to solve Partition Connector.

4.1 Solving ∃-Partition Connector in Randomized Polynomial Time

The objective of this subsection is to prove the following lemma which implies a randomized polynomial time algorithm for ∃-Partition Connector when $g(n) = \mathcal{O}(\log n)$.

▶ **Lemma 16.** *There exists an algorithm that, given a graph G, a function $f : V \to \mathbb{N}$, and a partition $\mathcal{Q}$ of $V(G)$, runs in time $\mathcal{O}(2^{|\mathcal{Q}|}|V(G)|^{\mathcal{O}(1)})$ and outputs*

- NO *if G has no f-factor connecting $\mathcal{P}$*
- YES *with probability at least $1 - \frac{1}{n^2}$ otherwise.*

We design this algorithm by starting from the exact-exponential algorithm in [14] and making appropriate modifications. During the description, we will point out the main differences between our algorithm and that in [14]. We now proceed to the details of the algorithm. We begin by recalling a few important definitions and known results on f-factors. These are mostly standard and are also present in [14], but since they are required in the description and proof of correctness of our algorithm, we will state them here.

▶ **Definition 17** (f-Blowup)**.** Let G be a graph and let $f : V(G) \to \mathbb{N}$ be such that $f(v) \leq deg(v)$ for each $v \in V(G)$. Let H be the graph defined as follows

1. For each vertex v of G, we add a vertex set $A(v)$ of size $f(v)$ to H.
2. For each edge $e = \{v, w\}$ of G we add to H vertices v_e and w_e and edges (u, v_e) for every $u \in A(v)$ and (w_e, u) for every $u \in A(w)$. Finally, we add the edge (v_e, w_e).

This completes the construction. The graph H is called the *f-blowup* of graph G. We use $\mathcal{B}_f(G)$ to denote the f-blowup of G. We omit the subscript when there is no scope for ambiguity.

▶ **Definition 18** (Induced f-blowup)**.** For a subset $S \subseteq V(G)$, we define the f-blowup of G *induced* by S as follows. Let the f-blowup of G be H. Begin with the graph H and for every edge $e = (v, w) \in E(G)$ such that $v \in S$ and $w \notin S$, delete the vertices v_e and w_e. Let the graph H' be the union of those connected components of the resulting graph which contain the vertex sets $A(v)$ for vertices $v \in S$. Then, the graph H' is called the f-blowup of G *induced* by the set S and is denoted by $\mathcal{B}_f(G)[S]$.

We now recall the relation between perfect matchings in the f-blowup and f-factors (see Figure 1).

▶ **Lemma 19** ([10])**.** *A graph G has an f-factor if and only if the f-blowup of G has a perfect matching.*

The relationship between the Tutte matrix and perfect matchings is well-known and this has already been exploited in the design of fixed-parameter and exact algorithms [21, 7].

▶ **Definition 20** (Tutte matrix)**.** The *Tutte matrix* of a graph G with n vertices is an $n \times n$ skew-symmetric matrix T over the set $\{x_{ij} | 1 \leq i < j \leq |V(G)|\}$ of indeterminates whose $(i, j)^{th}$ element is defined to be

$$T(i,j) = \begin{cases} x_{ij} & \text{if } \{i,j\} \in E(G) \text{ and } i < j \\ -x_{ij} & \text{if } \{i,j\} \in E(G) \text{ and } i > j \\ 0 & \text{otherwise} \end{cases}$$

We use $\mathcal{T}(G)$ to denote the Tutte matrix of the graph G.

Following terminology in [14], when we refer to expanded forms of *succinct* representations (such as summations and determinants) of polynomials, we use the term *naive expansion* (or summation) to denote that expanded form of the polynomial which is obtained by merely writing out the operations indicated by the succinct representation. We use the term *simplified expansion* to denote the expanded form of the polynomial which results after we apply all possible simplifications (such as cancellations) to a naive expansion. We call a monomial m which has a non-zero coefficient in a simplified expansion of a polynomial P, a *surviving* monomial of P in the simplified expansion.

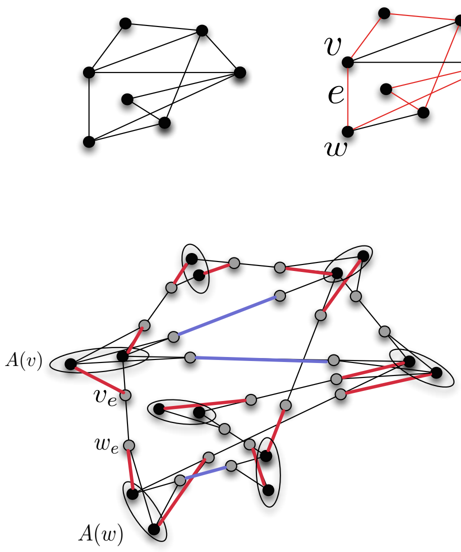


Figure 1 An illustration of a graph G with a 2-factor H (the red edges) and one possible corresponding perfect matching in $\mathcal{B}(G)$. It is important to note that an edge $e = (v, w)$ is *not* in H if and only if the edge (v_e, w_e) is present in the corresponding perfect matching.

The following basic facts about the Tutte matrix $\mathcal{T}(G)$ of a graph G are well-known. When evaluated over any field of characteristic two, the determinant and the permanent of the matrix $\mathcal{T}(G)$ (indeed, of any matrix) coincide. That is,

$$\det \mathcal{T}(G) = \mathrm{perm}(\mathcal{T}(G)) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} \mathcal{T}(G)(i, \sigma(i)), \qquad (1)$$

where S_n is the set of all *permutations* of $[n]$. Furthermore, there is a one-to-one correspondence between the set of all *perfect matchings* of the graph G and the *surviving monomials* in the above expression for $\det \mathcal{T}(G)$ when its simplified expansion is computed over any field of characteristic two.

▶ **Proposition 21** ([11])**.** If $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_\ell, j_\ell)\}$ is a perfect matching of a graph G, then the product $\prod_{(i_k, j_k) \in M} x_{i_k j_k}^2$ appears exactly once in the naive expansion and hence as a surviving monomial in the sum on the right-hand side of Equation 1 when this sum is expanded and simplified over any field of characteristic two. Conversely, each surviving monomial in a simplified expansion of this sum over a field of characteristic two is of the form $\prod_{(i_k, j_k) \in M} x_{i_k j_k}^2$ where $M = \{(i_1, j_1), (i_2, j_2), \ldots, (i_\ell, j_\ell)\}$ is a perfect matching of G. In particular, $\det \mathcal{T}(G)$ is identically zero when expanded and simplified over a field of characteristic two if and only if the graph G does not have a perfect matching.

▶ **Lemma 22** (Schwartz-Zippel Lemma, [17, 23])**.** *Let $P(x_1, \ldots, x_n)$ be a multivariate polynomial of degree at most d over a field $\mathbb{F}$ such that P is not identically zero. Furthermore, let $r_1, \ldots, r_n$ be chosen uniformly at random from $\mathbb{F}$. Then, Prob$[P(r_1, \ldots, r_n) = 0] \leq \frac{d}{|\mathbb{F}|}$.*

▶ **Definition 23.** For a partition of $V(G)$, $\mathcal{Q} = \{Q_1, \ldots, Q_\ell\}$ and a subset $I \subseteq [\ell]$, we denote by $\mathcal{Q}(I)$ the set $\bigcup_{i \in I} Q_i$. Furthermore, with every set $\emptyset \neq I \subset [\ell]$, we associate a specific monomial m_I which is defined to be the product of the terms x_{ij}^2 where $i < j$

and $\{i,j\} = \{v_e, w_e\}$, $e = (v,w) \in E(G)$ crosses the cut $(\mathcal{Q}(I), \overline{\mathcal{Q}(I)})$ and v_e, w_e, are as in Definition 17 of the f-blowup $\mathcal{B}(G)$ of G. For $I = [\ell]$, we define $m_I = 1$.

From now on, for a set $X \subseteq V(G)$, we denote by $\overline{X}$ the set $V(G) \setminus X$. Also, since we always deal with a fixed graph G and function f, for the sake of notational convenience, we refer to the graph $\mathcal{B}_f(G)$ simply as $\mathcal{B}$. We now define a polynomial $P_{\mathcal{Q}}(\bar{x})$ over the indeterminates from the Tutte matrix $\mathcal{T}(\mathcal{B})$ of the f-blowup of G, as follows:

$$P_{\mathcal{Q}}(\bar{x}) = \sum_{\{1\} \subseteq I \subseteq [\ell]} (\det \mathcal{T}(\mathcal{B}[\mathcal{Q}(I)])) \cdot (\det \mathcal{T}(\mathcal{B}[\overline{\mathcal{Q}(I)}])) \cdot m_I, \tag{2}$$

where if a graph H has no vertices or edges then we set $\det \mathcal{T}(H) = 1$. In future, we will always deal with a fixed partition $\mathcal{Q} = \{Q_1, \ldots, Q_\ell\}$ of $V(G)$.

▶ Remark. The definition of the polynomial $P_{\mathcal{Q}}(\bar{x})$ is the main difference between our algorithm and the algorithm in [14]. The rest of the details are identical. The main algorithmic consequence of this difference is the time it takes to evaluate this polynomial at a given set of points. This is captured in the following lemma whose proof follows from the fact that determinant computation is a polynomial time solvable problem.

▶ **Lemma 24.** *Given values for the variables x_{ij} in the matrix $\mathcal{T}(\mathcal{B})$, the polynomial $P_{\mathcal{Q}}(\bar{x})$ can be evaluated over a field $\mathbb{F}$ of character 2 and size $\Omega(n^6)$ in time $\mathcal{O}(2^\ell n^{\mathcal{O}(1)})$.*

Having shown that this polynomial can be efficiently evaluated, we will now turn to the way we use it in our algorithm. Our algorithm for $\exists$-Partition Connector takes as input $G, f, \mathcal{Q}$, evaluates the polynomial $P_{\mathcal{Q}}(\bar{x})$ at points chosen independently and uniformly at random from a field $\mathbb{F}$ of size $\Omega(n^6)$ and characteristic 2 and returns Yes if and only if the polynomial does not vanish at the chosen points. In what follows we will prove certain properties of this polynomial which will be used in the formal proof of correctness of this algorithm. We need another definition before we can state the main lemma capturing the properties of the polynomial. Recall that for every $v \in V(G)$, the set $A(v)$ is the set of 'copies' of v in the f-blowup of G. Furthermore, for a set $X \subseteq V(G)$, we say that an edge $e \in E(G)$ *crosses* the cut $(X, \overline{X})$ if e has exactly one endpoint in X.

▶ **Definition 25.** We say that an f-factor H of G *contributes* a monomial $x^2_{i_1 j_1} \ldots x^2_{i_r j_r}$ to the naive expansion of the right-hand side of Equation 2 if and only if the following conditions hold.

1. For every $e = (v,w) \in E(H)$, there is a $u \in A(v)$, $u' \in A(w)$ and $1 \le p, q \le r$ such that $\{u, v_e\} = \{i_p, j_p\}$ and $\{u', w_e\} = \{i_q, j_q\}$.
2. For every $e = (v,w) \in E(G) \setminus E(H)$, there is a $1 \le p \le r$ such that $\{v_e, w_e\} = \{i_p, j_p\}$.
3. For every $1 \le p, q \le r$, if $\{u, v_e\} = \{i_p, j_p\}$ and $\{u', w_e\} = \{i_q, j_q\}$ for some $e \in E(G)$, then $e \in E(H)$.
4. For every $1 \le p \le r$, if $\{i_p, j_p\} = \{v_e, w_e\}$ for some $e \in E(G)$, then $e \notin E(H)$.
5. For every $1 \in I \subseteq [\ell]$ such that H has no edge crossing the cut $(\mathcal{Q}(I), \overline{\mathcal{Q}(I)})$, there is a pair of monomials m_1 and m_2 such that m_1 is a surviving monomial in the simplified expansion of $\det \mathcal{T}(\mathcal{B}[\mathcal{Q}(I)])$, m_2 is a surviving monomial in the simplified expansion of $\det \mathcal{T}(\mathcal{B}[\overline{\mathcal{Q}(I)}])$, and $m_1 \cdot m_2 \cdot m_I = x^2_{i_1 j_1} \ldots x^2_{i_r j_r}$.

Having set up the required notation, we now state the main lemma which allows us to show that monomials contributed by f-factors that do not connect $\mathcal{Q}$, do not survive in the simplified expansion of the right hand side of Equation 2.

▶ **Lemma 26.** *Every monomial in the polynomial $P_{\mathcal{Q}}(\bar{x})$ which is a surviving monomial in the simplified expansion of the right-hand side of Equation 2 is contributed by an f-factor of G to the naive expansion of the right-hand size of Equation 2. Furthermore, for any f-factor of G, say H, the following statements hold.*

1. *If H does not connect $\mathcal{Q}$ then every monomial contributed by H occurs an* even *number of times in the polynomial $P_{\mathcal{Q}}(\bar{x})$ in the naive expansion of the right-hand side of Equation 2.*
2. *If H connects $\mathcal{Q}$, then every monomial contributed by H occurs exactly once in the polynomial $P_{\mathcal{Q}}(\bar{x})$ in the naive expansion of the right-hand side of Equation 2.*

This implies the following result, which is the last ingredient we need to prove Lemma 16.

▶ **Lemma 27.** *The polynomial $P_{\mathcal{Q}}(\bar{x})$ is not identically zero over $\mathbb{F}$ if and only if G has an f-factor connecting $\mathcal{Q}$.*

Proof of Lemma 16. It follows from the definition of $P(\bar{x})$ that its degree is $\mathcal{O}(n^4)$ since the number of vertices in the f-blowup of G is $\mathcal{O}(n^2)$. As mentioned earlier, our algorithm for ∃-Partition Connector takes as input $G, f, \mathcal{Q}$, evaluates the polynomial $P_{\mathcal{Q}}(\bar{x})$ at points chosen independently and uniformly at random from a field $\mathbb{F}$ of size $\Omega(n^6)$ and characteristic 2 and returns Yes if and only if the polynomial does not vanish at the chosen points. Due to Lemma 27, we know that the polynomial $P_{\mathcal{Q}}(\bar{x})$ is identically zero if and only if G has an f-factor containing $\mathcal{Q}$ and by the Schwartz-Zippel Lemma, the probability that the polynomial is not identically zero and still vanishes upon evaluation is at most $\frac{1}{n^2}$. This completes the proof of the lemma. ◀

Having obtained the algorithm for ∃-Partition Connector, we now return to the algorithm for the computational version, Partition Connector.

4.2 Solving Partition Connector in Randomized Polynomial Time

▶ **Lemma 28.** *The* Partition Connector *problem can be solved by a randomized algorithm with running time $\mathcal{O}(2^{|\mathcal{Q}|}n^{\mathcal{O}(1)})$ and error probability $\mathcal{O}(1-(1-\frac{1}{n^2})^{|\mathcal{Q}|})$.*

Sketch of Proof. Consider the following algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ takes as input an n-vertex instance of Partition Connector with the partition $\mathcal{Q} = \{Q_1, \ldots, Q_\ell\}$, along with a separate set of edges F which will store the edges that have been previously selected to be included in the partition connector. Let F be initialized as $\emptyset$. As its first step, Algorithm $\mathcal{A}$ checks if $\ell = 1$; if this is the case, then it computes an arbitrary f-factor H, and outputs $H \cup F$. To proceed, let us denote the algorithm of Lemma 16 as $\mathcal{B}$. If $\ell > 1$, then $\mathcal{A}$ first calls $\mathcal{B}$ and outputs NO if $\mathcal{B}$ outputs NO. Otherwise, it fixes an arbitrary ordering $E^{\leq}$ of the edge set E and recursively proceeds as follows.

$\mathcal{A}$ constructs the set E_1 of all edges with precisely one endpoint in Q_1, and loops over all edges in E_1 (in the ordering given by $E^{\leq}$). For each processed edge e between Q_1 and some Q_i with endpoints c and d, it will compute a subinstance $(G^e, f^e, \mathcal{Q}^e)$ defined by setting:

- $G^e = G - e$, and
- $f^e(c) = f(c) - 1$, $f^e(d) - f(d) - 1$ and $f^e = f$ for all other vertices of G, and
- $\mathcal{Q}^e$ is obtained from $\mathcal{Q}$ by merging Q_1 and Q_i into a new set; formally, $\mathcal{Q}^e = (\mathcal{Q} \setminus \{Q_1, Q_i\}) \cup \{Q_1 \cup Q_i\}$.

Intuitively, each such new instance corresponds to us forcing the f-factor to choose the edge e. $\mathcal{A}$ then queries $\mathcal{B}$ on $(G^e, f^e, \mathcal{Q}^e)$. If $\mathcal{B}$ answers NO for each such tuple $(G^e, f^e, \mathcal{Q}^e)$ obtained from each edge in E_1, then $\mathcal{A}$ immediately terminates and answers NO. Otherwise let e be the first edge where $\mathcal{B}$ answered YES; then $\mathcal{A}$ will add e into F. If $|\mathcal{Q}^e| = 1$ then the algorithm computes an arbitrary f-factor H of (G^e, f^e) and outputs $H \cup F$. On the other hand, if $|\mathcal{Q}| > 1$ then $\mathcal{A}$ restarts the recursive procedure with $(G, f, \mathcal{Q}) := (G^e, f^e, \mathcal{Q}^e)$; observe that $|\mathcal{Q}^e| \leq |\mathcal{Q}| - 1$. To complete the proof, it suffices to verify the correctness and the running time. ◀

5 Classification Results

In this section, we prove Theorem 4 which we restate for the sake of completeness.

▶ **Theorem 4.** *For every $c > 1$, for every $g(n) = \Theta((\log n)^c)$,* Connected g-Bounded f-Factor *is neither in* P *nor* NP*-hard unless the Exponential Time Hypothesis fails.*

The result relies on the established Exponential Time Hypothesis, which we recall below.

▶ **Definition 29** (Exponential Time Hypothesis (ETH), [8]). There exists a constant $s > 0$ such that 3-SAT with n variables and m clauses cannot be solved in time $2^{sn}(n+m)^{\mathcal{O}(1)}$.

We first show that the problem is not NP-hard unless the ETH fails. We remark that we can actually prove a stronger statement here by weakening the premise to "NP is not contained in Quasi-Polynomial Time". However, since we are only able to show the other part of Theorem 4 under the ETH, we phrase the statement in this way.

▶ **Lemma 30.** *For every $c > 1$, for every $g(n) = \Theta((\log n)^c)$,* Connected g-Bounded f-Factor *is not* NP*-hard unless the Exponential Time Hypothesis fails.*

Proof. Due to Theorem 2, we know that when $g(n) = \Theta((\log n)^c)$, Connected g-Bounded f-Factor can be solved in quasi-polynomial time. Hence, this problem cannot be NP-hard unless NP is contained in the complexity-class Quasi-Polynomial Time, QP. Furthermore, observe that $NP \subseteq QP$ implies that the ETH is false. Hence, we conclude that Connected g-Bounded f-Factor is not NP-hard unless the Exponential Time Hypothesis fails. ◀

Next, we use a reduction from Hamiltonian Cycle to obtain:

▶ **Lemma 31.** *For every $c > 1$, for every $g(n) = \Theta((\log n)^c)$,* Connected g-Bounded f-Factor *is not in* P *unless the Exponential Time Hypothesis fails.*

Lemmas 30 and 31 together give us Theorem 4.

6 Concluding remarks

We obtained new complexity results for Connected f-Factor with respect to lower bounds on the function f. As our main results, we showed that when $f(v)$ is required to be at least $\frac{n}{(\log n)^c}$, the problem can be solved in quasi-polynomial time in general and in randomized polynomial time if $c \leq 1$. Consequently, we show that the problem can be solved in polynomial-time when $f(v)$ is at least $\frac{n}{c}$ for any constant c. We complement the picture with matching classification results.

As a by-product we obtain a generic approach reducing Connected f-Factor to the "simpler" Partition Connector problem. Hence future algorithmic improvements of Partition Connector carry over to the Connected f-Factor problem. Finally, it would be interesting to investigate the possibility of derandomizing the polynomial-time algorithm for the case that $g(n) = \mathcal{O}(\log n)$.

Acknowledgments. The authors wish to thank the anonymous reviewers for their helpful comments.

References

1 Jin Akiyama and Mikio Kano. Factors and factorizations of graphs—a survey. *Journal of Graph Theory*, 9(1):1–42, 1985.

2 F. Cheah and D. G. Corneil. The complexity of regular subgraph recognition. *Discrete Applied Mathematics*, 27(1-2):59–68, 1990.

3 FRK Chung and RL Graham. Recent results in graph decompositions. *London Mathematical Society, Lecture Note Series*, 52:103–123, 1981.

4 Kamiel Cornelissen, Ruben Hoeksma, Bodo Manthey, N.S. Narayanaswamy, and C.S. Rahul. Approximability of connected factors. In Christos Kaklamanis and Kirk Pruhs, editors, *Approximation and Online Algorithms*, volume 8447 of *Lecture Notes in Computer Science*, pages 120–131. Springer International Publishing, 2014. `doi:10.1007/978-3-319-08001-7_11`.

5 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 *FOCS*, pages 150–159, 2011.

6 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

7 Gregory Gutin, Magnus Wahlström, and Anders Yeo. Parameterized rural postman and conjoining bipartite matching problems. *CoRR*, abs/1308.2599, 2013. URL: `http://arxiv.org/abs/1308.2599`.

8 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512 – 530, 2001. `doi:10.1006/jcss.2001.1774`.

9 Tomáš Kaiser. A short proof of the tree-packing theorem. *Discrete Mathematics*, 312(10):1689–1691, 2012.

10 László Lovász. The factorization of graphs. ii. *Acta Mathematica Academiae Scientiarum Hungarica*, 23(1-2):223–246, 1972.

11 László Lovász. On determinants, matchings, and random algorithms. In L. Budach, editor, *Fundamentals of Computation Theory FCT '79*, pages 565–574, Berlin, 1979. Akademie-Verlag.

12 NS Narayanaswamy and CS Rahul. Approximation and exact algorithms for special cases of connected f-factors. In *Computer Science–Theory and Applications*, pages 350–363. Springer, 2015.

13 Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15(1):193–220, 1891.

14 Geevarghese Philip and M. S. Ramanujan. Vertex exponential algorithms for connected f-factors. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 61–71, 2014.

15 Geevarghese Philip and MS Ramanujan. Vertex exponential algorithms for connected f-factors. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 29. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

16 M. D. Plummer. Graph factors and factorization: 1985–2003: a survey. *Discrete Mathematics*, 307(7):791–821, 2007.

17 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

18 W. T. Tutte. A short proof of the factor theorem for finite graphs. *Canadian Journal of Mathematics*, 6(1954):347–352, 1954. `doi:10.4153/CJM-1954-033-3`.

19 WT Tutte. The factors of graphs. *Canad. J. Math*, 4(3):314–328, 1952.

20 Preben Dahl Vestergaard and Mekkia Kouider. Connected factors in graphs - a survey. *Graphs and Combinatorics*, 21(1):1–26, 2005.

21 Magnus Wahlström. Abusing the tutte matrix: An algebraic instance compression for the k-set-cycle problem. In *STACS*, pages 341–352, 2013.

22 D. B. West. *Introduction to Graph Theory.* Prentice Hall, 2001.

23 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation*, volume 72, pages 216–226. 1979.

On Existential MSO and its Relation to ETH

Robert Ganian[1], **Ronald de Haan**[2], **Iyad Kanj**[3], **and Stefan Szeider**[4]

1 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
ganian@ac.tuwien.ac.at
2 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
dehaan@ac.tuwien.ac.at
3 **School of Computing, DePaul University, Chicago, IL, USA**
ikanj@cs.depaul.edu
4 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
sz@ac.tuwien.ac.at

Abstract

Impagliazzo *et al.* proposed a framework, based on the logic fragment defining the complexity class SNP, to identify problems that are equivalent to k-CNF-Sat modulo subexponential-time reducibility (serf-reducibility). The subexponential-time solvability of any of these problems implies the failure of the Exponential Time Hypothesis (ETH). In this paper, we extend the framework of Impagliazzo *et al.*, and identify a larger set of problems that are equivalent to k-CNF-Sat modulo serf-reducibility. We propose a complexity class, referred to as Linear Monadic NP, that consists of all problems expressible in existential monadic second order logic whose expressions have a linear *measure* in terms of a complexity parameter, which is usually the universe size of the problem.

This research direction can be traced back to Fagin's celebrated theorem stating that NP coincides with the class of problems expressible in existential second order logic. Monadic NP, a well-studied class in the literature, is the restriction of the aforementioned logic fragment to existential *monadic* second order logic. The proposed class Linear Monadic NP is then the restriction of Monadic NP to problems whose expressions have linear measure in the complexity parameter.

We show that Linear Monadic NP includes many natural complete problems such as the satisfiability of linear-size circuits, dominating set, independent dominating set, and perfect code. Therefore, for any of these problems, its subexponential-time solvability is equivalent to the failure of ETH. We prove, using logic games, that the aforementioned problems are inexpressible in the monadic fragment of SNP, and hence, are not captured by the framework of Impagliazzo *et al.* Finally, we show that Feedback Vertex Set is inexpressible in existential monadic second order logic, and hence is not in Linear Monadic NP, and investigate the existence of certain reductions between Feedback Vertex Set (and variants of it) and 3-CNF-Sat.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity, F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases Exponential Time Hypothesis (ETH), Monadic Second Order Logic, Subexponential Time Complexity, Serf-Reducibility, Logic Games

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.42

1 Introduction

Motivation and related work

The area of exact algorithms seeks moderately exponential-time algorithms for NP-hard problems that improve on the trivial brute-force algorithms. This area of research has been gaining a lot of traction in the last two decades. Nowadays, there is a large set of important NP-complete problems such that for each problem there is a long list of exact algorithms, each improving slightly on the running time of the preceding one; we refer the interested reader to [19] for exposure to the area of exact algorithms. Perhaps the most well-known—and important—NP-hard problem is the satisfiability of Boolean formulas in conjunctive normal form, abbreviated CNF-Sat. Its importance stems from its practical relevance as well as from the canonical role it plays for the complexity class NP, being the first problem shown to be complete for this class. Despite all the exercised efforts over the last several decades, the best algorithms remain equivalent (in terms of the upper bound on their running time) to the trivial brute-force algorithm that solves CNF-Sat in time $\mathcal{O}^*(2^n)$,[1] where n is the number of Boolean variables in the input formula. It has become a common belief among a significant number of researchers in the area of exact algorithms that no $\mathcal{O}^*(2^{cn})$-time algorithm, for any constant $c < 1$, exists for CNF-Sat. If we restrict the clause-width to be at most k (where $k \geq 3$) in the instances of CNF-Sat, we obtain the k-CNF-Sat problem, for which there is a long list of moderately exponential-time algorithms culminating in the currently-best deterministic algorithm (for a general value of k) running in time $\mathcal{O}((2 - 2/(k+1))^n)$ [12].

The sequence of improvements in the running time of moderately exponential-time algorithms for k-CNF-Sat, as well as for other NP-complete problems, led researchers to ask whether we can "indefinitely" keep improving the running time of exact algorithms for these problems. More formally, it triggered the question of whether an algorithm solving k-CNF-Sat in time $\mathcal{O}(2^{o(n)})$, or equivalently in time $\mathcal{O}(2^{\varepsilon n})$ for every $\varepsilon > 0$, exists; an algorithm with such running time is referred to as a *subexponential-time algorithm.* In their seminal paper, Impagliazzo *et al.* [22] investigated this question. They proved that the existence of a subexponential-time algorithm for k-CNF-Sat, for any integer-constant $k \geq 3$, is related to its membership in the class SNP, introduced in [30]. The class SNP consists of all search problems expressible by second-order existential formulas whose first-order part is universal; we shall refer to the aforementioned logic fragment as *SNP logic.* For a problem in SNP, they defined a complexity parameter based on its SNP logic expression. They introduced the notion of completeness for the class SNP under serf-reductions, which are subexponential-time complexity preserving reductions with respect to the aforementioned complexity parameter, and identified a class of problems that are complete for SNP under serf-reducibility; the subexponential-time solvability of any of these problems with respect to its corresponding complexity parameter implies the subexponential-time solvability of all problems in SNP. They proved that many well-known NP-hard problems are SNP-complete under serf-reducibility with respect to a complexity parameter that is linear in the natural universe size of the problem, including k-CNF-Sat (universe is the Boolean variables in the formula), Vertex Cover, Independent Set, and c-Colorability ($c \geq 3$) (universe is the vertex set of the graph), for which extensive efforts to develop subexponential-time

Kanj was partially supported by DFG, project NI 369/12. Ganian, de Haan and Szeider were partially supported by FWF, projects P 26200 and P 26696. Ganian is also affiliated with FI MU, Brno, Czech Republic.

[1] The $\mathcal{O}^*$ notation suppresses a polynomial factor in the input length.

algorithms have been made in the last three decades with no success. This led them to formulate the Exponential Time Hypothesis (ETH), stating that k-CNF-Sat is not solvable in subexponential-time, which is equivalent to the statement that not all SNP problems are solvable in subexponential time. ETH has become a standard hypothesis in complexity theory for proving hardness results that is closely related to the computational intractability of a large class of well-known NP-hard problems, measured from a number of different angles, such as subexponential-time complexity, fixed-parameter tractability, and approximation [6, 7, 10, 28, 29].

Our results

The SNP logic framework developed by Impagliazzo *et al.* [22] captures some well-known NP-complete problems that are serf-reducible to k-CNF-Sat with respect to their universe size, but fails to capture many others. Using logic games, we prove in Section 3 that there are many natural NP-complete problems that are not captured by this SNP logic framework, and yet are serf-reducible to k-CNF-Sat. Examples of such problems include the satisfiability problem for linear-size circuits (Linear Circuit-Sat), Dominating Set, Independent Dominating set, and Perfect Code. The restrictedness of the SNP-based framework, due to the restrictedness of SNP logic to allowing only universal quantifiers in the first order part, prevents the expressibility of problems such as the aforementioned ones, as we show in this paper that none of these problems can be expressed in SNP logic with a complexity parameter that is linear in the natural universe size of the problem.

We propose a complexity class, referred to as Linear Monadic NP, that captures more problems than the SNP-based framework of Impagliazzo *et al.* [22]. Fagin's celebrated theorem [15] states that NP coincides with the class of properties expressible in existential second-order logic. The class monadic NP, introduced by Fagin *et al.* [16], is the restriction of NP to problems expressible in monadic second-order logic in which the second-order part is existential, that is, all second-order variables have arity at most one (*i.e.*, set variables), and no universal quantification is allowed over these relations. The class monadic NP is a well-studied complexity class, and several properties have been shown to be inexpressible in the logic fragment defining this class [2, 3, 16, 23, 25]. We show in this paper that the class monadic NP plays an important role as well in identifying problems that are equivalent to k-CNF-Sat under serf-reducibility. We define a measure (for expressions), and use it to define a logic fragment, *Linear Existential Monadic Second-Order Logic* (LEMSO), consisting of the restriction of existential monadic second-order logic to expressions whose measure is linear in the complexity parameter, where the complexity parameter is defined as in Impagliazzo *et al.* [22]. The class Linear Monadic NP consists of all search problems expressible in LEMSO. All natural problems described in [22] including k-CNF-Sat, Independent Set, Vertex Cover, c-Colorability, which are complete for SNP under serf-reducibility with respect to their natural universe size (number of variables/vertices), are also complete for Linear Monadic NP under serf-reducibility. In fact, each problem in the class of problems defined by Impagliazzo *et al.* [22] consisting of all problems expressible in the SNP logic with a linear complexity parameter (as defined in [22]) is in Linear Monadic NP. We prove that problems such as Linear Circuit-Sat, Dominating Set, and Independent Dominating set are in Linear Monadic NP (and are actually complete for Linear Monadic NP), but *are not* expressible in the SNP logic, thus showing that the set of problems expressible in the SNP logic with a linear complexity parameter is a proper subset of Linear Monadic NP. This implies that the subexponential-time solvability of any of the aforementioned problems is equivalent to the failure of ETH. Our inexpressibility proofs use the framework of logic games, namely the Ajtai-Fagin game and the Ehrenfeucht-Fraïssé game.

Finally, we show using logic games that Feedback Vertex Set is inexpressible in EMSO, and hence is not in Linear Monadic NP. Whereas it can be easily shown that 3-CNF-Sat is serf-reducible to Feedback Vertex Set (by composing the two folklore polynomial-time reductions from 3-CNF-Sat to Vertex Cover and from Vertex Cover to Feedback Vertex Set), it is open whether a serf-reduction exists in the other direction. We show that there is a polynomial-time reduction from Feedback Vertex Set to 3-CNF-Sat with a quasi-linear increase in the universe size, and define a variant of Feedback Vertex Set that is equivalent to 3-CNF-Sat under serf-reducibility.

2 Preliminaries

2.1 Satisfiability

The CNF-Satisfiability problem, shortly CNF-Sat, is given a formula F in the CNF form, decide whether or not F is satisfiable. The *width* of a clause in a CNF formula F is the number of literals in the clause. The k-CNF-Sat problem, where $k \geq 2$, is the restriction of the CNF-Sat problem to instances in which the width of each clause is at most k. It is well known that the k-CNF-Sat problem for $k \geq 3$ is NP-complete [20]. Linear CNF-Sat is the restriction of CNF-Sat to formulas with a linear number of clauses with respect to the number of variables.

A *circuit* is a directed acyclic graph. The vertices of indegree 0 are called the *variables* or *input gates*, and are labeled either by *positive literals* x_i or by *negative literals* $\overline{x}_i$. The vertices of indegree larger than 0 are called the *gates* and are labeled with Boolean operators $\wedge$ or $\vee$. A special gate of outdegree 0 is designated as the *output* gate. We do not allow negation gates in the circuit since, by De Morgan's laws, a general circuit can be efficiently converted into the above circuit model. The *size* of a circuit C, denoted $|C|$, is the number of gates in it. A circuit C is *satisfiable* if there is a truth assignment to the input variables of C that makes C evaluate to 1. The Circuit-Satisfiability problem, shortly Circuit-Sat, is given a circuit C, decide whether or not C is satisfiable. Linear Circuit-Sat is the restriction of Circuit-Sat to circuits of linear size with respect to the number of variables.

2.2 Subexponential time and SNP

For any $n \in \mathbb{N}$, by $[n]$ we denote the set $\{1, 2, \ldots, n\}$. The time complexity functions used in this paper are assumed to be proper complexity functions.

It is clear that CNF-Sat is solvable in time $2^n|F|^{\mathcal{O}(1)}$, where F is the input instance and n is the number of variables in F. We say that CNF-Sat is solvable in *subexponential time* if there exists an algorithm that solves the problem in time $2^{o(n)}|F|^{\mathcal{O}(1)}$. Using the results of [8] and [17], the above definition is equivalent to the following: CNF-Sat is solvable in *subexponential time* if there exists an algorithm that for all $\varepsilon = 1/\ell$, where ℓ is a positive integer, solves the problem in time $2^{\varepsilon n}|I|^{\mathcal{O}(1)}$.

Let Q and Q' be two problems, and let μ and μ' be two parameter functions defined on instances of Q and Q', respectively. In the case of CNF-Sat, μ and μ' will be the number of variables in the instances of these problems. A *subexponential-time Turing reduction family* [22], see also [17], shortly a *serf-reduction*[2], is an algorithm A with an oracle to Q' such that there are computable functions $f, g : \mathbb{N} \longrightarrow \mathbb{N}$ satisfying: (1) given a pair (I, ε)

[2] Serf-reductions were introduced in [22]. Here we use the definition given in [17]. There is a slight difference between the two definitions, and the latter definition is more flexible for our purposes.

where $I \in Q$ and $\varepsilon = 1/\ell$ (ℓ is a positive integer), A decides I in time $f(1/\varepsilon)2^{\varepsilon\mu(I)}|I|^{\mathcal{O}(1)}$; and (2) for all oracle queries of the form "$I' \in Q'$" posed by A on input (I, ε), we have $\mu'(I') \leq g(1/\varepsilon)(\mu(I) + \log |I|)$.

The class SNP consists of all search problems expressible by second-order existential formulas whose first-order part is universal [30]; that is, search problems expressible by second-order formulas of the form $\exists R_1 \ldots \exists R_q \, \forall z_1 \ldots \forall z_r \; \Phi(S, R_1, \ldots, R_q, z_1, \ldots, z_r)$, where S is the input structure, $R_1, \ldots, R_q$ are (bound) relations, $z_1, \ldots, z_r$ are first-order variables, and Φ is a quantifier-free Boolean formula. We will refer to the preceding logic fragment as the *SNP logic*. Impagliazzo *et al.* [22] defined a complexity parameter for each such expression equal to $\sum_{i=1}^{q} |R_i|^{\alpha_i}$, where $|R_i|$ and α_i, $i = 1, \ldots, q$, are the encoding length of R_i and its arity, respectively. For monadic relations, the number of bits needed to describe R_i, is the size of the universe on which R_i is interpreted, and hence, as noted in [22], the complexity parameter in this case is linear in the universe size. They formulated the *Exponential Time Hypothesis* (ETH) stating that k-CNF-Sat (for any $k \geq 3$) cannot be solved in subexponential time $2^{o(n)}$, where n is the number of variables in the input formula. Therefore, there exists $c > 0$ such that k-CNF-Sat cannot be solved in time $\mathcal{O}(2^{cn})$. ETH is equivalent to the statement that not all SNP problems are solvable in subexponential time.

3 Linear Monadic NP and Serf-Reducibility to Linear Circuit-Sat

In this section, we present a fragment of existential second-order logic that extends the SNP logic introduced by Impagliazzo *et al.* [22] in the sense that it captures a larger set of problems that are serf-reducible to 3-CNF-Sat. The logic fragment we propose is a restriction of the well-studied existential monadic second-order logic (EMSO) that defines the complexity class Monadic NP (modulo standard syntactic augmentations to allow the expression of optimization problems). To define this logic fragment, we introduce a measure/function that associates with each expression in EMSO a value in terms of the complexity parameter of the expression, and we define Linear Monadic NP to be the restriction of Monadic NP to those expressions whose measure is linear in the complexity parameter. The complexity parameter of the expression we use is the same parameter defined by [22] for expressions in SNP: If $R_1, \ldots, R_q$ are the existentially-quantified second-order relations in the expression, where R_i has arity α_i, for $i = 1, \ldots, q$, then the complexity parameter of the expression is $\sum_{i=1}^{q} |R_i|^{\alpha_i}$.

We note that the SNP logic does not restrict the existentially quantified relations to be monadic, and we could have opted to do the same in this paper (*i.e.*, not restrict ourselves to monadic relations and Monadic NP), and the results in this paper would not have been affected. However, observe that the aforementioned complexity parameter defined by [22] and also used by us, when interpreted on a given structure (graph, CNF-formula, etc.) yields a parameter that is equal to $\sum_{i=1}^{q} |U|^{\alpha_i}$, where U is the natural universe on which the relations $R_1, \ldots, R_q$ are interpreted. Hence, this parameter can only be linear in the universe size (number of vertices in a graph, number of variables in a formula, etc.) if all relations in the expression are monadic; otherwise, the complexity parameter will be at least quadratic. To be able to use serf-reductions to claim that a subexponential-time algorithm for 3-CNF-Sat yields a subexponential-time algorithm for the search problem under consideration with respect to a complexity parameter that is linear in the universe size of the problem, the existentially-quantified relations in the SNP logic expression of the problem need to be monadic. We will refer to this fragment of SNP logic as *monadic SNP logic*, which, by the above, is the fragment of interest in the framework of Impagliazzo *et al.* [22] when studying the equivalence between natural problems modulo serf-reducibility.

We refer to the corresponding complexity class as Monadic SNP. By the same token, and without loss of generality, in this paper we restrict ourselves to Monadic NP.

3.1 Linear Monadic NP

The fragment of existential second-order logic that we consider is based on an inductively defined measure. Intuitively, this measure captures the size of the Boolean circuit that can be constructed for a concrete instance of the problem that is satisfiable if and only if there is an assignment to the second-order variables that makes the first-order part of the formula true.

We consider EMSO formulas with a single free monadic second-order variable, to be able to express optimization problems where solutions of a given (exact) size are sought.

In particular, let $\varphi(S) = \exists R_1 \dots \exists R_q \; Q_1 z_1 \dots Q_r z_r \; \Phi(S, R_1, \dots, R_q, z_1, \dots, z_r)$ be an existential second-order logic formula, where S is a free monadic second-order variable, where $R_1, \dots, R_q$ are (bound) monadic second-order variables, where $z_1, \dots, z_r$ are first-order variables and $Q_1, \dots, Q_r \in \{\exists, \forall\}$, and where Φ is quantifier-free.

Moreover, let A be an optimization problem consisting of inputs (I, N), where I is a relational structure over the same relational vocabulary as Φ, and where N is a positive integer.

We say that $\varphi(S)$ expresses the problem A if for each input (I, N) it holds that $(I, N) \in A$ if and only if there is an interpretation S_0 of S in I of size exactly N such that $I \models \varphi(S_0)$.

More generally, we also consider existential second-order logic formulas $\varphi(S)$ where the first-order part does not need to be in prenex form, that is, first-order quantifications and Boolean connectives can freely alternate.

Without loss of generality, we suppose that formulas $\varphi(S)$ are in negation normal form.

For search problems that do not involve an optimization component (*e.g.*, 3-CNF-Sat), we can simply omit the free variable S in the logic expression and the integer N in the problem input. All definitions and results extend straightforwardly to this setting.

For optimization problems where solutions of a given minimum or maximum size are sought, it suffices to investigate the variant of the problem involving solutions of exactly the given size, because it can easily be shown that there is a serf-reduction between these different problem variants.

Our size measure s is based on inductively defined measures, $s_\forall$ and $s_\exists$, that we define below. All measures take as input a second-order logic formula, and return an arithmetic expression over the single variable n. For any formula $\varphi(S)$ that expresses a search problem A, intuitively, the expression $s(\varphi(S)) = f(n)$ denotes an upper bound on the size of the Boolean circuit that expresses a given instance in terms of the input size n.

We inductively define these measures as follows. Here we let Q range over $\{\forall, \exists\}$, and we let $\overline{Q}$ denote the unique quantifier in $\{\forall, \exists\} \backslash \{Q\}$. Moreover, we let $\circ$ range over $\{\wedge, \vee\}$, and we let a denote any atom.

$$s(\exists R_i\ \psi) = s(\psi); \qquad (1) \qquad s(\psi_1 \circ \psi_2) = 1 + s(\psi_1) + s(\psi_2); \qquad (2)$$

$$s(Qz_i\ \psi) = \begin{cases} n & \text{if } \psi \text{ does not contain any occurrence of } \overline{Q}; \\ 1 + s_Q(Qz_i\ \psi) & \text{otherwise;} \end{cases} \qquad (3)$$

$$s_Q(\overline{Q}z_i\ \psi) = 1 + n \cdot s_{\overline{Q}}(\psi); \qquad (4) \qquad s_Q(a) = s_Q(\neg a) = 0; \qquad (6)$$

$$s_Q(Qz_i\ \psi) = n \cdot s_Q(\psi); \qquad (5)$$

$$s_Q(\psi_1 \circ \psi_2) = s_Q(\psi_2) \quad \text{if } \psi_1 \text{ contains no occurrences of second-order variables;} \qquad (7)$$

$$s_Q(\psi_1 \circ \psi_2) = s_Q(\psi_1) \quad \text{if } \psi_2 \text{ contains no occurrences of second-order variables;} \qquad (8)$$

for the case where both ψ_1 and ψ_2 contain second-order variables, we let:

$$s_\exists(\psi_1 \vee \psi_2) = s_\exists(\psi_1) + s_\exists(\psi_2); \qquad (9) \qquad s_\forall(\psi_1 \vee \psi_2) = 1 + s_\exists(\psi_1) + s_\exists(\psi_2); \qquad (11)$$

$$s_\forall(\psi_1 \wedge \psi_2) = s_\forall(\psi_1) + s_\forall(\psi_2); \qquad (10) \qquad s_\exists(\psi_1 \wedge \psi_2) = 1 + s_\forall(\psi_1) + s_\forall(\psi_2). \qquad (12)$$

The intuition behind the definition of the measure is that $s(\varphi(S))$ measures the size of the Boolean circuit C that is needed to express the formula $\varphi(S)$ when interpreted over the input I; $s(\varphi(S))$ returns an arithmetic expression with a single symbol, n, that is interpreted as the size of the universe U of the relational structure I. The variables of C are of the form $x_{T,e}$, representing whether an element e in the domain of I is chosen to be part of the interpretation of the monadic second-order variable T. The circuit C then encodes the first-order part of φ.

The computation of $s(\varphi)$ works as follows. The existential second-order quantifiers are disregarded, and the outermost Boolean connectives are dealt with in a straightforward inductive manner.

For every maximal subformula ψ that starts with a first-order quantifier Q, there are two options. Either ψ contains only first-order quantifiers of a single type—in which case the measure s returns n for this subformula—or ψ contains quantifiers of both types. In the latter case, the size of the (sub)circuit to represent ψ is measured inductively using s_Q.

The measure s_Q keeps track of what logic gate is the parent of the subcircuit representing the subformula ψ (*i.e.*, $Q = \forall$ corresponds to an $\wedge$-gate as parent and $Q = \exists$ corresponds to an $\vee$-gate as parent), and increases the size only if the subcircuit cannot be integrated/merged with its parent gate. For example, if the output gate of the subcircuit is an $\wedge$-gate, and its parent is also an $\wedge$-gate, these two gates can be merged into a single large $\wedge$-gate.

▶ **Definition 1.** We define Linear Monadic NP to be the class of NP search problems such that each is expressible using an EMSO formula $\varphi(S)$ with $s(\varphi(S)) = \mathcal{O}(n)$.

The following theorem shows that Linear Circuit-Sat can serve as the canonical satisfiability problem for Linear Monadic NP:

▶ **Theorem 2.** *Let A be a problem in Linear Monadic NP, and suppose that A is expressible using an EMSO formula $\varphi(S)$, where $s(\varphi(S)) = \mathcal{O}(n)$. Then A is serf-reducible to* Linear Circuit-Sat*, where the parameter of A is the size of the universe of the input structure, and that for* Linear Circuit-Sat *is the number of variables of the circuit.*

Sketch of Proof. Let A be a search problem with input of the form (I, N), where I is a relational structure with universe U, and $N \in \mathbb{N}$, such that A is expressed by the EMSO formula $\varphi(S) = \exists R_1, \dots, R_q\ \Phi(R_1, \dots, R_q, S)$, with $s(\varphi(S)) = \mathcal{O}(n)$.

We know that Φ consists of first-order formulas $\psi_1, \dots, \psi_k$—each starting with a first-order quantifier—that are combined using the connectives $\wedge$ and $\vee$, and that $s(\psi_i) = \mathcal{O}(n)$ for each $i \in [k]$.

We reduce the problem to Linear Circuit-Sat by transforming each such formula ψ_i into a sequence of (subexponentially many) circuits $C_{i,1}, \dots, C_{i,b_i}$ where each such circuit C_{i,b_i} is of size linear in $|U|$.

If ψ_i contains only universal first-order quantifiers, or only existential first-order quantifiers, constructing such a sequence of circuits can be done using known results [22, Theorems 1 and 2].

Otherwise, if ψ_i contains both universal and existential first-order quantifiers, we transform ψ_i into a single circuit C_i inductively, breaking the formula ψ_i into its subformulas and constructing subcircuits for each subformula.

Intuitively, this circuit C_i is constructed by 'unfolding' universal quantifiers into $\wedge$-gates and existential quantifiers into $\vee$-gates. Using the inductive definition of the measure s in Equations (1)–(12), and by our assumption that $s(\psi_i)$ is a linear function in n, we can derive that this construction yields a circuit C_i that is of size linear in $|U|$. ◀

▶ **Definition 3.** A problem A is *Linear Monadic NP-complete under serf-reducibility* if (1) it is in Linear Monadic NP and (2) every problem in Linear Monadic NP is serf-reducible to A.

▶ **Corollary 4.** Linear Circuit-Sat *is Linear Monadic NP-complete under serf-reducibility.*

Note that all problems in Monadic SNP [22] are in Linear Monadic NP. This is because every EMSO expression (in negation normal form) that contains no first-order quantifier alternations has a linear measure—due to Equation (3). Because Linear Circuit-Sat is serf-reducible to 3-CNF-Sat [24], Corollary 4 implies the following:

▶ **Corollary 5.** *Every Monadic SNP-complete problem is Linear Monadic NP-complete under serf-reducibility. Therefore, a Linear Monadic NP-complete problem is solvable in subexponential time if and only if ETH fails.*

The above implies that the well-known SNP-complete problems k-CNF-Sat ($k \geq 3$), c-Colorability, Independent Set, Clique, and Vertex Cover, among others, are Linear Monadic NP-complete. (Note that every problem that is complete for Linear Monadic NP under serf-reducibility is clearly hard for SNP under serf-reducibility.)

3.2 Applications: Expressing Natural Optimization Problems

We saw in the previous subsection that Linear Circuit-Sat is in Linear Monadic NP. We prove in Subsection 3.3 that Linear Circuit-Sat is not in monadic SNP. The same can be shown for problems that are serf-equivalent to Linear Circuit-Sat, such as Linear Hitting Set and Linear Set Cover. In this subsection, we give several examples of natural graph problems that are in Linear Monadic NP, but are inexpressible in monadic SNP logic as will be shown in Subsection 3.3. The definition of these natural problems can be found in [14].

Our first example is Dominating Set, with the number of vertices as the complexity parameter. This problem can be expressed using the following formula $\varphi_{\mathrm{DS}}(S)$:

$$\varphi_{\mathrm{DS}}(S) = \forall x \ (S(x) \vee \exists y(S(y) \wedge E(x,y))).$$

From Equations (1)–(12), we get that $s(\varphi_{\mathrm{DS}}(S)) = n + 1$. In effect:

$$\begin{aligned}
s(\varphi_{\mathrm{DS}}(S)) &= s[\forall x \ (S(x) \vee \exists y(S(y) \wedge E(x,y)))] && = 1 + s_\forall[\forall x \ (S(x) \vee \exists y(S(y) \wedge E(x,y)))] \\
&= 1 + n \cdot s_\forall[(S(x) \vee \exists y(S(y) \wedge E(x,y)))] && = 1 + n \cdot (1 + s_\exists[S(x)] + s_\exists[\exists y(S(y) \wedge E(x,y))]) \\
&= 1 + n \cdot (1 + 0 + n \cdot s_\exists[S(y) \wedge E(x,y)]) && = 1 + n \cdot (1 + 0 + n \cdot s_\exists[S(y)]) \\
&= 1 + n \cdot (1 + 0 + n \cdot 0) && = n + 1.
\end{aligned}$$

It follows that Dominating Set is in Linear Monadic NP. Our second example is the Red-Blue Dominating Set problem [11], with the number of vertices as the complexity parameter. The expression is $\varphi_{\text{RB-DS}}(S) = \forall x\,(\neg S(x) \vee R(x)) \wedge \forall y(\neg B(y) \vee \exists z(S(z) \wedge E(z,y)))$, for which it can be easily verified that $s(\varphi_{\text{RB-DS}}(S)) = 2n+2$.

Similarly, we can express the problem Non-Blocker [14], with the number of vertices as complexity parameter, using the following formula $\varphi_{\text{NB}}(S) = \forall x\,(\neg S(x) \vee \exists y\,(\neg S(y) \wedge E(x,y)))$. It can be easily verified that $s(\varphi_{\text{NB}}(S)) = n+1$, and hence Non-Blocker is in Linear Monadic NP.

The Independent Dominating Set problem [11] is also in Linear Monadic NP because it can be expressed as a conjunction of the two expressions for Dominating Set and Independent Set. Specifically, the formula $\varphi_{\text{IDS}}(S) = \varphi_{\text{DS}}(S) \wedge [\forall x \forall y\,(\neg E(x,y) \vee \neg S(x) \vee \neg S(y))]$ expresses the problem and $s(\varphi_{\text{IDS}}(S)) = 2n+2$.

A similar expression to the above shows that the Dominating Clique problem [9] is also in Linear Monadic NP.

Further examples of problems that are in Linear Monadic NP are Distance-r Dominating Set [21], r-Threshold Dominating Set [14], and r-Domatic Partition [32], for any integer-constant r.

We give the expression for r-Dominating Set below, whose measure can verified to be $\mathcal{O}(n)$. The expressions for the other two problems are omitted.

$$\varphi_{r\text{-DS}}(S) = \forall x\ \exists y_1 \exists y_2 \ldots \exists y_r\ [((x = y_1) \vee E(x,y_1)) \wedge \ldots \wedge ((y_{r-1} = y_r) \vee E(y_{r-1},y_r)) \wedge S(y_r)].$$

Finally, the Perfect Code problem [11] is in Linear Monadic NP because it can be expressed using the following formula $\varphi_{\text{PC}}(S)$, for which it can be verified that $s(\varphi_{\text{PC}}(S)) = 3n+3$:

$$\begin{aligned}\varphi_{\text{PC}}(S) = \quad & [\forall x \forall y\,(\neg E(x,y) \vee \neg S(x) \vee \neg S(y))] \wedge [\forall z\,(S(z) \vee \exists y\,(S(y) \wedge E(z,y)))] \wedge \\ & [\forall z \forall x \forall y\,(\neg S(x) \vee \neg S(y) \vee \neg E(x,z) \vee \neg E(y,z))].\end{aligned}$$

Via standard reductions from 3-CNF-Sat, it can be easily shown that all the problems discussed above are complete for Linear Monadic NP under serf-reducibility. Therefore, we have the following:

▶ **Corollary 6.** *For any of the problems* Dominating Set, Red-Blue Dominating Set, Independent Dominating Set, Dominating Clique, r-Threshold Dominating Set, Distance-r Dominating Set, r-Domatic Partition, *and* Perfect Code, *the following holds: the problem is solvable in subexponential time if and only if ETH fails.*

3.3 Inexpressibility in Monadic SNP Logic

We prove in this subsection that Dominating Set is inexpressible in monadic SNP logic, and show how the proof can be straightforwardly adapted to other considered problems. In particular, we show that there exists no formula φ in this logic with one free set variable such that $(G,S) \models \varphi(S)$ if and only if S is a dominating set in G. This shows that the greater freedom in quantifier alternation offered by our new logic fragment is necessary to express this problem. Before we give a formal proof of this result, we will need several notions from the area of logic; we refer the reader to Libkin's book [26]. The key tool we use to prove inexpressibility is the so-called Ajtai-Fagin game [1, 26], defined below.

▶ **Definition 7.** Let $\mathcal{P}$ be a property of graphs equipped with a single set, let Γ be a fragment of FO logic, and let $\ell, k \in \mathbb{N}$. Then the $(\mathcal{P}, \ell, k, \Gamma)$-Ajtai-Fagin game is a 2-player game between the duplicator and the spoiler which proceeds in the following 4 steps:

1. The duplicator selects a graph G equipped with a set S such that $(G, S) \in \mathcal{P}$.
2. The spoiler selects ℓ subsets $U_1, \ldots, U_\ell$ of $V(G)$.
3. The duplicator selects a graph G' equipped with a set S' and also ℓ subsets $U'_1, \ldots, U'_\ell$ of $V(G')$.
4. The duplicator wins if and only if he can prove that $(G, S, U_1, \ldots, U_\ell)$ and $(G', S', U'_1, \ldots, U'_\ell)$ agree on $\Gamma[k]$.

▶ **Proposition 8** ([1, 26])**.** The duplicator has a winning strategy in the $(\mathcal{P}, \ell, k, \Gamma)$-Ajtai-Fagin game if and only if $\mathcal{P}$ is not definable by any formula with a single free set variable S of the form $\exists X_1, \ldots, \exists X_\ell \chi(S, X_1, \ldots, X_\ell)$, where χ is a formula in Γ.

The final ingredient we need is a result which links the number of quantifier alternations in an FO formula to the moves of the Ehrenfeucht-Fraïssé game. The *j-alternations k-round Ehrenfeucht-Fraïssé game* is a restriction of the standard Ehrenfeucht-Fraïssé game to at most j alternations (“switches”) of the structure where the spoiler makes his moves. As a special case of a result by Pezzoli [31], it follows that the duplicator wins the 0-alternations k-round Ehrenfeucht-Fraïssé game on $(\mathcal{A}, \mathcal{B})$ if and only if $\mathcal{A}$ and $\mathcal{B}$ agree on all universal first-order formulas of quantifier rank at most k. We are now ready to prove our inexpressibility result.

▶ **Lemma 9.** Dominating Set *is inexpressible in monadic SNP logic.*

Sketch of Proof. We use the Ajtai-Fagin game and describe a winning strategy for the duplicator for every fixed $\ell, k \in \mathbb{N}$. In Step 1, the duplicator selects a graph G which consists of $2^{2\ell} \cdot k + 1$ copies of K_2, and a set S which contains a single vertex from each copy of K_2; observe that S is a dominating set of G. In Step 2, the spoiler arbitrarily selects his subsets $U_1, \ldots, U_\ell$ of vertices of G.

Before proceeding to Step 3, we need to find a “sufficiently frequent” configuration in $(G, U_1, \ldots, U_\ell)$ for the duplicator to exploit. Let the *type* $T(v)$ of a vertex $v \in G$ be defined as $T(v) = \{i \mid v \in U_i\}$. Let a, b be vertices of G which form a K_2 such that $b \in S$. Then the *configuration* $C(a, b)$ of a, b is the tuple $(T(a), T(b))$. Since the number of distinct types is upper-bounded by 2^ℓ, the number of distinct configurations is upper-bounded by $2^{2\ell}$. By construction, there must exist some configuration which occurs at least $k + 1$ times in G; let us now fix an arbitrary such configuration (T_1, T_2).

In Step 3, the duplicator selects a graph H equipped with a set Q. The graph H consists of a copy of G and one isolated vertex p. Let f be a bijection witnessing the isomorphism from $H - p$ to G; then the duplicator selects H and also the vertex-subsets $W_1, \ldots, W_\ell$ of H as follows: (1) for each vertex $w \in H - p$, $w \in Q$ if and only if $f(w) \in S$; (2) for each vertex $w \in H - p$ and each $i \in [\ell]$, $w \in W_i$ if and only if $f(w) \in U_i$; and (3) $p \in W_i$ if and only if $i \in T_1$.

In Step 4, it suffices to prove that $(G, S, U_1, \ldots, U_\ell)$ and $(H, Q, W_1, \ldots, W_\ell)$ agree on all first-order formulas with alternation number 0 and quantifier rank at most k. This is done by giving a winning strategy for the duplicator in the 0-alternations k-round Ehrenfeucht-Fraïssé game. On a high level, if the spoiler chooses to play on G, then the duplicator can precisely copy the spoiler's moved on H. On the other hand, if the spoiler decides to play on H then the duplicator can also precisely copy all spoiler's moves on G with the exception of a move on p; there, the duplicator uses the fact that the configuration (T_1, T_2) is sufficiently frequent to find a suitable replacement for p in G. ◀

The same technique can be applied to prove the inexpressibility of other problems considered in Subsection 3.2. For some, it suffices to merely adapt the above construction, while for others, such as Linear Circuit-Sat, new constructions are required.

4 Feedback Vertex Set

In the previous section, we gave a logic fragment such that every problem that is expressible in this logic fragment is serf-reducible to 3-CNF-Sat. A natural question to ask is whether there exist NP-complete problems that are not serf-reducible to 3-CNF-Sat, under some plausible complexity-theoretic hypothesis. This question has been answered by several works. For instance, Calabro *et al.* [4] showed that, unless ETH fails, the restriction of CNF-Sat to instances in which the number of clauses is super-linear in the number of variables is not serf-reducible to 3-CNF-Sat. More unlikely complexity-theoretic consequences befall if we replace CNF-Sat restricted to instances with super-linear number of clauses with "harder" satisfiability problems (*e.g.*, general CNF-Sat or Circuit-Sat). While the aforementioned problems are all expressible in EMSO, their expressions yield super-linear measures.

We also saw in the previous section that many natural graph problems, such as c-Colorability (for any integer-constant $c > 0$), Independent Set, Vertex Cover and Dominating Set are serf-reducible to 3-CNF-Sat, which raises the question of whether there is any natural graph problem that is not serf-reducible to 3-CNF-Sat. While it is not difficult to define a graph problem that, unless ETH fails, is not serf-reducible to 3-CNF-Sat, we do not know of any natural graph problem for which the aforementioned statement can be proved. In this section, we propose Feedback Vertex Set as a possible such candidate problem. The reason we believe that Feedback Vertex Set might be such a problem is that Feedback Vertex Set implicitly embodies a Hitting Set problem [5] (or a satisfiability problem) with possibly exponentially-many cycles to hit. Whereas in CNF-Sat and in Hitting Set the sets/clauses are given explicitly (*i.e.*, are part of the input), which allows us to quantify over the set of all sets/clauses, and hence, express these problems in monadic NP, the cycles in an instance of Feedback Vertex Set are implicitly encoded in the input graph. Moreover, enumerating all the cycles in a graph may require exponential time, which surpasses the allowed time in a serf-reduction.

While we are unable to prove that Feedback Vertex Set is not serf-reducible to 3-CNF-Sat (assuming ETH does not fail), we could prove that Feedback Vertex Set is inexpressible in EMSO, which rules out the possibility of using the proposed framework in this paper to show that Feedback Vertex Set is serf-reducible to 3-CNF-Sat. We leave it as an open problem whether Feedback Vertex Set is serf reducible to 3-CNF-Sat. We also show in this section that there is a polynomial-time reduction from Feedback Vertex Set to 3-CNF-Sat that maps an instance of Feedback Vertex Set with n vertices to an equivalent instance of 3-CNF-Sat with $\mathcal{O}(n \lg n)$ variables, and define a variant of Feedback Vertex Set that is equivalent to 3-CNF-Sat under serf-reducibility.

4.1 Inexpressibility of Feedback Vertex Set in EMSO

Recall that, by Theorem 2, we could obtain a serf-reduction from Feedback Vertex Set to 3-CNF-Sat if there existed an EMSO formula $\varphi(X)$ with linear measure such that $(G, S) \models \varphi(S)$ if and only if S is a feedback vertex set in G. In this subsection, we will prove that this approach cannot work; in particular, we show that feedback vertex set is not even expressible in EMSO. Our first step lies in showing that acyclicity is inexpressible by an EMSO sentence. However, instead of using the Ajtai-Fagin game (which would require a highly technical specification of a strategy for the duplicator), our proof will rely on the classical notion of *Hanf-locality.* We refer to Libkin's book [26] for the omitted definitions.

▶ **Lemma 10.** *Acyclicity is inexpressible in EMSO.*

Sketch of Proof. Suppose for a contradiction that acyclicity is definable by an EMSO sentence $\Psi = \exists Z_1 \dots Z_\ell\ \psi$. Since ψ is a first-order sentence, it is Hanf-local. Let G be a path of length at least r. Since G is acyclic, we have $G \models \Psi$. Let $U_1, \dots, U_m$ witness this fact, that is, $(G, U_1, \dots, U_m) \models \psi$. Let a, b be vertices of distance at least $2d+2$ from each other such that $N_d^{(G,U_1,\dots,U_m)}(a)$ is isomorphic to $N_d^{(G,U_1,\dots,U_m)}(b)$. Let p be the unique endpoint of the path that is closer to b than to a, and let a' be the unique neighbor of a which is closer to p (than the other neighbor of a), and let b' be that of b. We construct a new graph G' by removing edges aa' and bb' from G and adding edges ab' and ba' into G. We have that, for every vertex c, $N_d^{(G,U_1,\dots,U_m)}(c) \approx N_d^{(G',U_1,\dots,U_m)}(c)$. Observe that G' is not acyclic. To complete the proof, we use the Hanf-locality to prove that G' also satisfies Ψ. ◀

▶ **Lemma 11.** Feedback Vertex Set *is inexpressible in EMSO logic.*

4.2 Reductions between 3-CNF-Sat and Feedback Vertex Set

We now examine the existence of polynomial-time serf-reductions between 3-CNF-Sat and Feedback Vertex Set. Based on the framework in [13], we can show that it is unlikely that 3-CNF-Sat is polynomial-time serf-reducible to Feedback Vertex Set:

▶ **Proposition 12.** Unless the polynomial-time hierarchy collapses to its third level, 3-CNF-Sat is not polynomial-time serf-reducible to Feedback Vertex Set.

While we are unable to rule out—under plausible conditions—the existence of a polynomial-time serf-reduction from Feedback Vertex Set to 3-CNF-Sat, we give a polynomial-time reduction from Feedback Vertex Set to 3-CNF-Sat at the cost of a logarithmic factor increase in the number of variables in the 3-CNF-Sat instances. This reduction relies on the following characterization of Feedback Vertex Set that we prove: A graph G has a feedback vertex set of cardinality k if and only if there is a bijection $\varphi : V(G) \longrightarrow [n]$ such that, for all $uv, uw \in E(G)$, we have

$$(\varphi(u) > k) \wedge (\varphi(v) > k) \wedge (\varphi(w) > k) \implies (\varphi(v) > \varphi(u)) \vee (\varphi(w) > \varphi(u)).$$

Using the above characterization, we can reduce Feedback Vertex Set to 3-CNF-Sat by encoding each vertex using a block of $\lg n$ Boolean variables, and adding polynomially-many 3-CNF clauses, each of logarithmic width, encoding the existence of a function φ satisfying the above property. We have:

▶ **Theorem 13.** *There is a polynomial-time many-one reduction that takes an instance* (G, k) *of* Feedback Vertex Set *and produces an equivalent instance* F *of* CNF-Sat *such that* F *has* $\mathcal{O}(n \lg n)$ *variables,* $n^{\mathcal{O}(1)}$ *clauses, and width* $\mathcal{O}(\lg n)$*, where* $n = n(G)$*.*

We now define a variant of Feedback Vertex Set, denoted Monochromatic 3-Feedback Vertex Set, that we show to be equivalent under serf-reducibility to 3-CNF-Sat. An instance of Monochromatic 3-Feedback Vertex Set consists of a graph G and a nonnegative integer k. Each edge $e \in E(G)$ is associated with a set of colors, denoted Colors(e). The question is to decide if there exists a subset $Q \subseteq V(G)$ of vertices of cardinality at most k such that, for every 3-cycle C in G satisfying $\bigcap_{e \in E(C)} \text{Colors}(e) \neq \emptyset$, C contains at least one vertex of Q. That is, Q breaks every 3-cycle in G whose edges can all be assigned the same color from their color lists. It can be easily shown that Monochromatic 3-Feedback Vertex Set is NP-complete by adapting the standard reduction from Vertex Cover to Feedback Vertex Set (the color lists of all edges are singletons and consist of the same color). By d-Monochromatic 3-Feedback Vertex Set, where $d \geq 1$, we denote the restriction of MONOCHROMATIC 3-FEEDBACK VERTEX SET to instances in which the total number of colors assigned is at most d (that is, $|\bigcup_{e \in E(G)} \text{Colors}(e)| \leq d$). We have:

▶ **Theorem 14.** *For any integer $d \geq 10$, 3-CNF-SAT and d-MONOCHROMATIC 3-FEEDBACK VERTEX SET are equivalent under serf-reductions. Therefore, d-MONOCHROMATIC 3-FEEDBACK VERTEX SET is solvable in subexponential time if and only if ETH fails.*

▶ Remark. A natural question to ask is whether serf-reducibility to 3-CNF-SAT is equivalent to the notion of self-sparsification (as in the sparsification lemma for 3-CNF-SAT): The existence of a subexponential-time self-reduction that reduces an instance of the problem to an equivalent instance whose (instance) size is linear in the designated parameter. These two notions, in general, seem to be orthogonal. On one hand, it can be shown that FEEDBACK VERTEX SET is self-sparsifiable [18, 27], but this does not seem to imply that FEEDBACK VERTEX SET is serf-reducible to 3-CNF-SAT. On the other hand, CLIQUE is serf-reducible to 3-CNF-SAT, but unless ETH fails, CLIQUE is not self-sparsifiable since this would imply that CLIQUE is solvable in subexponential time.

References

1 Miklós Ajtai and Ronald Fagin. Reachability is harder for directed than for undirected finite graphs. *J. Symb. Log.*, 55(1):113–150, 1990.

2 Miklós Ajtai, Ronald Fagin, and Larry J. Stockmeyer. The closure of monadic NP. *J. Comput. Syst. Sci.*, 60(3):660–716, 2000.

3 Sanjeev Arora and Ronald Fagin. On winning strategies in Ehrenfeucht-Fraïssé games. *Theor. Comput. Sci.*, 174(1-2):97–121, 1997.

4 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 252–260. IEEE Computer Society, 2006.

5 Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno-Centeno, and Santosh Vempala. Algorithms for implicit hitting set problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 614–629. SIAM, 2011.

6 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005.

7 Jianer Chen, Xiuzhen Huang, Iyad Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. of Computer and System Sciences*, 72(8):1346–1367, 2006.

8 Jianer Chen, Iyad Kanj, and Ge Xia. On parameterized exponential time complexity. *Theoretical Computer Science*, 410(27-29):2641–2648, 2009.

9 Margaret B. Cozzens and Laura L. Kelleher. Dominating cliques in graphs. *Discrete Mathematics*, 86(1-3):101–116, 1990.

10 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight bounds for graph homomorphism and subgraph isomorphism. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1643–1649, 2016.

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

12 Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic

$(2-2/(\mathrm{k}+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002.

13 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014.

14 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer Verlag, New York, 1999.

15 Ronald Fagin. Generalized first-order spectra, and polynomial. time recognizable sets. *SIAM-AMS Proceedings*, 7:43–73, 1974.

16 Ronald Fagin, Larry J. Stockmeyer, and Moshe Y. Vardi. On monadic NP vs. monadic co-NP. *Information and Computation*, 120(1):78–92, 1995.

17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series.* Springer Verlag, Berlin, 2006.

18 Fedor V. Fomin, Serge Gaspers, Artem V. Pyatkin, and Igor Razgon. On the minimum feedback vertex set problem: Exact and enumeration algorithms. *Algorithmica*, 52(2):293–307, 2008.

19 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms.* Springer Verlag, 2010.

20 Michael R. Garey and David R. Johnson. *Computers and Intractability.* W. H. Freeman and Company, New York, San Francisco, 1979.

21 Adriana Hansberg, Dirk Meierling, and Lutz Volkmann. Distance domination and distance irredundance in graphs. *Electr. J. Comb.*, 14(1), 2007.

22 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. of Computer and System Sciences*, 63(4):512–530, 2001.

23 David Janin and Jerzy Marcinkowski. A toolkit for first order extensions of monadic games. In *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer*, volume 2010 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2001.

24 Iyad Kanj and Stefan Szeider. Parameterized and subexponential-time complexity of satisfiability problems and applications. *Theoretical Computer Science*, 607:282–295, 2015.

25 Martin Kreidler and Detlef Seese. Monadic NP and graph minors. In *Proceedings of the 12th International Workshop on Computer Science Logic*, volume 1584 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 1998.

26 Leonid Libkin. *Elements of Finite Model Theory.* Springer Verlag, 2004.

27 Daniel Lokshtanov. Personal communication, 2015.

28 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the European Association for Theoretical Computer Science*, 105:41–72, 2011.

29 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6:85–112, 2010.

30 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. of Computer and System Sciences*, 43(3):425–440, 1991.

31 Elena Pezzoli. Computational complexity of ehrenfeucht-fraïssé games on finite structures. In *Computer Science Logic, 12th International Workshop, CSL '98, Annual Conference of the EACSL, Brno, Czech Republic, August 24-28, 1998, Proceedings*, pages 159–170, 1998.

32 Sheung-Hung Poon, William Chung-Kung Yen, and Chin-Ting Ung. Domatic partition on several classes of graphs. In *Combinatorial Optimization and Applications - 6th International Conference, COCOA 2012, Banff, AB, Canada, August 5-9, 2012. Proceedings*, pages 245–256, 2012.

Programming Biomolecules That Fold Greedily During Transcription[*]

Cody Geary[1], Pierre-Étienne Meunier[2], Nicolas Schabanel[3], and Shinnosuke Seki[4]

1 **California Institute of Technology, Pasadena, CA, USA.** `codyge@gmail.com`.
2 **Department of Computer Science, Aalto University, Finland and Aix Marseille Université, CNRS, LIF UMR 7279, 13288, Marseille, France.** `http://users.ics.aalto.fi/meunier/`
3 **CNRS, Université Paris Diderot, France and IXXI, Université de Lyon, France.** `http://www.irif.univ-paris-diderot.fr/users/nschaban/`
4 **University of Electro-Communications, Tokyo, Japan.** `s.seki@uec.ac.jp`

Abstract

We introduce and study the computational power of Oritatami, a theoretical model to explore greedy molecular folding, by which a molecule begins to fold before awaiting the end of its production. This model is inspired by a recent experimental work demonstrating the construction of shapes at the nanoscale by folding an RNA molecule during its transcription from an engineered sequence of synthetic DNA.

An important challenge of this model, also encountered in experiments, is to get a single sequence to fold into different shapes, depending on the surrounding molecules. Another big challenge is that not all parts of the sequence are meaningful for all possible inputs. Hence, to prevent them from interfering with subsequent operations in the Oritatami folding pathway we must structure the unused portions of the sequence depending on the context in which it folds.

Next, we introduce general design techniques to solve these challenges and program molecules. Our main result in this direction is an algorithm that is time linear in the sequence length that finds a rule for folding the sequence deterministically into a prescribed set of shapes, dependent on its local environment. This shows that the corresponding problem is fixed-parameter tractable, although we also prove it NP-complete in the number of possible environments.

1998 ACM Subject Classification F.1.1 Models of Computation, J.3 Life and Medical Sciences

Keywords and phrases Natural computing, Self-Assembly, Molecular Folding

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.43

1 Introduction

The process by which one-dimensional sequences of nucleotides or amino-acids acquire their complex three-dimensional geometries, which are key to their *function*, is a major puzzle of biology today. Understanding molecular folding will not only shed light on the origin and functions of the molecules existing in nature, it will also enable us to *control* the process

[*] C. Geary is a Carlsberg Postdoctoral Fellow supported by the Carlsberg Foundation and is supported by USNSF Grant no. 1317694 (http://molecular-programming.org). N. Schabanel is supported by Grants ANR-12-BS02-005 RDAM and IXXI-MOLECAL. S. Seki is in part supported by: the Academy of Finland; Postdoctoral Researcher Grant 13266670/T30606; JST Program, MEXT, Japan, No. 6F36; and JSPS Grants No. 15H06212 and No. 16H05854.

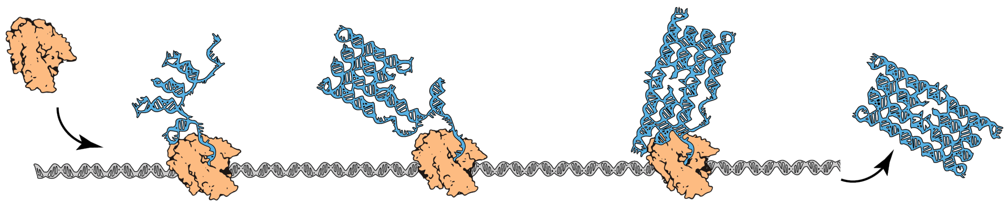


Figure 1 An RNA molecule folding over itself while being transcribed, as the experiments in [14].

more finely, and engineer artificial molecules with a wide range of uses, from performing missing functions inside living organisms, to producing precisely targeted drugs.

Biomolecular nano-engineering includes DNA self-assembly, which gave rise to an impressive number of successful experimental realizations, from arbitrary 2D shapes [23] to molecule cyclic machines [28], or counters [11]. First pioneered by Seeman [24], DNA nanotechnologies only really started to take off once a computer science model was devised by Winfree [26] to *program* molecular self assembly in a computer science way.

Since then, many models have been designed to refine different features of experiments: hierarchical self-assembly [4, 5], modeling the absence of a *seed*, kinetic tile assembly [26, 13], 3D and probabilistic tile assembly [6], among others.

The applications of DNA are limited by the high-fidelity of DNA base-pairing. The elaborate structures that researchers are able to produce with DNA are typically formed by the thermodynamically-driven assembly process of *annealing*, where many DNA strands are brought near equilibrium by heating them up to a high temperature and then cooling them down at a controlled rate, a folding process that is essentially incompatible with living cells.

By contrast, the RNA nanostrutures are designed to mimic the natural folding process, and are designed to fold under conditions that are cell-like. However, their assembly process is harder to program: intuitively, the shape depends on both the sequence and the environment, and the sequence is read *linearly*, independent from the environment. This contrasts with more classical programming models such as Turing machines, or even tile assembly, which are able to *jump* to different parts of the program depending on the input.

Another important difficulty is that even predicting the final shape of a sequence is still the center of active research, especially for proteins [29, 17, 18, 8, 21, 22, 16]

In particular, a large body of computer science literature focused on energy optimization, one of the main drivers of folding. For example, in different variants of the *hydrophobic-hydrophilic (HP) model* [9], it has been shown that the problem of predicting the most likely geometry (or *conformation*) of a sequence is NP-complete [25, 20, 2, 3, 7, 1], both in two and three dimensions, and in different variants of the model.

A few years ago, the *kinetics* of folding, which is the step-by-step dynamics of the reaction, has been demonstrated by biochemists to play a role in the final shape of molecules [15], even a *prevalent role* in the case of RNA [12]. In recent experimental results [14], researchers have even been able to *control* this mechanism to engineer their own shapes out of RNA.

This paper introduces a new model of RNA folding, intended to capture the kinetics of folding and model the experiments in [14]. In particular, it focuses on the *co-transcriptional* nature of RNA folding, which is the fact that, in real conditions, molecules fold while being transcribed (see Figure 1): in computer science terms, the folding process is a *local energy optimization*, or otherwise put, a *greedy algorithm*.

The first experimental results have used a standard benchmark: making simple shapes, such as squares (as shown for instance on Figure 2). With the new model introduced in

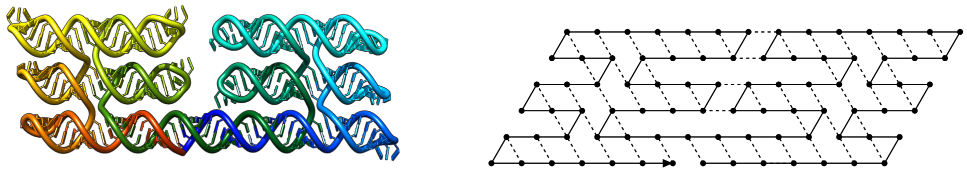


Figure 2 The design of a rectangle, co-transcriptionally folded with RNA, and the corresponding path on the triangular lattice, where each bead corresponds to two to four nucleotides.

this paper, our goal is twofold: first, explore the engineering possibilities of this mechanism, in order to make arbitrary shapes and structures. Then, the other aim of our study is to understand the complexity of sequence operations, to understand the computational processes which led to the creation of complex molecular networks.

Main contributions. In our model, called Oritatami, we consider a sequence of "beads", which are abstract basic components, standing for nucleotides or even sequences of nucleotides (also called *domains*). In Oritatami, only the latest produced beads of the molecules are allowed to move in order to adopt a more favorable configuration. The folding is driven by the respective attraction between the beads.

Our main construction is a *binary counter.* Counters are an essential component of many sophisticated constructions in biological computing, in particular in tile assembly [10, 19]. Counters are also an important benchmark in experiments [11].

▸ **Theorem 1.** *There is a fixed periodic sequence* $0, 1, \ldots, 59, 0, 1, \ldots$ *of period 60 whose rule is given in Fig. 11, which, when started from a seed encoding an integer* x *in binary with at most* $2k+1$ *bits for some* k*, folds into a structure encoding* $x+1$*,* $x+2$*,* $\ldots$*,* $2^{2k+1}-1$*, on the successive rows of the triangular grid.*

We *prove the correctness* of this construction by designing an abstract module system to handle the complexity of the base mechanism of the model, which is about as low-level as assembly code in more standard computing models.

We then show a generic construction method in this model, which we applied to automate parts of the design of the counter. Moreover, this result helps understanding the computational complexity of sequence programming. Precisely, we prove two results in this direction:

▸ **Theorem 2.** *Designing a single sequence that folds into different target shapes in a set of surrounding environments, is NP-complete in the number of environments.*

More surprisingly, it turns out that there is an algorithm to solve this problem in time *linear in the length of the sequence.* This algorithm is also practical, as we were able to use it to find sequences for our main construction:

▸ **Theorem 3.** *The sequence design problem is FPT with respect to the length* ℓ *of the sequence: there is an algorithm linear in* ℓ *(but exponential in the number of environments) to design a single sequence that folds into the target shapes in the given environments.*

2 Model and Main Results

2.1 Model

Oritatami system. Oritatami is about the folding of finite sequences of beads, each from a finite set B of *bead types*, using an attraction rule ♥, on the triangular lattice graph $\mathbb{T} = (\mathbb{Z}^2, \sim)$

where $(x,y) \sim (u,v)$ if and only if $(u,v) \in \{(x-1,y),(x+1,y),(x,y+1),(x+1,y+1),(x-1,y-1),(x,y-1)\}$.

A *conformation* c of a sequence $w \in B^*$ is a self-avoiding path of length ℓ labelled by w in $\mathbb{T}$, i.e. a path whose vertices $c_1,\dots,c_\ell$ are pairwise distinct and labelled by the letters of w. A *partial conformation* of a sequence w is a conformation of a prefix of w. For any partial conformation c of some sequence w, an *elongation* of c by k beads is a partial conformation of w of length $|c|+k$. We denote by $\mathcal{C}_w$ the set of all partial conformations of w (the index w will be omitted when the context is clear). We denote by $c^{\rhd k}$ the set of all elongations by k beads of a partial conformation c of a sequence w and by $c^{\lhd k}$ the singleton containing the prefix of length $|c|-k$ of c.

An *Oritatami system* $\mathcal{O} = (p, \heartsuit, \delta)$ is composed of (1) a (possibly infinite) *primary structure* p, which is a sequence of *beads*, of a type chosen from a finite set B, (2) an *attraction rule*, which is a symmetric relation $\heartsuit \subseteq B^2$ and (3) a parameter δ called the *delay time*.

Given an attraction rule $\heartsuit$ and a conformation c of a sequence w, we say that there is a *bond* between two adjacent positions c_i and c_j of c in $\mathbb{T}$ if $w_i \heartsuit w_j$. The *energy* of a conformation c of w, written $\mathrm{E}(c)$, is the negation of the number of bonds within c: formally, $\mathrm{E}(c) = -|\{(i,j) : c_i \sim c_j,\ j > i+1, \text{ and } w_i \heartsuit w_j\}|$.

Oritatami dynamics. A *dynamics* for a sequence w is a function $\mathcal{D} : 2^{\mathcal{C}} \to 2^{\mathcal{C}}$ such that for all subset S of partial conformations of length ℓ of w, $\mathcal{D}(S)$ is a subset of the elongations by one bead of the partial conformations in S (thus, partial conformations of length $\ell+1$).

Given an Oritatami system $\mathcal{O} = (p, \heartsuit, \delta)$ and a *seed conformation* σ of a seed sequence s of length ℓ, the set of partial conformations of the primary structure p at time t under dynamics $\mathcal{D}$ is $\mathcal{D}^t_{sp}(\{\sigma\})$, [1] i.e. the set of all elongations by t beads of the seed conformation prolongated by the primary structure according to dynamics $\mathcal{D}$.

We explore greedy folding dynamics where only the most recently transcribed beads can move, all other beads remain in place. These are controlled by integer parameter δ (in this article, $\delta \leqslant 4$). Several dynamics could model the "greedy" nature of the process. We choose the following dynamics, called the *hasty dynamics*:

The hasty dynamics does not question previous choices but chooses the energy-minimal positions for the δ last beads among all elongations of the previously adopted partial conformations. It lets the $\delta - 1$ already placed last beads where they are and abandons the extension of a conformation if no extension with the newly transcribed bead allows to reach a lowest energy conformation available for the δ last beads. Formally, $\mathcal{H}$ starts from a set of partial conformations, elongates each of them by one bead, and keeps the elongated conformations that have minimal energy among those who share the same prefix of length $|\sigma| + t - \delta$:

$$\mathcal{H}(S) = \bigcup_{\gamma \in S^{\lhd(\delta-1)}} \left(\operatorname*{arg\,min}_{c \in (S^{\rhd 1}) \cap (\gamma^{\rhd \delta})} \mathrm{E}(c) \right)$$

An Oritatami system $\mathcal{O} = (p, \heartsuit, \delta)$ is *deterministic* for dynamics $\mathcal{D}$ and seed σ of sequence s if for all $i \geqslant 1$, the position of the i-th bead of p is deterministic at time $i - 1 + \delta$, i.e. if for all $i \geqslant 1$, $|\{c_{|\sigma|+i} : c \in \mathcal{D}^{i-1+\delta}_{sp}(\{\sigma\})\}| = 1$. We say that $\mathcal{O}$ *stops* at time t with seed σ and dynamics $\mathcal{D}$ if $D^t_{sp}(\{\sigma\}) = \varnothing$ and $\mathcal{D}^z_{sp}(\{\sigma\}) \neq \varnothing$ for $z < t$. The folding process may

[1] Given two words $a, b \in B^*$, we denote by ab their concatenation.

only stop because of a geometric obstruction (no more elongation are possible because the conformation gets trapped in a closed area).

3 Folding a binary counter

3.1 General idea of the construction

Our construction works with $\delta = 4$. The counter is implemented by folding the periodic sequence of bead types $0, 1, \ldots, 58, 59, 0, 1, \ldots$ with period 60. Our construction proceeds in zig-zags as the classic implementation of a counter with a sweeping Turing machine whose head goes back and forth between the two ends of the coding part of the tape. Each pass is 3-rows thick and folds each part of the molecule into a parallelogram of size 4×3 or 6×3 except for the last and the first parts of each pass which are folded into parallelograms of size 3×6 to accomplish the U-turn downwards and start the next pass. The *zig pass*, folding three rows at a time from right to left, computes the carry propagation in the current value of the counter. The *zag pass*, folding three rows at a time from left to right, writes down the bits of the newly incremented value, and gets the folding to resume at the right-hand side of the conformation.

The molecule is semantically divided into 4 parts, called *modules*:

- Module A (beads 0–11, in blue in all figures): the First Half-Adder
- Module B (beads 12–29, in red in all figures): the Left-Turn module
- Module C (beads 30–41, in blue in all figures): the Second Half-Adder
- Module D (beads 42–59, in red in all figures): the Right-Turn module

Encoding. The current value of the counter is encoded in standard binary with the most significant bit to the left. Each bit is encoded into a specific folding of the modules A and C of the molecule in the rows corresponding to a zag pass: namely folding A0 and C0 for 0, and A1 and C1 for 1. During the zig pass, the *value of the carry* is encoded by the position of the molecule when it starts to fold Module A or C: `carry` $= 0$ if Module A or C starts to fold in the top row; `carry` $= 1$ if Module A or C starts to fold from the bottom row.

In the zig pass ($\leftarrow$), modules A and C "read" from the row above the value encoded into the folding in the row above during the previous zag-phase (or in the seed conformation for the first zig pass), and fold into a shape (called a *brick*, see Section 4) A00, A10, A01, A11 or C00, C10, C01, C11 accordingly where Axc is the brick corresponding to the case where x is the bit read in the row above and c is the carry. In the zig pass, modules B and D just propagate the carry value (0 or 1, i.e. start from top or bottom row) output by the preceding module A or C to the next.

When the zig pass reaches the leftmost part of the row on top, the beads there forces the module B to adopt the Left-turn shape which reverses the folding direction and starts the next zag pass.

In the zag pass ($\rightarrow$), modules A and C "read" the bricks above Axc or Cxc and folds into the bricks that encodes the corresponding bits, namely Ay or Cy where $y = (x + c) \mod 2$. There are no carry propagation and all the modules B and D fold into the same brick B2 or D2 in this pass.

When the molecule reaches the rightmost part of the row on top of it, the special beads there force the module D to fold into the Right-turn brick which reverses the folding direction and starts the next zig pass.

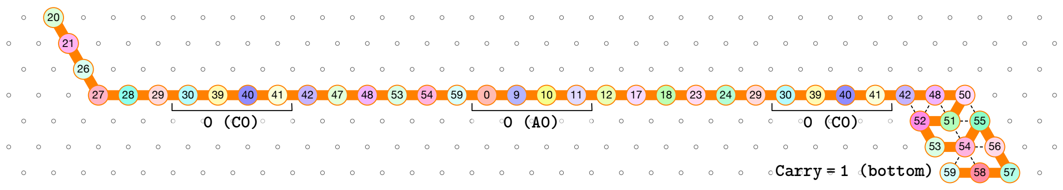


Figure 3 The seed conformation for the 3-bits counter encoding the three bits 000 as the initial value of the counter.

3.2 The first two passes of the folding

Let's run the first passes of the 3 bits counter to get acquainted with the process.

The seed conformation. is shown in Fig. 3. The seed conformation for the $(2k+1)$-bit counter is composed of $4k+3$ parts:

- The first part $20 \underset{\mathrm{SE}}{\searrow} 21 \underset{\mathrm{SE}}{\searrow} 26 \underset{\mathrm{SE}}{\searrow} 27 \underset{\mathrm{E}}{\rightarrow} 28 \underset{\mathrm{E}}{\rightarrow} 29$, made of beads from Module B, encodes a sequence that will trigger the carriage return at the end of the next zig pass.
- The central part consists in k repetitions of the same sequence of 4 patterns, plus an extra repetition of the first pattern at the end (the central part consists thus in $4k+1$ parts in total):
  - $30 \underset{\mathrm{E}}{\rightarrow} 39 \underset{\mathrm{E}}{\rightarrow} 40 \underset{\mathrm{E}}{\rightarrow} 41$ encoding a bit 0 using beads from Module C,
  - followed by $42 \underset{\mathrm{E}}{\rightarrow} 47 \underset{\mathrm{E}}{\rightarrow} 48 \underset{\mathrm{E}}{\rightarrow} 53 \underset{\mathrm{E}}{\rightarrow} 54 \underset{\mathrm{E}}{\rightarrow} 59$ encoding nothing but padding using beads from Module B,
  - followed by $0 \underset{\mathrm{E}}{\rightarrow} 9 \underset{\mathrm{E}}{\rightarrow} 10 \underset{\mathrm{E}}{\rightarrow} 11$ encoding a bit 0 using beads from Module A,
  - followed by $12 \underset{\mathrm{E}}{\rightarrow} 17 \underset{\mathrm{E}}{\rightarrow} 18 \underset{\mathrm{E}}{\rightarrow} 23 \underset{\mathrm{E}}{\rightarrow} 24 \underset{\mathrm{E}}{\rightarrow} 29$ encoding nothing but padding using beads from Module D.

  Note the symmetry by a shift of 30 of the beads values in the patterns involving Modules A and C, and Modules B and D.
- The last part $42 \underset{\mathrm{E}}{\rightarrow} 48 \underset{\mathrm{E}}{\rightarrow} 50 \underset{\mathrm{SW}}{\swarrow} 51 \underset{\mathrm{W}}{\leftarrow} 52 \underset{\mathrm{SE}}{\searrow} 53 \underset{\mathrm{E}}{\rightarrow} 54 \overset{\mathrm{NE}}{\nearrow} 55 \underset{\mathrm{SE}}{\searrow} 56 \underset{\mathrm{SE}}{\searrow} 57 \underset{\mathrm{W}}{\leftarrow} 58 \underset{\mathrm{W}}{\leftarrow} 59$, made of beads from Module D, encodes a sequence that will first initiate the next zig pass and later trigger the carriage return at the end of the next zag pass.

Note that the seed conformation ends at the bottom row of the upcoming zig pass, which encodes thus that initially the carry is 1.

The first zig pass (←). Each zig pass starts with a carry equal to 1, i.e. starts folding from the bottom row. In the first zig pass, the first module A (see Figure 4) folds into the brick A01, encoding the bit $1 = 0 + 1$ with no carry propagation, as a consequence of the carry being 1 and of reading the first bit, 0, from the seed above. Note that the folding A01 ends at the top row, encoding that the carry is now 0. The reading of the bit from the seed is accomplished by the way the module binds to the seed which shapes the module accordingly as we will see in details in Section 3.3.

Then, as illustrated in Fig. 5, the next modules B, C, D, and A fold into shapes B0, C00, D0 and A00 respectively: B0 and D0 meaning that no carry is propagated (they start and end on the top row); and C00 and A00 meaning that the (input) carry is 0 and the bit read from the seed is 0.

Finally, as illustrated in Fig. 6, the last module, B, of the zig pass binds to the 3-beads long carriage-return pattern at the leftmost part of the seed and folds into the shape BT

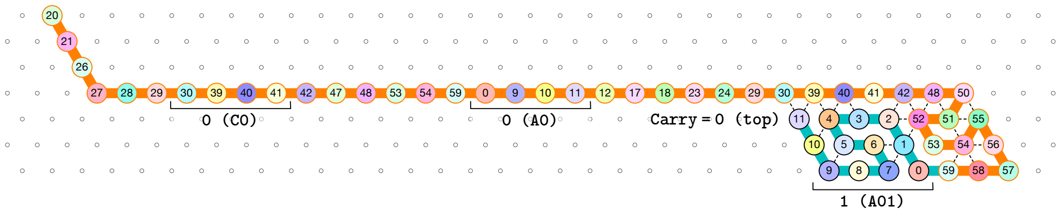


Figure 4 The folding of the first module, A: starting with a carry 1, encoded by the position of the first bead (on the bottom row), this module "reads" a 0 from the seed by binding to the seed, and folds into A01, encoding a 1 with no carry propagation, as encoded by the position of the last bead (on the top row of the module).

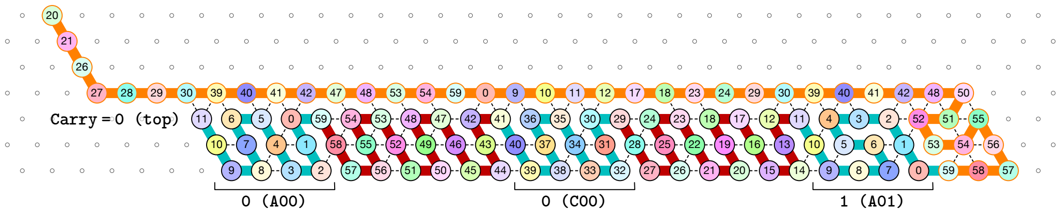


Figure 5 The folding of the central part of the first zig pass in the 3-bits counter.

conducing the molecule to go down by 6 rows, reverse direction and start the following zag pass. Note that the bottom of the shape BT contains the exact same carriage-return pattern.

The first zag pass (→). The zag pass is mostly a normalization pass as illustrated in Fig. 7–8. It progresses from left to right and computes the new value of each bit by rewriting each shape A00 and A11 as C0, C00 and C11 as A0, A10 and A01 as C1, and C10 and C10 as A0. Shapes A0 and C0 encode 0, and Shapes A1 and C1 encode 1, both to be read during the upcoming zig pass. Modules B and D just fold into the shapes B2 and D2 respectively, encoding nothing but padding.

Finally, as illustrated in Fig. 9, the last module, D, of the zag pass binds to the 3-beads long carriage-return pattern in the rightmost part of the seed and folds into the shape DT conducing the molecule to go down by 6 rows, reverse direction and start the next zig pass. Note that, as for the shape BT, the bottom of the shape DT contains the exact same carriage-return pattern.

Figure 10 shows the 3-bits counter folded upto the value $3 = 011$ in binary. One can observe the shape A11 in the second zig pass. A11 corresponds to reading a 1 with a carry 1 which propagates the carry: indeed, the folding ends at the bottom row which propagates the carry to the next module C which folds into C01 as it reads a 0 from above with carry 1. Note that shape A11 is then rewritten as C0 in the following zag pass below.

3.3 How does computation take place: modules, functions, states and environment

Each module A, B, C and D implement various "functions" that are "called" when the molecule is folded. Which function is called depends on two things:

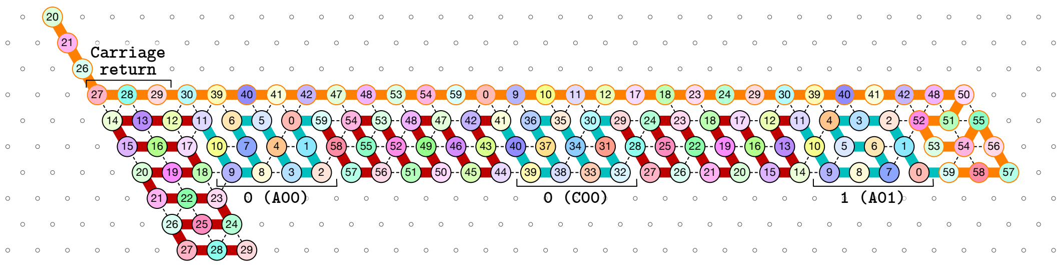


Figure 6 In our construction, the leftmost three beads of any conformation are different from the other beads the left U-turn module binds to inside the zig or zag pass: when the left U-turn module folds next to these bead types, it “triggers” the production of an actual U-turn.

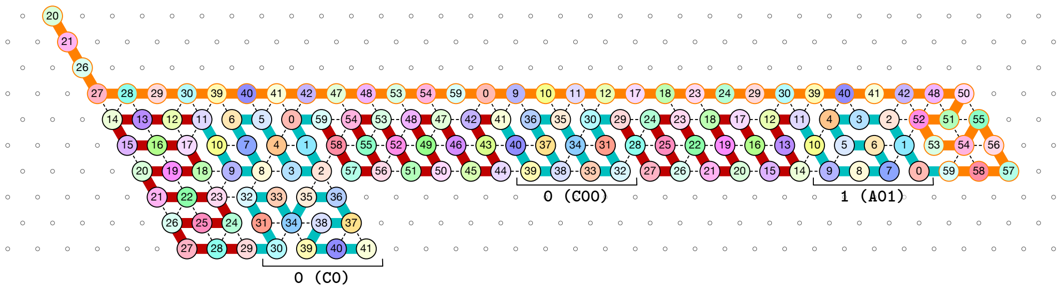


Figure 7 During the zag pass, all modules start from the bottom row, computing the value of each new bit by rewriting shapes A00 and A11 as C0, C00 and C11 as A0, A10 and A01 as C1, and C10 and C10 as A1.

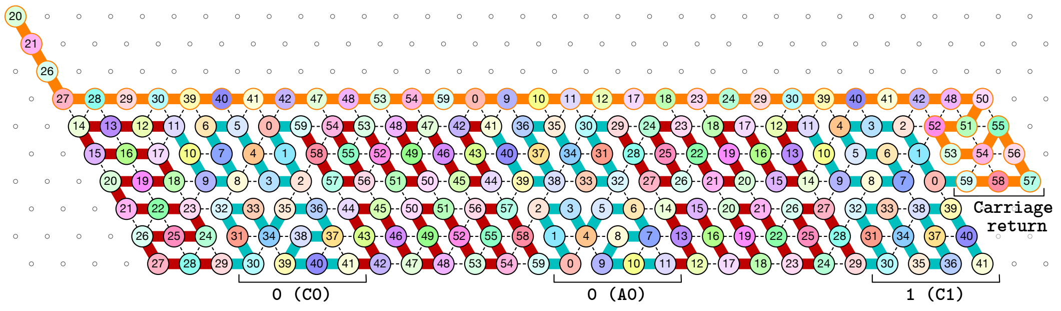


Figure 8 At the end of the first zag pass, the new value of each bit have been encoded into shapes: A0 or C0 for bits equal to 0, A1 or C1 for bit equal to 1.

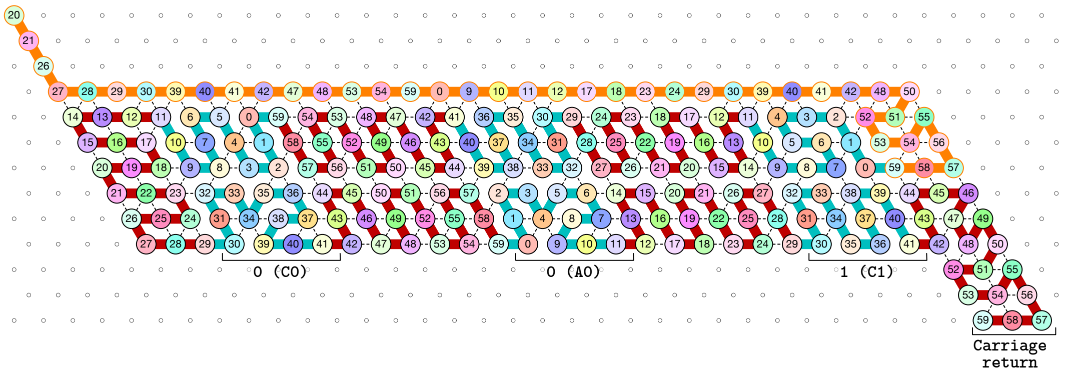


Figure 9 Finally, at the end of the first zag pass, the last module D binds to the carriage-return pattern in the seed and fold into the shape DT to accomplish the right U-turn from which the next zig pass can start.

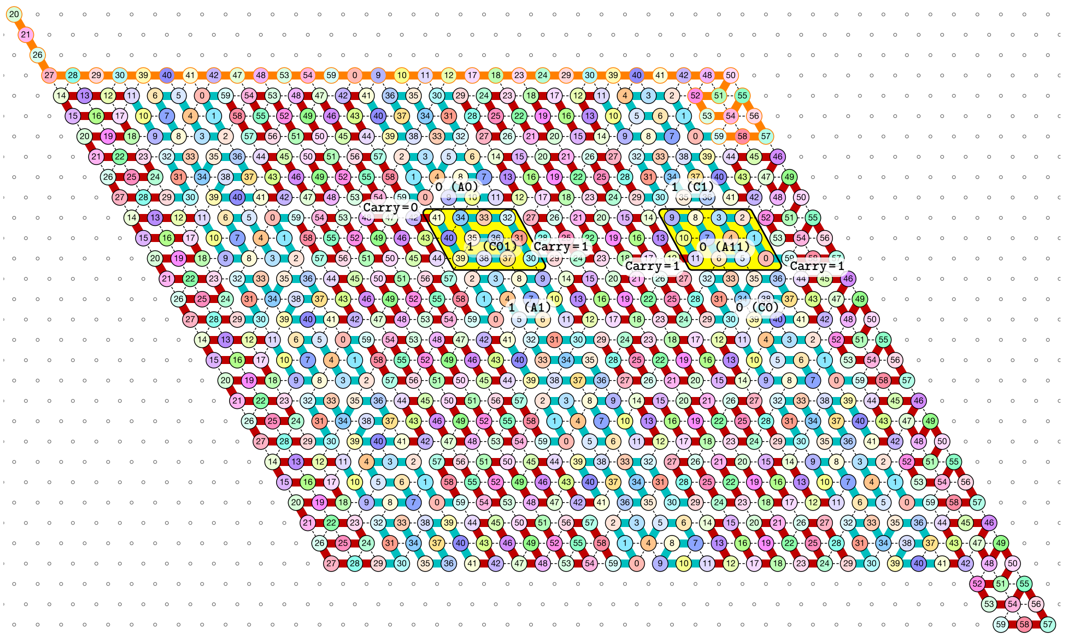


Figure 10 The folding of the 3-bits counter upto value 4 = 100 in binary. Observe the carry propagation in the second zig pass.

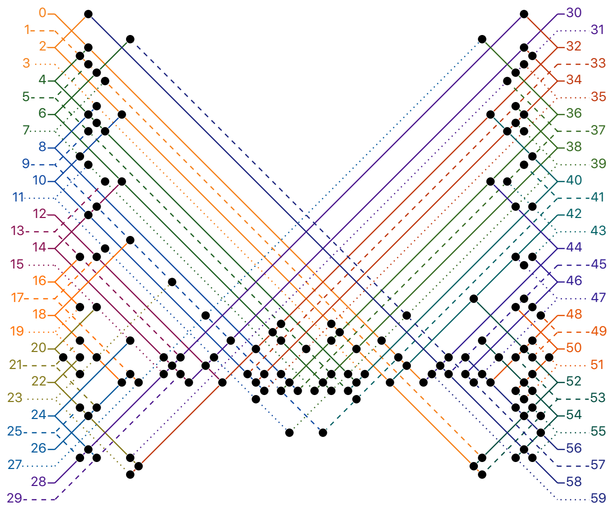


Figure 11 *The rule ♥ of the Counter Oritatami system*: in this diagram, we have b ♥ b' iff there is a bullet • at the intersection of one the two lines coming from b and from b'; for instance, we have 4 ♥ 8 but not 4 ♥ 7.

the current "state" of the molecule: here, the *state* is whether the carry is 0 or 1. As mentionned earlier this is encoded in the position of the molecule when the module starts to fold: it starts in the top row if the carry is 0; in the bottom row if the carry is 1.

the local environment of the molecule: the *environment*, i.e. the beads already placed around the current area where the folding take place, acts as the memory in the computation.

The position where the folding of a module starts, determines which beads of a given module will be exposed to and interact with the environment. Then, by creating bonds (or not) with the environment, each module will adopt a specific shape. Therefore, the possible binding schemes will be different depending of this initial position. Similarly, depending on the beads already placed in the environment, the part of the module exposed to it will adopt one form or another depending on how many bonds it can create with the environment. Adopting the language of computer science: the position at which a module starts to fold, determines which *function* of the module is called; the function then *reads the input* encoded by the beads already placed in the environment.

Fig. 12 provides a precise description on how the function of the Half-Adder Module C are implemented in the zig pass. As the zig pass goes from right to left, the figure is meant to be read from right to left. In the zig pass, Module C implements two functions: 1) Add 1 to the bit above and propagate the carry if needed, or 2) Copy the bit above unchanged. Add is called if the carry is 1 at the beginning of the folding and Copy is called if the carry is 0. The following step-by-step description of the folding explains how:

Beads 30-33 (rightmost column in Fig. 12): *if the carry is* 0 *at start,* then bead 30 is able to bind with beads 11 and 12 from the environment and depending on whether the input encodes a bit 0 or 1, bead 32 will be able to bind to bind either to 28 or to 5 and 6 respectively. *Whereas if the carry is* 1*,* then bead 30 cannot reach beads 11 and 12. Thus, these are beads 31 and 32 that will bind with beads 10 and 12 from the environment, giving to the molecule a completely different shape.

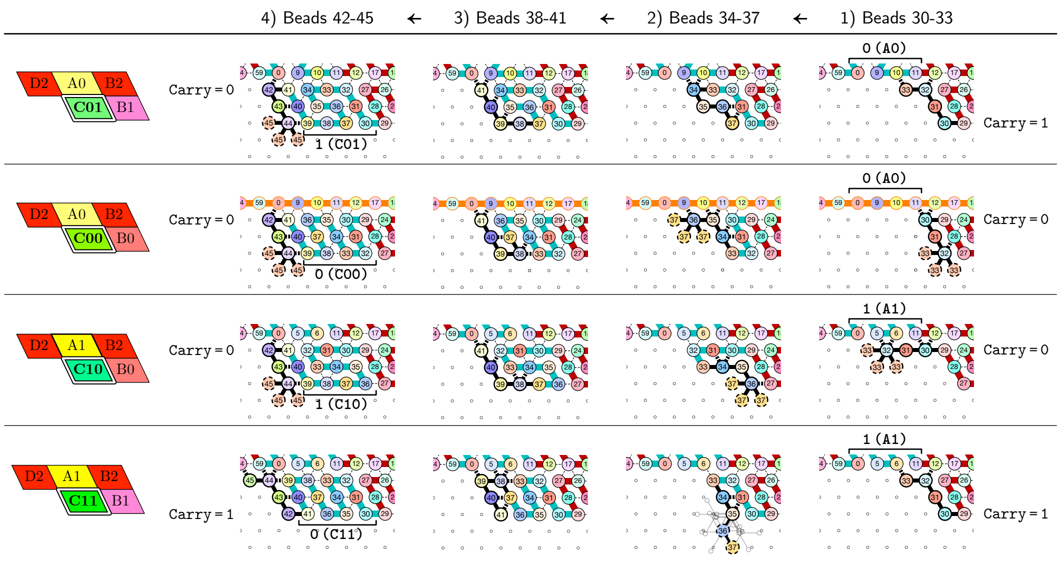


Figure 12 An illustration of how the module C applies a different function which results in different foldings according to the initial state of the molecule (carry = 0 or 1) at the beginning of the folding of the module, and to the environment (the bit 0 or 1 encoded) read above by the function. This figure is meant to be read from right to left (zig pass ←).

Beads 34-37 with carry = 0 at start: as bead 34 is attracted by both beads 30 and 31, the molecule folds upon itself similarly but with a different rotation depending on whether it had read a 0 or a 1 in the environement above: vertically if it has read a 0, horizontally if it has read a 1. Bead 36 attracted by beads 9 and 27 fixes the end of the tip in place leaving bead 37 free to move for now.

Beads 34-37 with carry = 1 at start: Bead 34 is attracted by beads 9 and 10 encoding a bit 0 above which allows beads 36 and 37 to bind with 31 as well, and but prefers to bind with 31 together with 35 otherwise. This induces two different shapes: the beginning of a "wave" pattern ( ) if the bit read above is 0; or the beginning of a "switchback" pattern ( ) if the bit read is 1.

Beads 38-41, without carry propagation (carry = 0, or carry = 1 and bit read = 0): in these three cases the folding of the beads 38-41 starts from the same position. As the environment is different for each of them, we could design the rule so that this part of the module prefers to adopt the same shape, climing along the part already folded to the top row to start the next module with a carry = 0.

Beads 38-41, with carry propagation (carry = 1 and bit read = 1): because the switchback pattern is upside down in this case, bead 37 stays low and bead 38 can firmly attach to the top with beads 5, 6 and 33, and the tip of the module folds downwards as 40 and 41 are attracted by 37. This ensures that the folding of the module ends at the bottom row, propagating the carry = 1 to the next module.

Note that the bottom rows of the four resulting foldings differ significantly: 39–38–37–30 for C01, 39–38–33–32 for C00, 39–38–37–36 for C10, and 41–36–35–30 for C11. This will allow to distinguish between them in the following zag pass to write the correct bit for the new value of the counter.

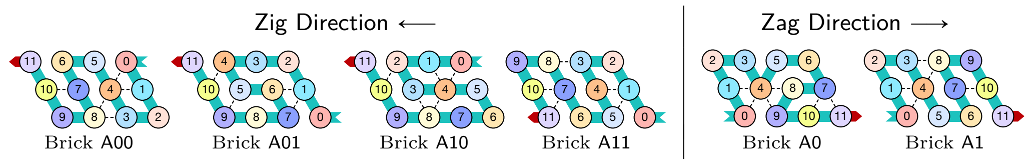


Figure 13 The six different bricks for Module A, First Half-Adder (beads 0–11).

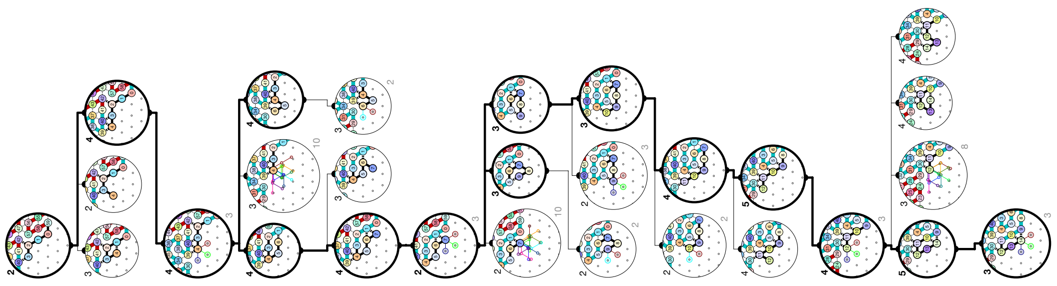
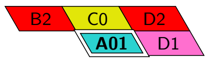


Figure 14 The folding certificate for the brick A01 in the environment:



4 Proof Sketch of Theorem 1

We call *bricks* the various conformations each module will adopt in the final folding, as they are the bricks upon which the whole folding is built. Designing the bricks is one of the biggest challenges when programming Oritatami. There are six bricks for Modules A and C: Axc and Cxc in the zig pass where $x, c \in \{0,1\}$ are the bit read from the brick above (Ax or Cx) and the (input) carry from the preceding module (Bc or Dc); and Ay and Cy in the zag pass where y is the bit written: namely $y = x + c \bmod 2$ if the brick above is Axc or Cxc. There are four bricks for Modules B and D: Bc' and Dc' in the zig pass where $c' \in \{0,1\}$ is the carry output by the preceding brick Cxc or Axc, namely $c' = x \wedge c$; B2 and D2 in the zag pass. Figure 13 lists the six bricks for Module A.

The proof works in two stages. First, we describe the final target folding and show that it implements correctly a binary counter: i.e. that the bricks implement correctly the relationships between the y, x, c and c' described above. Second, we show that indeed, each module folds into the expected brick in each given environment. For that matter, we produce a *folding certificate*, which shows for each step all the possible extension of the current conformation together with the number of bonds created (the number in the north-east corner), grouping together several extensions when they share a common path (the number of paths groupes are in the south-east corner). Figure 14 displays the folding certificate for Brick A01 surrounded by the bricks B2, C0, D2 and D1.

5 Rule design is NP-hard and FPT

Once we have agreed on the desired conformations in our construction, an important issue is to find an attraction rule such that a primary structure folds into its correct functions. The *rule design problem* consists in: given a delay time δ, a list of $n > 0$ seeds $\sigma_1, \sigma_2, \ldots, \sigma_n$, and a list of n conformations $c_1, c_2, \ldots, c_n$ of the same length ℓ, output an attraction rule ♥ such that for all $i \in \{1, 2, \ldots, n\}$, Oritatami system $\mathcal{O}_i = (s, \sigma_i, ♥, \delta)$ deterministically folds into conformation c_i, where $s = \langle 0, 1, \ldots, \ell - 1\rangle$.

Theorem 2 (proof omitted) shows by a reduction from 3-SAT that this problem is NP-hard. However, Theorem 3 (proof omitted) shows using the locality of the bindings that it is fixed-parameter tractable with respect to the length of the sequence ℓ. Then, as n and δ are usually small for the designs we considered, we could use this algorithm to help us designing an attraction rule compatible with our brick designs.

6 Perspectives

The purpose of our new model is not to be entirely accurate with respect to phenomena observed in nature, but instead to start developing an intuition about the *kind of problem* that need to be solved in order to engineer RNA shapes, and later, even proteins.

In the future, a number of extensions of this model seem natural. In particular, extending it with a more realistic notion of *thermodynamics and molecular agitation*. Using existing works in molecular dynamics [27], would allow to explore stochastic optimization processes.

Acknowledgments. The authors thank Abdulmelik Mohammed, Andrew Winslow, Damien Woods for discussions and encouragements.

References

1 O. Aichholzer, D. Bremner, E. D. Demaine, H. Meijer, V. Sacristán, and M. Soss. Long proteins with unique optimal foldings in the H-P model. *Computational Geometry*, 25(1–2):139–159, 2003.

2 J. Atkins and W. E. Hart. On the intractability of protein folding with a finite alphabet of amino acids. *Algorithmica*, 25(2–3):279–294, 1999.

3 Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.

4 Sarah Cannon, Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Matthew J. Patitz, Robert Schweller, Scott M. Summers, and Andrew Winslow. Two hands are better than one (up to constant factors). In *STACS: Proceedings of the Thirtieth International Symposium on Theoretical Aspects of Computer Science*, pages 172–184, 2013. Arxiv preprint: `1201.1650`.

5 Ho-Lin Chen and David Doty. Parallelism and time in hierarchical self-assembly. In *SODA 2012: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1163–1182. SIAM, 2012.

6 Matthew Cook, Yunhui Fu, and Robert T. Schweller. Temperature 1 self-assembly: Deterministic assembly in 3D and probabilistic assembly in 2D. In *SODA 2011: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2011. Arxiv preprint: `arXiv:0912.0027`.

7 Pierluigi Crescenzi, Deborah Goldman, Christos Papadimitriou, Antonio Piccolboni, and Mihalis Yannakakis. On the complexity of protein folding. *Journal of computational biology*, 5(3):423–465, 1998.

8 R. Das and D. Baker. Automated de novo prediction of native-like RNA tertiary structures. *PNAS*, 104:14664–14669, 2007.

9 K.A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, 1985.

10 David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. The tile assembly model is intrinsically universal. In *FOCS 2012*, pages 439–446, October 2012.

11 Constantine Glen Evans. *Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly.* PhD thesis, California Institute of Technology, 2014.

12 Kirsten L. Frieda and Steven M. Block. Direct observation of cotranscriptional folding in an adenine riboswitch. *Science*, 338(6105):397–400, 2012.

13 Kenichi Fujibayashi, David Yu Zhang, Erik Winfree, and Satoshi Murata. Error suppression mechanisms for dna tile self-assembly and their simulation. *Natural Computing: an international journal*, 8(3):589–612, 2009. doi:10.1007/s11047-008-9093-9.

14 Cody Geary, Paul W. K. Rothemund, and Ebbe S. Andersen. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science*, 345:799–804, 2014.

15 Boyle J, Robillard G, and Kim S. Sequential folding of transfer RNA. a nuclear magnetic resonance study of successively longer tRNA fragments with a common 5' end. *J Mol Biol*, 139:601–625, 1980.

16 Hosna Jabbari and Anne Condon. A fast and robust iterative algorithm for prediction of rna pseudoknotted secondary structures. *BMC bioinformatics*, 15(1):147, 2014.

17 D. H. Mathews, M. D. Disney, J. L. Childs, S. J. Schroeder, M. Zuker, and D. H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *PNAS*, 101:7287–7292, May 2004. doi:10.1073/pnas.0401799101.

18 D.H. Mathews. Revolutions in rna secondary structure prediction. *Journal of molecular biology*, 359(3):526–32, 2006. doi:10.1016/j.jmb.2006.01.067.

19 Pierre-Étienne Meunier, Matthew J. Patitz, Scott M. Summers, Guillaume Theyssier, Andrew Winslow, and Damien Woods. Intrinsic universality in tile self-assembly requires cooperation. *SODA 2014: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2014.

20 M. Paterson and T. Przytycka. On the complexity of string folding. In F. Meyer and B. Monien, editors, *ICALP 1996*, volume 1099 of *LNCS*, pages 658–669. Springer Berlin Heidelberg, 1996.

21 M. Popenda, M. Szachniuk, M. Antczak, K.J. Purzycka, P. Lukasiak, N. Bartol, J. Blazewicz, and R.W. Adamiak. Automated 3D structure composition for large RNAs. *Nucleic Acids Research*, 40(14):e112, 2012. doi:doi:10.1093/nar/gks339.

22 Elena Rivas. The four ingredients of single-sequence rna secondary structure prediction. a unifying perspective. *RNA Biol*, 10(7):1185–1196, Jul 2013. 23695796[pmid]. doi:10.4161/rna.24971.

23 Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, March 2006. doi:10.1038/nature04586.

24 Nadrian C. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247, 1982.

25 R. Unger and J. Moult. Finding the lowest free energy conformation of a protein is an NP-hard problem: proof and implications. *Bulletin of Mathematical Biology*, 55(6):1183–1198, 1993.

26 Erik Winfree. *Algorithmic Self-Assembly of DNA.* PhD thesis, Caltech, June 1998.

27 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of ITCS 2013: Innovations in Theoretical Computer Science*, pages 353–354, 2013.

28 Bernard Yurke, Andrew J Turberfield, Allen P Mills, Friedrich C Simmel, and Jennifer L Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406(6796):605–608, 2000.

29 Michael Zuker and David Sankoff. Rna secondary structures and their prediction. *Bulletin of Mathematical Biology*, 46(4):591–621, 1984. doi:10.1007/BF02459506.

Connected Reversible Mealy Automata of Prime Size Cannot Generate Infinite Burnside Groups [*]

Thibault Godin[1] and Ines Klimann[2]

1 **Univ. Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 8243 CNRS, F-75013 Paris, France** godin@liafa.univ-paris-diderot.fr
2 **Univ. Paris Diderot, Sorbonne Paris Cité, IRIF, UMR 8243 CNRS, F-75013 Paris, France** klimann@liafa.univ-paris-diderot.fr

Abstract

The simplest example of an infinite Burnside group arises in the class of automaton groups. However there is no known example of such a group generated by a reversible Mealy automaton. It has been proved that, for a connected automaton of size at most 3, or when the automaton is not bireversible, the generated group cannot be Burnside infinite. In this paper, we extend these results to automata with bigger stateset, proving that, if a connected reversible automaton has a prime number of states, it cannot generate an infinite Burnside group.

1998 ACM Subject Classification F.4.3 Formal Languages, F.1.1 Models of Computation, G.2.m Miscellaneous

Keywords and phrases Burnside problem, Automaton groups, Reversibility, Orbit trees

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.44

1 Mealy automata and the General Burnside problem

The Burnside problem is a famous, long-standing question in group theory. In 1902, Burnside asked if a *finitely generated* group whose all elements have finite order –henceforth called a *Burnside group*– is necessarily finite [3].

The question stayed open until Golod and Shafarevitch exhibit in 1964 an infinite group satisfying Burnside's conditions [8, 9], hence solving the general Burnside problem. In the early 60's, Glushkov suggested using automata to attack the Burnside problem [6]. Later, Aleshin [2] in 1972 and then Grigorchuk [10] in 1980 gave simple examples of automata generating infinite Burnside groups. Over the years, automaton groups have been successfully used to solve several other group theoretical problems and conjectures such as Atiyah, Day, Gromov or Milnor problems; the underlying automaton structure can indeed be used to better understand the generated group.

It is remarkable that every known examples of infinite Burnside automaton groups are generated by non reversible Mealy automata, that is, Mealy automata where the input letters do not all act like permutations on the stateset. We conjecture that it is in fact impossible for a reversible Mealy automaton to generate an infinite Burnside group. Our past work with several co-authors has already given some partial results to this end. In [7] it is proven that non bireversible Mealy automata cannot generate Burnside groups. For the whole class of reversible automata, it has been proved in [12] that 2-state reversible Mealy automata cannot generate infinite Burnside groups. This result has later been extended in [13] to 3-state

* This work was partially supported by the French *Agence Nationale pour la Recherche*, through the Project **MealyM** ANR-JS02-012-01.

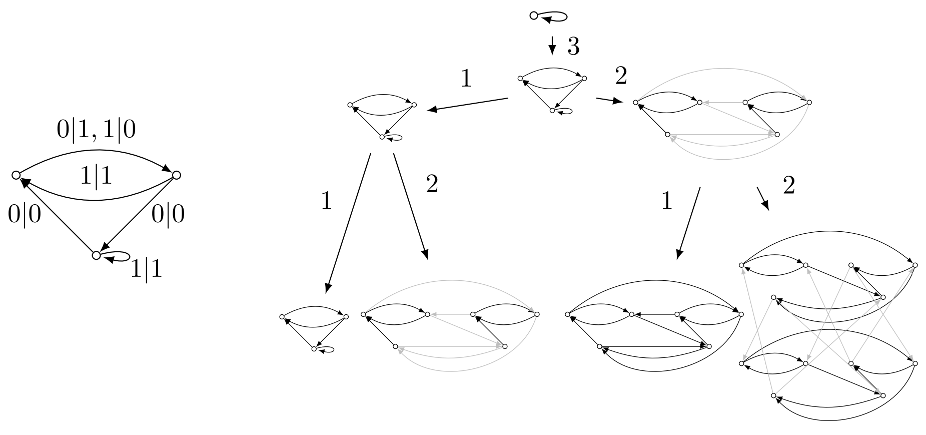


Figure 1 The Bellaterra automaton $\mathcal{B}$ and four levels of the orbit tree $\mathfrak{t}(\mathcal{B})$.

connected reversible automata. In this paper we generalize these results to any connected revertible automaton with a prime number of states:

▶ **Theorem.** *A connected invertible-reversible Mealy automaton of prime size cannot generate an infinite Burnside group.*

Our proof is inspired by the former work in the 3-state case of the second author with Picantin and Savchuk [13]. However the extension from 3 to any prime p required the introduction of a new machinery. This constitutes the main part of our paper, see Section 5.

The paper is organized as follows. In Section 2 we set up notations and recall useful facts on Mealy automata, automaton groups, and rooted trees. Then in Section 3 we link some characteristics of an automaton group to the connected components of the powers of the generating automaton. In Section 4 we introduce a tool developed in [13], the labeled orbit tree, that is used in Section 5 to define our main tool, the *jungle tree*. In this former section we also present some constructions and properties connected to this jungle tree. At last, in Section 6, we gather our information and prove our main result.

2 Basic notions

2.1 Groups generated by Mealy automata

We first recall the formal definition of an automaton. A *(finite, deterministic, and complete) automaton* is a triple $\big(Q, \Sigma, \delta = (\delta_i \colon Q \to Q)_{i\in\Sigma}\big)$, where the *stateset* Q and the *alphabet* Σ are non-empty finite sets, and the δ_i are functions.

A *Mealy automaton* is a quadruple $\mathcal{A} = (Q, \Sigma, \delta, \rho)$, where both (Q, Σ, δ) and (Σ, Q, ρ) are automata. In other terms, it is a complete, deterministic, letter-to-letter transducer with the same input and output alphabet. Its *size* $\#\mathcal{A}$ is the cardinality of its stateset.

The graphical representation of a Mealy automaton is standard, see Figure 1 left.

A Mealy automaton $(Q, \Sigma, \delta, \rho)$ is *invertible* if the functions ρ_x are permutations of Σ and *reversible* if the functions δ_i are permutations of Q.

In a Mealy automaton $\mathcal{A} = (Q, \Sigma, \delta, \rho)$, the sets Q and Σ play dual roles. So we may consider the *dual (Mealy) automaton* defined by $\mathfrak{d}(\mathcal{A}) = (\Sigma, Q, \rho, \delta)$. Obviously, a Mealy automaton is reversible if and only if its dual is invertible.

An invertible Mealy automaton is *bireversible* if it is reversible (i.e. the input letters of the transitions act like permutations on the stateset) and the output letters of the transitions act like permutations on the stateset.

Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a Mealy automaton. Each state $x \in Q$ defines a mapping from Σ^* into itself recursively defined by:

$$\forall i \in \Sigma,\ \forall \mathbf{s} \in \Sigma^*, \qquad \rho_x(i\mathbf{s}) = \rho_x(i)\rho_{\delta_i(x)}(\mathbf{s})\ .$$

The mapping ρ_x for each $x \in Q$ is length-preserving and prefix-preserving: it is the function *induced* by x. For $\mathbf{x} = x_1 \cdots x_n \in Q^n$ with $n > 0$, set $\rho_{\mathbf{x}} : \Sigma^* \to \Sigma^*, \rho_{\mathbf{x}} = \rho_{x_n} \circ \cdots \circ \rho_{x_1}$.

Denote dually by $\delta_i : Q^* \to Q^*, i \in \Sigma$, the functions induced by the states of $\mathfrak{d}(\mathcal{A})$. For $\mathbf{s} = s_1 \cdots s_n \in \Sigma^n$ with $n > 0$, set $\delta_{\mathbf{s}} : Q^* \to Q^*,\ \delta_{\mathbf{s}} = \delta_{s_n} \circ \cdots \circ \delta_{s_1}$.

The semigroup of mappings from Σ^* to Σ^* generated by $\{\rho_x, x \in Q\}$ is called the *semigroup generated by* $\mathcal{A}$ and is denoted by $\langle \mathcal{A} \rangle_+$. When $\mathcal{A}$ is invertible, the functions induced by its states are permutations on words of the same length and thus we may consider the group of mappings from Σ^* to Σ^* generated by $\{\rho_x, x \in Q\}$. This group is called the *group generated by* $\mathcal{A}$ and is denoted by $\langle \mathcal{A} \rangle$.

2.2 Terminology on trees

Throughout this paper, we will use different sorts of labeled trees. Here we set up some common terminology for all of them.

All our trees are rooted, *i.e.* with a selected vertex called the *root*. We visualize the trees traditionally as growing down from the root. A *path* is a (possibly infinite) sequence of adjacent edges without backtracking from top to bottom. A path is *initial* if it starts at the root of the tree. A *branch* is an infinite initial path. The lead-off vertex of a non-empty path $\mathbf{e}$ is denoted by $\top(\mathbf{e})$ and its terminal vertex by $\bot(\mathbf{e})$ whenever the path is finite.

The *level of a vertex* is its distance to the root and the *level of an edge* or *a path* is the level of its initial vertex.

If the edges of a rooted tree are labeled by elements of some finite set, the *label* of a (possibly infinite) path is the ordered sequence of labels of its edges.

Extending the notions of children, parents and descendent to the edges, we will say that an edge f is the *child* of an edge e if $\bot(e) = \top(f)$ (*parent* being the converse notion, and *descendent* the transitive closure).

All along this article we will follow walks on some trees. A *walk* is just a path in a tree, which is build gradually. In particular if $\mathbf{e}$ is a finite path (or can identify one), to say that it can be followed by f in some tree means that $\mathbf{e}f$ is (or identifies) also a path in that tree.

3 Connected components of the powers of an automaton

In this section we detail the basic properties of the connected components of the powers of a reversible Mealy automaton, as it has been done in [13]. The link between these components is central in our construction.

Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a reversible Mealy automaton.

By reversibility, all the connected components of its underlying graph are strongly connected.

Consider the powers of $\mathcal{A}$: for $n > 0$, its *n-th power* $\mathcal{A}^n$ is the Mealy automaton

$$\mathcal{A}^n = \left(Q^n, \Sigma, (\delta_i : Q^n \to Q^n)_{i \in \Sigma}, (\rho_{\mathbf{x}} : \Sigma \to \Sigma)_{\mathbf{x} \in Q^n} \right)\ .$$

By convention, $\mathcal{A}^0$ is the trivial automaton on the alphabet Σ.

As $\mathcal{A}$ is reversible, so are its powers and the connected components of $\mathcal{A}^n$ coincide with the orbits of the action of $\langle \mathfrak{d}(\mathcal{A}) \rangle$ on Q^n.

Since $\mathcal{A}$ is reversible, there is a very particular connection between the connected components of $\mathcal{A}^n$ and those of $\mathcal{A}^{n+1}$ as highlighted in [12]. More precisely, take a connected component $\mathcal{C}$ of some $\mathcal{A}^n$, and let $\mathbf{u} \in Q^n$ (also written $|\mathbf{u}| = n$) be a state of $\mathcal{C}$. Take also $x \in Q$ a state of $\mathcal{A}$, and let $\mathcal{D}$ be the connected component of $\mathcal{A}^{n+1}$ containing the state $\mathbf{u}x$. Then, for any state $\mathbf{v}$ of $\mathcal{C}$, there exists a state of $\mathcal{D}$ prefixed with $\mathbf{v}$:

$$\exists \mathbf{s} \in \Sigma^* \mid \delta_{\mathbf{s}}(\mathbf{u}) = \mathbf{v} \quad \text{and so} \quad \delta_{\mathbf{s}}(\mathbf{u}x) = \mathbf{v}\delta_{\rho_{\mathbf{u}}(\mathbf{s})}(x) \ .$$

Furthermore, if $\mathbf{u}y$ is a state of $\mathcal{D}$, for some state $y \in Q$ different from x, then $\delta_{\mathbf{s}}(\mathbf{u}x)$ and $\delta_{\mathbf{s}}(\mathbf{u}y)$ are two different states of $\mathcal{D}$ prefixed with $\mathbf{v}$, because of the reversibility of $\mathcal{A}^{n+1}$: the transition function $\delta_{\rho_{\mathbf{u}}(\mathbf{s})}$ is a permutation. Hence $\mathcal{D}$ can be seen as consisting of several copies of $\mathcal{C}$ and $\#\mathcal{C}$ divides $\#\mathcal{D}$. They have the same size if and only if, for each state $\mathbf{u}$ of $\mathcal{C}$ and any different states $x, y \in Q$, $\mathbf{u}x$ and $\mathbf{u}y$ cannot simultaneously lie in $\mathcal{D}$.

The connected components of the powers of a Mealy automaton and the finiteness of the generated group or of a monogenic subgroup are closely related, as shown in the following propositions (obtained independently in [13, 4]).

▶ **Proposition 1.** *A reversible Mealy automaton generates a finite group if and only if the connected components of its powers have bounded size.*

▶ **Proposition 2.** *Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be an invertible-reversible Mealy automaton and let $\mathbf{u} \in Q^+$ be a non-empty word. The following conditions are equivalent:*

(i) *$\rho_{\mathbf{u}}$ has finite order,*

(ii) *the sizes of the connected components of $(\mathbf{u}^n)_{n\in\mathbb{N}}$ are bounded,*

(iii) *there exists a word $\mathbf{v}$ such that the sizes of the connected components of $(\mathbf{v}\mathbf{u}^n)_{n\in\mathbb{N}}$ are bounded,*

(iv) *for any word $\mathbf{v}$, the sizes of the connected components of $(\mathbf{v}\mathbf{u}^n)_{n\in\mathbb{N}}$ are bounded.*

4 The Labeled Orbit Tree and the Order Problem

Most of the notions of this section have been introduced in [13]. We refer the reader to this reference for the proofs of the results in this section.

We build a tree capturing the links between the connected components of consecutive powers of a reversible Mealy automaton. See an example in Figure 1. As recalled at the end of this section, the existence of elements of infinite order in the semigroup generated by an invertible-reversible automaton is closely related to some path property of this tree.

Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a reversible Mealy automaton. Consider the tree with vertices the connected components of the powers of $\mathcal{A}$, and the incidence relation built by adding an element of Q: for any $n \geq 0$, the connected component of $\mathbf{u} \in Q^n$ is linked to the connected component(s) of $\mathbf{u}x$, for any $x \in Q$. This tree is called the *orbit tree* of $\mathfrak{d}(\mathcal{A})$ [5, 11]. It can be seen as the quotient of the tree Q^* under the action of the group $\langle \mathfrak{d}(\mathcal{A}) \rangle$.

We label any edge $\mathcal{C} \to \mathcal{D}$ of the orbit tree by the ratio $\frac{\#\mathcal{D}}{\#\mathcal{C}}$, which is always an integer (less than or equal to $\#\mathcal{A}$) by the reversibility of $\mathcal{A}$. We call this labeled tree the *labeled orbit tree* of $\mathfrak{d}(\mathcal{A})$ [13]. We denote by $\mathfrak{t}(\mathcal{A})$ the labeled orbit tree of $\mathfrak{d}(\mathcal{A})$. Note that for each vertex of $\mathfrak{t}(\mathcal{A})$, the sum of the labels of all edges going down from this vertex always equals to $\#Q$, the size of $\mathcal{A}$.

Each vertex of $\mathfrak{t}(\mathcal{A})$ is labeled by a connected automaton with stateset in Q^n, where n is the level of this vertex in the tree. By a minor abuse, we can consider that each vertex is labeled by a finite language in Q^n, or even by a word in Q^n.

Let $\mathbf{u}$ be a (possibly infinite) word over Q. The *path of* $\mathbf{u}$ in the orbit tree $\mathfrak{t}(\mathcal{A})$ is the unique initial path going from the root through the connected components of the prefixes of $\mathbf{u}$; $\mathbf{u}$ can be called *a representative* of this initial path (we can say equivalently that this path is *represented* by $\mathbf{u}$ or that the word $\mathbf{u}$ *represents* the path).

▶ **Definition 3.** Let $\mathcal{A}$ be a reversible Mealy automaton and $\mathfrak{s}$ be a subtree of $\mathfrak{t}(\mathcal{A})$. An $\mathfrak{s}$-*word* is a word in $Q^* \cup Q^\infty$ representing an initial path of $\mathfrak{s}$. A *cyclic* $\mathfrak{s}$-*word* is a word in Q^* whose all powers are $\mathfrak{s}$-words (equivalently, it is an $\mathfrak{s}$-word viewed as a cyclic word).

The structure of an orbit tree is not arbitrary and it is possible to identify some similarities inside this tree.

▶ **Definition 4.** Let e and f be two edges in the orbit tree $\mathfrak{t}(\mathcal{A})$. We say that e *is liftable to* f if each word of $\bot(e)$ admits some word of $\bot(f)$ as a suffix.

Consider $\mathbf{u}$ in $\top(e)$ and its suffix $\mathbf{v}$ in $\top(f)$: any state $x \in Q$ such that $\mathbf{u}x \in \bot(e)$ satisfies $\mathbf{v}x \in \bot(f)$. Informally, "e liftable to f" means that what can happen after $\top(e)$ by following e can also happen after $\top(f)$ by following f. This condition is equivalent to a weaker one:

▶ **Lemma 5.** *Let $\mathcal{A}$ be a reversible Mealy automaton, and let e and f be two edges in the orbit tree $\mathfrak{t}(\mathcal{A})$. If there exists a word of $\bot(e)$ which admits a word of $\bot(f)$ as suffix, then e is liftable to f.*

Obviously if e is liftable to f, then f is closer to the root of the orbit tree. The fact that an edge is liftable to another one reflects a deeper relation stated below. The following lemma is one of the key observations.

▶ **Lemma 6.** *Let e and f be two edges in the orbit tree $\mathfrak{t}(\mathcal{A})$. If e is liftable to f, then the label of e is less than or equal to the label of f.*

The notions of children of an edge and of being liftable to it are not linked, but it is interesting to consider their intersection.

▶ **Definition 7.** Let e and f be two edges in an orbit tree: e is a *legitimate child* of f if f is its parent and e is liftable to f.

The notion of liftability can be generalized to paths:

▶ **Definition 8.** Let $\mathbf{e} = (e_i)_{i\in I}$ and $\mathbf{f} = (f_i)_{i\in I}$ be two paths of the same (possibly infinite) length in the orbit tree $\mathfrak{t}(\mathcal{A})$. The path $\mathbf{e}$ *is liftable to* the path $\mathbf{f}$ if, for any $i \in I$, the edge e_i is liftable to the edge f_i.

▶ **Definition 9.** Let $\mathcal{A}$ be a bireversible Mealy automaton and $\mathfrak{s}$ be a (possibly infinite) path or subtree of $\mathfrak{t}(\mathcal{A})$. For $k > 0$, $\mathfrak{s}$ is k-*self-liftable* whenever any path in $\mathfrak{s}$ starting at level $i + k$ is liftable to a path in $\mathfrak{s}$ starting at level i, for any $i \geq 0$. A path or a subtree is *self-liftable* if it is k-self-liftable for some $k > 0$.

The path represented by x^ω, for some state x, is an example of an infinite initial 1-self-liftable path.

▶ **Lemma 10.** *Let* $\mathbf{e}$ *be a non-empty finite initial* 1*-self-liftable path of some orbit tree* $\mathfrak{t}(\mathcal{A})$*, with last edge* e*. The edge* e *has at least one legitimate child. The sum of the labels of the legitimate children of* e *is equal to the label of* e*.*

Proof. Denote by k the label of e. Let $\mathbf{u}$ be some state in $\top(e)$ and x some state of $\mathcal{A}$ such that $\mathbf{u}x$ is a state of $\bot(e)$ —this is possible by the definition of an orbit tree. We decompose $\mathbf{u}$ in its first letter and some suffix: $\mathbf{u} = z\mathbf{v}$. As $\mathbf{e}$ is a 1-self-liftable path and $z\mathbf{v}x$ is a state in $\bot(\mathbf{e}) = \bot(e)$, we know that $\mathbf{v}x$ is a state in $\top(e)$. Hence by the construction of a label orbit tree, there exist exactly k states $(y_i)_{1\leq i\leq k}$ such that $(\mathbf{v}xy_i)_i$ are states of $\bot(e)$. So the connected components of the $(z\mathbf{v}xy_i)_i$ label legitimate children of e. Clearly e cannot have another legitimate child. ◀

We recall here a characterization of the existence of elements of infinite order in the semigroup generated by a reversible Mealy automaton $\mathcal{A}$ in terms of path properties of the associated orbit tree $\mathfrak{t}(\mathcal{A})$ [13].

▶ **Definition 11.** Any branch labeled by a word not suffixed by 1^ω is called *active.*

▶ **Theorem 12.** *[13] The semigroup generated by an invertible-reversible automaton* $\mathcal{A}$ *admits elements of infinite order if and only if the orbit tree* $\mathfrak{t}(\mathcal{A})$ *admits an active self-liftable branch.*

5 Jungle Trees

Our main result being known for non bireversible automata [7], we focus on the bireversible case. All the tools introduced in this section are new. They are used to get rid of the particularity of the stateset of size 3 in [13].

Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a connected bireversible Mealy automaton with no active self-liftable branch. From Theorem 12, all the elements of the semigroup $\langle\mathcal{A}\rangle_+$ have finite order. In this section we introduce the tools to prove that such an automaton of prime size generates a finite group (Theorem 32).

5.1 Jungle trees and stems

We focus on some particular subtrees of $\mathfrak{t}(\mathcal{A})$:

▶ **Definition 13.** Let $\mathbf{e}$ be a finite initial 1-self-liftable path such that:
- $\bot(\mathbf{e})$ has at least two legitimate children;
- every legitimate child of $\bot(\mathbf{e})$ has label 1.

The *jungle tree* $\mathfrak{j}(\mathbf{e})$ of $\mathbf{e}$ is the subtree of $\mathfrak{t}(\mathcal{A})$ build as follows:
- it contains the path $\mathbf{e}$ — its *trunk*;
- it contains the regular tree rooted by $\bot(\mathbf{e})$, and formed by all the edges which are descendant of $\bot(\mathbf{e})$ and liftable to the lowest (*i.e.* the last) edge of $\mathbf{e}$.

The *arity* of this jungle tree is the number of legitimate children of $\bot(\mathbf{e})$. Since every legitimate child has label 1, it is also the label of the last edge of $\mathbf{e}$.

Words in $\bot(\mathbf{e})$ are called *stems.* They have all the same length which is the length of the trunk of $\mathfrak{j}(\mathbf{e})$.

A tree is a *jungle tree* if it is the jungle tree of some finite initial 1-self-liftable path. An example of such a tree is depicted in Figure 3.

Graphically, a jungle tree starts with a linear part whose labels decrease (its trunk) and eventually ends as a regular tree with all labels 1. Any jungle tree is 1-self-liftable.

Note that: (i) there exists at least one jungle tree, from Lemma 10 and the hypothesis that $\mathcal{A}$ has no active self-liftable branch; (ii) there are finitely many jungle trees.

From now on, $\mathfrak{j}$ **denotes a jungle tree of** $\mathcal{A}$**, whose trunk has length** n**.**

As shown below, any cyclic $\mathfrak{j}$-words has finite order.

▶ **Remark 14.** *If* $\mathbf{uv}$ *is a* $\mathfrak{j}$*-word, with* $|\mathbf{v}| \geq n$*, what can follow* $\mathbf{uv}$ *in* $\mathfrak{j}$ *is independent from* $\mathbf{u}$*. In particular, if* $\mathbf{vw}$ *is also a* $\mathfrak{j}$*-word, then so is* $\mathbf{uvw}$*.*

The existence of cyclic $\mathfrak{j}$-words is ensured by the simple fact that any $\mathfrak{j}$-word of length $n \times (1 + \#Q^n)$ admits at least two identical factors of length n, and hence has a cyclic $\mathfrak{j}$-word as a factor by Remark 14.

▶ **Proposition 15.** *Every cyclic* $\mathfrak{j}$*-word induces an action of finite order, bounded by a uniform constant depending on* $\mathfrak{j}$*.*

Proof. Let $\mathbf{u}$ be a cyclic $\mathfrak{j}$-word, then, for any integer $k > n$, $\mathbf{u}^k$ is a $\mathfrak{j}$-word. By the definition of a jungle tree, the label of the path of $\mathbf{u}^\omega$ is ultimately 1 and, by Proposition 1, the action induced by $\mathbf{u}$ has finite order, bounded by a constant which depends on the connected component at the end of the trunk of $\mathfrak{j}$. ◀

Because of the self-liftability of $\mathfrak{j}$, any factor of a $\mathfrak{j}$-word is itself a $\mathfrak{j}$-word. Hence any factor of length n of a $\mathfrak{j}$-word is a stem. And by the construction of $\mathfrak{j}$, the end of its trunk has only one vertex whose label is hence a connected component, and all the stems are states of that same connected component.

▶ **Definition 16.** Let $\mathfrak{j}$ be a jungle tree of trunk of length n. A *liana covering up* $\mathfrak{j}$ is a language of $\mathfrak{j}$-words, of the form $\mathbf{w}L_\mathbf{w}$, where $\mathbf{w} \in Q^n$ is a stem, and $L_\mathbf{w} \subseteq Q^* \cup Q^\infty$ is a prefix-preserving language which, seen as a tree, is regular of the same arity than $\mathfrak{j}$.

Each vertex of $\mathfrak{j}$ has exactly one representative in $\mathbf{w}L_\mathbf{w}$. For each stem $\mathbf{w}$ there is exactly one suitable $L_\mathbf{w}$.

▶ **Remark 17.** *Let* $\mathbf{w}L_\mathbf{w}$ *be a liana covering up a jungle tree* $\mathfrak{j}$ *and* $\mathbf{uv}$ *be a finite* $\mathfrak{j}$*-word such that* $|\mathbf{v}| = n$*: if* $L_\mathbf{v}$ *is the greatest language such that* $\mathbf{uv}L_\mathbf{v} \subseteq \mathbf{w}L_\mathbf{w}$*, then* $\mathbf{v}L_\mathbf{v}$ *is also a liana covering up* $\mathfrak{j}$*.*

In what follows, we try to better understand the stucture of jungle trees and lianas. Let $S = \mathbf{s}L_\mathbf{s}$ be a liana covering up $\mathfrak{j}$ ($\mathbf{s} \in Q^n$). Our goal is to prove the following result:

▶ **Theorem 18.** *Let* $\mathbf{u}$ *be a factor in* S*. Then* $\mathbf{u}$ *has the following property:*

If $\mathbf{uv} \in Q^*$ *is a factor in* S*, then* $\mathbf{u}$ *exists further in* S*.* (**Ubiquity**)

More formally: if $\mathbf{tuv} \in S$*, there exists* $\mathbf{w} \in Q^*$ *such that* $\mathbf{tuvwu} \in S$*.*

The graphical sense of this theorem is that if you are walking on a $\mathfrak{j}$-word and you have already seen some factor, you can find eventually this same factor.

Proof. First, remember that if $\mathbf{u}$ is a stem (*i.e.* $\mathbf{u}$ is a factor in S of length n), what can follow $\mathbf{u}$ (in S) does not depend either of the choice of the liana (as long as you are in a liana covering up the same jungle tree), or of the location of $\mathbf{u}$ in this liana. Hence it is sufficient to prove Theorem 18 for $|\mathbf{u}| = n$.

We start by proving that there is at least one stem $\mathbf{u}_0$ with Property (**Ubiquity**). To obtain this word, we travel along $S = \mathbf{s}L_\mathbf{s}$ in the following way, starting with $\mathbf{u}_0 = \mathbf{s}$:

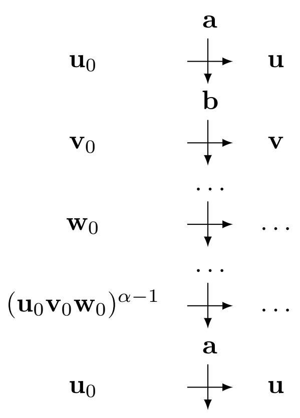


Figure 2 Extension of Property (**Ubiquity**) from $\mathbf{u}_0$ to $\mathbf{u}$.

- if $\mathbf{u}_0$ answers to the question, our journey is over;
- otherwise, at the end of $\mathbf{u}_0$ we follow some finite path such that $\mathbf{u}_0$ does not exist anymore after this path; then we replace $\mathbf{u}_0$ by the next word of length n in S, and back to the previous step.

Since S is infinite but has a finite arity and a finite number of factors of length n, the previous algorithm ends returning a stem $\mathbf{u}_0$ satisfying Property (**Ubiquity**). By Remark 17, the jungle tree $\mathfrak{j}$ is covered up by a liana of the form $\mathbf{u}_0 L_{\mathbf{u}_0}$.

The extension of Property (**Ubiquity**) to other words is illustrated by Figure 2. Let $\mathbf{uv}$ be a factor in S, with $|\mathbf{u}| = n$. In particular $\mathbf{u}$ is a stem, hence $\mathbf{u}_0$ and $\mathbf{u}$ are states of the same connected component, and there exists a path in this component from $\mathbf{u}_0$ to $\mathbf{u}$, say by the action of some $\mathbf{a} \in \Sigma^*$. The automaton being reversible, $\mathbf{v}$ is the image of some $\mathbf{v}_0 \in Q^{|\mathbf{v}|}$ by $\mathbf{b} = \delta_{\mathbf{u}_0}(\mathbf{a})$ (see Figure 2).

Now, we know that on the left part of Figure 2 we can find eventually $\mathbf{u}_0$, after some $\mathbf{w}_0$ (because $\mathbf{u}_0$ has Property (**Ubiquity**)). And by the invertibility of the automaton, there exists some power α of $\mathbf{u}_0\mathbf{v}_0\mathbf{w}_0$ which stabilizes $\mathbf{a}$ (see Figure 2).

Hence $\mathbf{u}$ can be seen again eventually. Furthermore, the vertical word on the right of Figure 2 is a $\mathfrak{j}$-word, as it is in the same connected component than the vertical $\mathfrak{j}$-word on the left of this same figure, and so it is a factor of S because, by hypothesis, its prefix of length n is a factor of S. Hence $\mathbf{u}$ has Property (**Ubiquity**). ◀

▶ **Remark 19.** *Note that, from Theorem 18, if $\mathbf{u}$, $\mathbf{v}$ are two stems such that $\mathbf{v}$ is a factor of some word in $\mathbf{u}L_{\mathbf{u}}$, then $\mathbf{u}$ is a factor of some word in $\mathbf{v}L_{\mathbf{v}}$.*

5.2 An equivalence on stems

Remember that $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ is a connected bireversible Mealy automaton such that $\mathfrak{t}(\mathcal{A})$ has no active self-liftable branch (and as a consequence all the elements of the semigroup $\langle \mathcal{A} \rangle_+$ have finite order). Let $\mathfrak{j}$ be a jungle tree of $\mathfrak{t}(\mathcal{A})$ with trunk of length n. All the stems considered from now on are stems of $\mathfrak{j}$.

In this subsection we prove several properties for the stems of the jungle tree $\mathfrak{j}$. Stems are used then in Section 6 to build a $\mathfrak{j}$-word inducing the same action than some given word.

Let us first introduce an equivalence relation on the set of stems.

▶ **Definition 20.** Let $\mathbf{u}$, $\mathbf{v}$ be two stems. We say that $\mathbf{u}$ is *equivalent* to $\mathbf{v}$, denoted by $\mathbf{u} \sim \mathbf{v}$, whenever there exists $\mathbf{s} \in Q^*$ such that $\mathbf{usv}$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{us}}$ acts like the identity on Σ^*.

▶ **Lemma 21.** *The relation $\sim$ is an equivalence relation on stems.*

Proof. Let $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ be three stems.

transitivity Suppose that $\mathbf{u} \sim \mathbf{v}$ and $\mathbf{v} \sim \mathbf{w}$: there exists $\mathbf{s}, \mathbf{t} \in Q^*$ such that $\mathbf{usv}$ and $\mathbf{vtw}$ are $\mathfrak{j}$-words, and $\rho_{\mathbf{us}}$ and $\rho_{\mathbf{vt}}$ act like the identity. As $\mathbf{v}$ is a stem, we obtain by Remark 14 that $\mathbf{usvtw}$ is a $\mathfrak{j}$-word, and $\rho_{\mathbf{usvt}}$ acts like the identity, so $\mathbf{u} \sim \mathbf{w}$.

reflexivity From Theorem 18, there exists $\mathbf{s} \in Q^*$ such that $\mathbf{usu}$ is a $\mathfrak{j}$-word (in fact from Theorem 18 one can even chose the beginning of $\mathbf{s}$, as long as we keep a $\mathfrak{j}$-word). As $\mathbf{u}$ is a stem, $\mathbf{usus}$ is also a $\mathfrak{j}$-word, and so are all the powers of $\mathbf{us}$. Now, by hypothesis and Theorem 12, $\mathbf{us}$ is of finite order, say α: $\mathbf{u}(\mathbf{su})^{\alpha-1}\mathbf{su}$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{u}(\mathbf{su})^{\alpha-1}\mathbf{s}} = \rho_{(\mathbf{us})^{\alpha}}$ acts like the identity.

symmetry Suppose that $\mathbf{u} \sim \mathbf{v}$: there exists $\mathbf{s} \in Q^*$ such that $\mathbf{usv}$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{us}}$ acts like the identity. From the reflexivity proof, there exists $\mathbf{t} \in Q^*$ such that $\mathbf{usvtu}$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{usvt}}$ acts like the identity. Hence $\mathbf{vtu}$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{vt}}$ acts like the identity, which proves the symmetry.

◀

Note that from reflexivity of $\sim$ and Theorem 18, if $\mathbf{u}$ and $\mathbf{v}$ are equivalent stems and $\mathbf{uw}$ is a $\mathfrak{j}$-word for some $\mathbf{w} \in Q^*$, then there exists $\mathbf{s} \in Q^*$ such that $\mathbf{uwsv}$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{uws}}$ acts like the identity. So not only $\mathbf{v}$ can be reached from $\mathbf{u}$ by producing first the identity, but even if you walk in $\mathfrak{j}$ after reading $\mathbf{u}$, you can still reach $\mathbf{v}$ and produce first the identity.

We can now consider the equivalence classes induced by $\sim$. The aim of this subsection is to prove that if $\mathcal{A}$ has a prime size, then for a given state q there is in each $\sim$-class a stem with prefix q (Theorem 30).

▶ **Proposition 22.** *All the equivalence classes of $\sim$ have the same size.*

Proof. Let $\mathbf{u}_0$ and $\mathbf{v}_0$ be two stems of $\mathfrak{j}$: they are states of the same connected component and so there exists $\mathbf{a} \in \Sigma^*$ such that $\delta_{\mathbf{a}}(\mathbf{u}_0) = \mathbf{v}_0$. Denote by $\{\mathbf{u}_0, \dots, \mathbf{u}_k\}$ the $\sim$-class of $\mathbf{u}_0$: for any i, $1 \leq i \leq k$, there exists $\mathbf{s}_i \in Q^*$ such that $\mathbf{u}_0\mathbf{s}_i\mathbf{u}_i$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{u}_0\mathbf{s}_i}$ acts like the identity. Define the words $\mathbf{v}_i \in Q^{|\mathbf{u}_i|}$ and $\mathbf{t}_i \in Q^{|\mathbf{s}_i|}$ in the following way: $\delta_{\mathbf{a}}(\mathbf{u}_0\mathbf{s}_i\mathbf{u}_i) = \mathbf{v}_0\mathbf{t}_i\mathbf{v}_i$. Note that $\mathbf{v}_0\mathbf{t}_i\mathbf{v}_i$ is also a $\mathfrak{j}$-word: any factor of size n of $\mathbf{v}_0\mathbf{t}_i\mathbf{v}_i$ is the image of a stem (the corresponding factor in $\mathbf{u}_0\mathbf{s}_i\mathbf{u}_i$) and therefore belongs to the connected component of $\mathbf{u}_0$ and $\mathbf{v}_0$, hence every prefix of $\mathbf{v}_0\mathbf{t}_i\mathbf{v}_i$ is on a 1-self-liftable path. Now $\rho_{\mathbf{v}_0\mathbf{t}_i}$ acts like the identity by the reversibility of $\mathcal{A}$, so $\mathbf{v}_i$ is $\sim$-equivalent to $\mathbf{v}_0$. Furthermore, as $\rho_{\mathbf{u}_0\mathbf{s}_i}$ acts like the identity, we know that $\mathbf{v}_i = \delta_{\mathbf{a}}(\mathbf{u}_i)$, and all the $\mathbf{v}_i$ are different. ◀

5.3 Combinatorial properties of stems

We now state several combinatorial properties for stems. Let $k_1, k_2, \dots, k_n$ be the labels, from root to $\perp(\mathbf{e})$, of the jungle tree $\mathfrak{j} = \mathfrak{j}(\mathbf{e})$. Recall that, since $\mathcal{A}$ is connected, $k_1 = p$ and by construction of the jungle tree $k_n \geq 2$. For instance in Figure 3, $n = 4$, $k_1 = k_2 = 3$, and $k_3 = k_4 = 2$.

First if we consider no restriction then we can directly count stems by looking to the labels of the trunk:

▶ **Lemma 23.** *The number of stem with a given prefix depends only on length i of the prefix and is $k_{i+1}k_{i+2}\dots k_n$.*

We are now interested in two somehow dual questions. Fix a $\mathfrak{j}$-word $\mathbf{u}$ of length less than n: (i) if $\mathbf{u}$ is the prefix of a stem in some $\sim$-class γ, in how many way can $\mathbf{u}$ be completed in γ (Proposition 24)? (ii) in how many $\sim$-classes is $\mathbf{u}$ the prefix of a stem (Corollary 29)?

▶ **Proposition 24.** *Fix some $\mathfrak{j}$-word $\mathbf{u}$ of length less than n, a $\sim$-class γ of stems including an element with prefix $\mathbf{u}$, and some integer k such that $|\mathbf{u}| + k \leq n$. The number of $\mathbf{v} \in Q^k$ such that $\mathbf{uv}$ is a prefix of a stem of γ depends only on $|\mathbf{u}|$ and k.*

Proof. By the same argument than in the proof of Proposition 22. ◀

Let $\mathbf{u} \in Q^*$ be a prefix of a stem in some $\sim$-class γ. Denote by $\mathcal{S}_{\text{eq}}(|\mathbf{u}|+1)$ the cardinality of the set $\{q \in Q \mid \mathbf{u}q \text{ is a prefix of some stem in } \gamma\}$ (from Proposition 24 it depends only of $|\mathbf{u}|$ and so it is well-defined).

In order to obtain a minimal bound on the size of a $\sim$-class, we introduce another equivalence relation between stems which is finer than $\sim$, as proved in Lemma 26:

▶ **Definition 25.** Let $\mathbf{u}, \mathbf{v}$ be two stems. Define the relation $\mathbf{u} \wedge_0 \mathbf{v}$ whenever there exists a stem $\mathbf{s}$ such that both $\mathbf{su}$ and $\mathbf{sv}$ are $\mathfrak{j}$-words. The equivalence relation $\wedge$ is defined as the transitive closure of $\wedge_0$.

Note that by the construction of the jungle tree, a $\wedge_0$-class contains k^n elements, where k stands for the arity of this jungle tree.

▶ **Lemma 26.** *The relation $\wedge$ is finer than the relation $\sim$: $\mathbf{u} \wedge \mathbf{v} \Rightarrow \mathbf{u} \sim \mathbf{v}$.*

Proof. By transitivity it is enough to prove that: $\mathbf{u} \wedge_0 \mathbf{v} \Rightarrow \mathbf{u} \sim \mathbf{v}$. Let $\mathbf{u}$ and $\mathbf{v}$ be two stems such that $\mathbf{u} \wedge_0 \mathbf{v}$: there exists a stem $\mathbf{s}$ such that $\mathbf{su}$ and $\mathbf{sv}$ are $\mathfrak{j}$-words. From Theorem 18, there exists a word $\mathbf{w} \in Q^*$ such that $\mathbf{uws}$ is a $\mathfrak{j}$-word. As $\mathbf{u}$ and $\mathbf{s}$ are stems, and $\mathbf{su}$ is a $\mathfrak{j}$-word, $(\mathbf{uws})^2$ is a $\mathfrak{j}$-word by Remark 14, and so are all the powers of $\mathbf{uws}$. Now, by hypothesis and Theorem 12, the word $\mathbf{uws}$ has finite order, say α: $(\mathbf{uws})^\alpha\mathbf{v}$ is a $\mathfrak{j}$-word and $\rho_{(\mathbf{uws})^\alpha}$ acts like the identity. ◀

▶ **Corollary 27.** *For any i, $\mathcal{S}_{eq}(i) \geq 2$.*

Proof. For a stem $\mathbf{u}$, the set of words in $\wedge_0$-relation with $\mathbf{u}$, seen as a tree, has the same arity than $\mathfrak{j}$; so, by Lemma 26, for any i, $\mathcal{S}_{\text{eq}}(i)$ is greater than or equal to the arity of $\mathfrak{j}$. ◀

▶ **Proposition 28.** *Fix a $\mathfrak{j}$-word $\mathbf{u}$ of length less than n. The number of stems prefixed by $\mathbf{u}$ in a $\sim$-class is either 0 or depends only on $|\mathbf{u}|$.*

Proof. By the same argument than in the proof of Proposition 22. ◀

From Propositions 24 and 28 we obtain:

▶ **Corollary 29.** *Fix a $\mathfrak{j}$-word $\mathbf{u}$ of length less than n. The number of $\sim$-classes where $\mathbf{u}$ is the prefix of some stem depends only on $|\mathbf{u}|$.*

Denote by $\mathcal{P}_{\text{eq}}(|\mathbf{u}|+1)$ the number of $\sim$-classes containing a stem prefixed by $\mathbf{u}$ (it is correctly define by Corollary 29).

We can now prove the main result of this section:

▶ **Theorem 30.** *Let $\mathcal{A}$ be a connected bireversible Mealy automaton of prime size and without any active self-liftable branch. The set of states which appear as first letter of a stem in a fixed $\sim$-class is the whole stateset.*

Proof. Suppose $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ has prime size p, and let $\mathfrak{j}$ be a jungle tree of $\mathfrak{t}(\mathcal{A})$ whose trunk $\mathbf{e}$ has length n. We denote by $k_1,\ldots,k_n$ the label of the edges of $\mathbf{e}$ (from top to bottom). By the connectivity of $\mathcal{A}$, $k_1 = p$.

Let γ be a $\sim$-class of stems for $\mathfrak{j}$ and $\mathbf{u} \in Q^*$ of length $i \leq n$ be the prefix of some stem in γ. Consider all the stems in γ with prefix $\mathbf{u}$.

From Lemma 23, the number of stems of $\mathfrak{j}$ prefixed by $\mathbf{u}$ is $k_{i+1} \times k_{i+2} \times \ldots \times k_n$. On the other hand, it is also the number of stems with prefix $\mathbf{u}$ in γ, *i.e.* $\mathcal{S}_{\text{eq}}(i+1) \times \cdots \times \mathcal{S}_{\text{eq}}(n)$, times the number of $\sim$-classes which has a stem prefixed by $\mathbf{u}$, *i.e.* $\mathcal{P}_{\text{eq}}(i+1) \times \cdots \times \mathcal{P}_{\text{eq}}(n)$. Hence

$$k_{i+1} \times k_{i+2} \times \ldots \times k_n = \mathcal{S}_{\text{eq}}(i+1) \times \mathcal{P}_{\text{eq}}(i+1) \times \mathcal{S}_{\text{eq}}(i+2) \times \mathcal{P}_{\text{eq}}(i+2) \times \ldots \times \mathcal{S}_{\text{eq}}(n) \times \mathcal{P}_{\text{eq}}(n) .$$

It is straightforward that $k_n = \mathcal{P}_{\text{eq}}(n) \times \mathcal{S}_{\text{eq}}(n)$ and by induction $\mathcal{P}_{\text{eq}}(\ell) \times \mathcal{S}_{\text{eq}}(\ell) = k_\ell$ for all ℓ. In particular for $\ell = 1$, we get that $\mathcal{S}_{\text{eq}}(1)$ devides k_1. Since $k_1 = p$ and, from Corollary 27, $\mathcal{S}_{\text{eq}}(1) \geq 2$, we obtain then $\mathcal{S}_{\text{eq}}(1) = p$. ◀

▶ **Corollary 31.** *Let $\mathcal{A} = (Q, \Sigma, \delta, \rho)$ be a connected bireversible Mealy automaton of prime size, with no active self-liftable branch. Let $\mathfrak{j}$ be a jungle tree of $\mathfrak{t}(\mathcal{A})$ and $\mathbf{u}$ some $\mathfrak{j}$-word. Then for any state $x \in Q$, there exists $\mathbf{w} \in Q^*$ such that $\mathbf{uw}x$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{w}}$ acts like the identity of Σ^*.*

Proof. Let $\mathbf{s}$ be a stem such that $\mathbf{us}$ is a $\mathfrak{j}$-word: there exists a stem $\mathbf{x}$ with first letter x in the $\sim$-class of $\mathbf{s}$, from Theorem 30, *i.e.* there exists $\mathbf{v} \in Q^*$ such that $\mathbf{svx}$ is a $\mathfrak{j}$-word and $\rho_{\mathbf{sv}}$ acts like the identity of Σ^*. Conclusion comes from Remark 14. ◀

Note that in the previous corollary, the word $\mathbf{u}$ can be empty.

6 Proof of the main theorem

We now have all elements to prove our main result.

▶ **Theorem 32.** *A connected invertible-reversible Mealy automaton of prime size cannot generate an infinite Burnside group.*

Proof. Let $\mathcal{A}$ be a connected invertible-reversible Mealy automaton of prime size. If $\mathcal{A}$ is not bireversible we can apply [1, 7] and we get that, on one hand, $\langle \mathcal{A} \rangle$ is necessarily infinite, but on the other hand, it cannot be Burnside. If $\mathcal{A}$ is bireversible and $\mathfrak{t}(\mathcal{A})$ has an active self-liftable branch, then $\langle \mathcal{A} \rangle$ has an element of infinite order by Theorem 12.

Therefore we can assume that $\mathcal{A}$ is bireversible and $\mathfrak{t}(\mathcal{A})$ has no active self-liftable branch. Let us show that $\langle \mathcal{A} \rangle$ is finite. Let $\mathfrak{j}$ be some jungle tree of $\mathfrak{t}(\mathcal{A})$. As in [13] we prove that for any word $\mathbf{u} \in Q^*$, $\rho_{\mathbf{u}}$ has some uniformly bounded power which acts like some cyclic $\mathfrak{j}$-word.

Let $\mathbf{u} \in Q^*$. We prove by induction that any prefix $\mathbf{u}$ induces the same action than some $\mathfrak{j}$-word. It is obviously true for the empty prefix. Fix some $k < |\mathbf{u}|$ and suppose that the prefix $\mathbf{v}$ of length k of $\mathbf{u}$ induces the same action than some $\mathfrak{j}$-word $\mathbf{s}$. Let $x \in Q$ be the $(k+1)$-th letter of $\mathbf{u}$. By Corollary 31, there exists a $\mathfrak{j}$-word $\mathbf{w}$ inducing the identity, such that $\mathbf{sw}x$ is a $\mathfrak{j}$-word. But $\mathbf{v}x$ and $\mathbf{sw}x$ induce the same action ; the result follows. Hence we obtain a $\mathfrak{j}$-word $\mathbf{u}^{(1)}$ inducing the same action than $\mathbf{u}$.

By the very same process, we can construct, for any $i \in \mathbb{N}$, a j-word $\mathbf{u}^{(i)}$ inducing the same action than $\mathbf{u}$, such that $\mathbf{u}^{(1)}\mathbf{u}^{(2)}\ldots\mathbf{u}^{(i)}$ is a j-word. Since the set Q^n is finite there exist $i < j$, $j - i \leq |Q|^n$, such that $\mathbf{u}^{(i)}$ and $\mathbf{u}^{(j)}$ have the same prefix of length n. Take $\mathbf{v} = \mathbf{u}^{(i)}\mathbf{u}^{(i+1)}\ldots\mathbf{u}^{(j-1)}$: $\mathbf{v}$ is a cyclic j-word and induces the same action than $\mathbf{u}^{j-i}$. By Proposition 15, the order of $\rho_{\mathbf{v}}$ is bounded by a constant depending only on j, hence so is the order of $\rho_{\mathbf{u}}$ (with a different constant, but still depending only on j). Consequently, every element of $\langle\mathcal{A}\rangle$ has a finite order, uniformly bounded by a constant, whence, as $\langle\mathcal{A}\rangle$ is residually finite, by Zelmanov's theorem [14, 15], $\langle\mathcal{A}\rangle$ is finite, which concludes the proof. ◀

The tools and techniques we have developed here enabled to bridge the gap between 3 and the set of all prime numbers. The next step is the extension of our result to any connected automaton. However, experiments suggest that there are strong similarities between the non prime case and the non connected case, bringing the hope to solve entirely the question of the (im)possible generation of an infinite Burnside group by a reversible Mealy automaton. Note that the primality of the stateset is not used here before Theorem 30. It is likely that the extension of Theorem 32 to more general statesets will require to choose carefully some k-self-liftable branches, with $k > 1$. In fact, there exist examples of automata for which the set of first letters in a $\sim$-class is not the whole stateset. However the $\sim$-classes seem to still play a crucial role in these examples. So our construction will certainly be a key element for a more general result.

References

1 A. Akhavi, I. Klimann, S. Lombardy, J. Mairesse, and M. Picantin. On the finiteness problem for automaton (semi)groups. *Int. J. Algebr. Comput.*, 22(6):26p., 2012.

2 S.V. Alešin. Finite automata and the Burnside problem for periodic groups. *Mat. Zametki*, 11:319–328, 1972.

3 W. Burnside. On an unsettled question in the theory of discontinuous groups. *Quart. J. Math.*, 33:230–238, 1902.

4 D. D'Angeli and E. Rodaro. A geometric approach to (semi)-groups defined by automata via dual transducers. *Geometriae Dedicata*, 174-1:375–400, 2015.

5 P.W. Gawron, V.V. Nekrashevych, and V.I. Sushchansky. Conjugation in tree automorphism groups. *Internat. J. Algebr. Comput.*, 11-5:529–547, 2001.

6 V.M. Gluškov. Abstract theory of automata. *Uspehi Mat. Nauk*, 16-5:3–62, 1961.

7 Th. Godin, I. Klimann, and M. Picantin. On torsion-free semigroups generated by invertible reversible Mealy automata. In *LATA'15*, volume 8977 of *LNCS*, pages 328–339, 2015.

8 E.S. Golod. On nil-algebras and finitely residual groups. *Izv. Akad. Nauk SSSR. Ser. Mat.*, 28:273–276, 1964.

9 E.S. Golod and I. Shafarevich. On the class field tower. *Izv. Akad. Nauk SSSR Ser. Mat.*, 28:261–272, 1964.

10 R. Grigorchuk. On Burnside's problem on periodic groups. *Funktsional. Anal. i Prilozhen.*, 14-1:53–54, 1980.

11 R. Grigorchuk and D. Savchuk. Ergodic decomposition of group actions on rooted trees. *to appear in Proc. of Steklov Inst. of Math.*, 2015.

12 I. Klimann. Automaton semigroups: The two-state case. *Theor. Comput. Syst. (special issue STACS'13)*, pages 1–17, 2014. `doi:10.1007/s00224-014-9594-0`.

13 I. Klimann, M. Picantin, and D. Savchuk. A connected 3-state reversible Mealy automaton cannot generate an infinite Burnside group. In *DLT' 15*, volume 9168 of *LNCS*, pages 313–325, 2015.

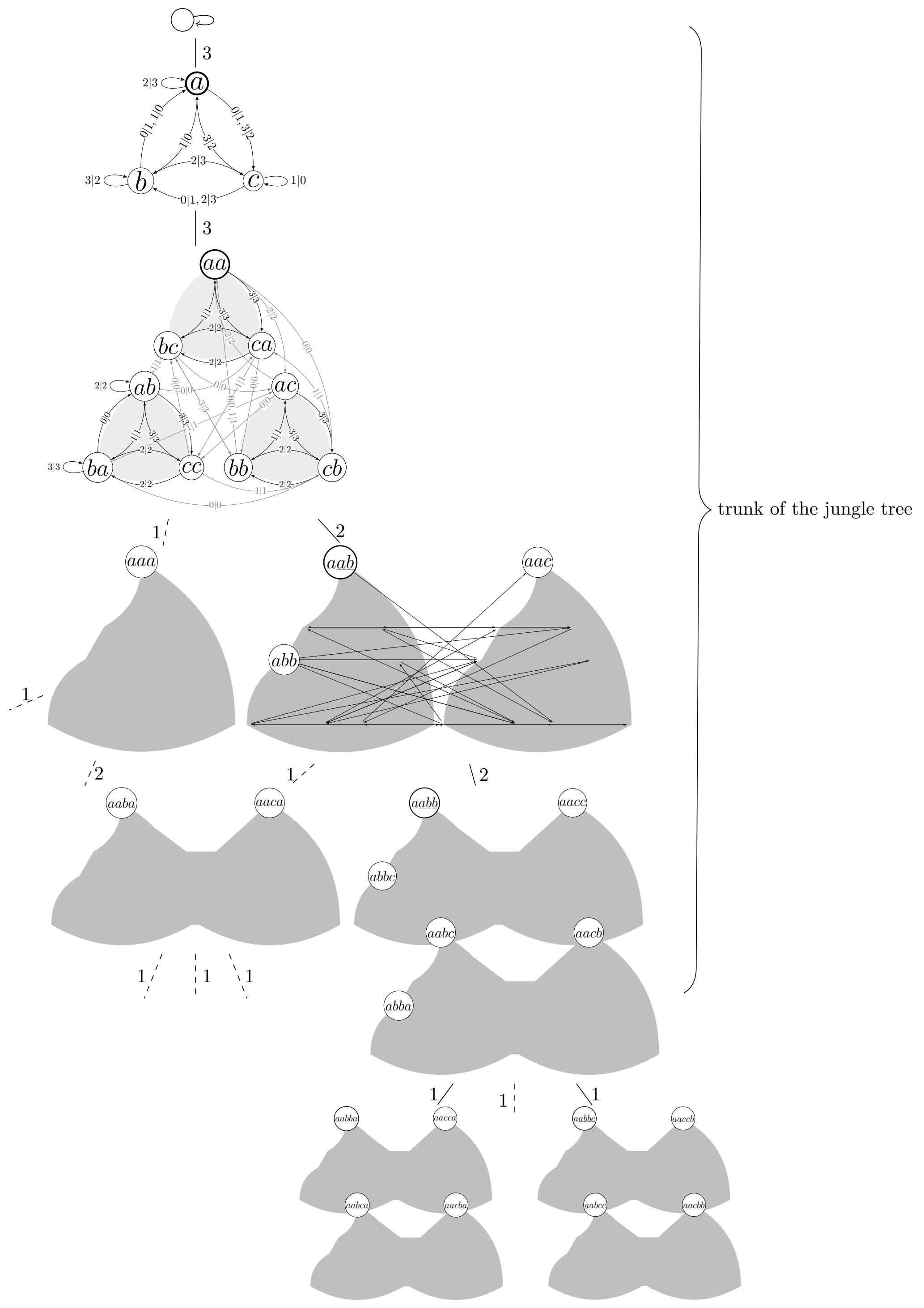


Figure 3 An exemple of the first levels of an orbit tree (all edges) and a jungle tree (plain edges). After the trunk the jungle tree consists in a regular binary tree (plain edges).

14 E.I. Zel′manov. Solution of the restricted Burnside problem for groups of odd exponent. *Izv. AN SSSR Math+*, 54-1:42–59, 221, 1990.

15 E.I. Zel′manov. Solution of the restricted Burnside problem for 2-groups. *Mat. Sb.*, 182-4:568–592, 1991.

Circuit Size Lower Bounds and #SAT Upper Bounds Through a General Framework *

Alexander Golovnev[1,2], Alexander S. Kulikov[2], Alexander V. Smal[3], and Suguru Tamaki[4]

1 **New York University, USA and St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia**
alex.golovnev@gmail.com

2 **St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia**
kulikov@logic.pdmi.ras.ru

3 **St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia**
smal@logic.pdmi.ras.ru

4 **Kyoto University, Japan**
tamak@kuis.kyoto-u.ac.jp

Abstract

Most of the known *lower bounds* for binary Boolean circuits with unrestricted depth are proved by the gate elimination method. The most efficient known *algorithms* for the #SAT problem on binary Boolean circuits use similar case analyses to the ones in gate elimination. Chen and Kabanets recently showed that the known case analyses can also be used to prove *average case circuit lower bounds*, that is, lower bounds on the size of approximations of an explicit function.

In this paper, we provide a general framework for proving worst/average case lower bounds for circuits and upper bounds for #SAT that is built on ideas of Chen and Kabanets. A proof in such a framework goes as follows. One starts by fixing three parameters: a class of circuits, a circuit complexity measure, and a set of allowed substitutions. The main ingredient of a proof goes as follows: by going through a number of cases, one shows that for any circuit from the given class, one can find an allowed substitution such that the given measure of the circuit reduces by a sufficient amount. This case analysis immediately implies an upper bound for #SAT. To obtain worst/average case circuit complexity lower bounds one needs to present an explicit construction of a function that is a disperser/extractor for the class of sources defined by the set of substitutions under consideration.

We show that many known proofs (of circuit size lower bounds and upper bounds for #SAT) fall into this framework. Using this framework, we prove the following new bounds: average case lower bounds of $3.24n$ and $2.59n$ for circuits over U_2 and B_2, respectively (though the lower bound for the basis B_2 is given for a quadratic disperser whose explicit construction is not currently known), and faster than 2^n #SAT-algorithms for circuits over U_2 and B_2 of size at most $3.24n$ and $2.99n$, respectively. Here by B_2 we mean the set of all bivariate Boolean functions, and by U_2 the set of all bivariate Boolean functions except for parity and its complement.

1998 ACM Subject Classification F.1.1 Models of Computation

* The research presented is Section 3.1 is supported in part by MEXT KAKENHI (24106003); JSPS KAKENHI (26330011, 16H02782); the John Mung Advanced Program of Kyoto University. The research presented in Section 3.2 is partially supported by NSF grant 1319051. The research presented in Sections 4–5 is supported by Russian Science Foundation (project 16-11-10123). Part of the work was performed while Suguru Tamaki was at Department of Computer Science and Engineering, University of California, San Diego and the Simons Institute for the Theory of Computing, Berkeley.

Keywords and phrases circuit complexity, lower bounds, exponential time algorithms, satisfiability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.45

1 Introduction

1.1 Background

In this paper, we study binary Boolean circuits with no restriction on the depth. This is a natural model for computing Boolean functions that can be viewed as a simple program where each instruction is just a binary Boolean operation. Shannon [56] showed that for almost all Boolean functions of n variables the size of a smallest circuit (equivalently, the minimal number of instructions) computing this function is $\Omega(2^n/n)$. The proof is based on a counting argument (the number 2^{2^n} of all functions of n variables is larger than the number of circuits of size $o(2^n/n)$) and, for this reason, does not give an explicit function of high circuit complexity. By saying "explicit" one usually means a function from NP. Showing a superpolynomial lower bound for an explicit function would imply P $\neq$ NP. However, despite of many efforts [53, 47, 57, 8, 20, 26, 66, 3], currently we have only small linear lower bounds: $(3 + 1/86)n$ for the full binary basis B_2 consisting of all binary Boolean functions [25] and $5n - o(n)$ for the basis U_2 consisting of all binary Boolean functions except for parity and its complement [38, 32].

Going to larger complexity classes, it is known that the classes MA/1 [50], O_2^p [10], and $\mathrm{P}^{\mathrm{prMA}}$ [11] require circuits of superlinear size and the class MAEXP [9] has superpolynomial circuit complexity. Proving a superlinear lower bound on the circuit complexity of E^{NP} remains to be a major open problem.

Recently, Williams [60, 64] presented the following approach to prove circuit size lower bounds against E^{NP} or NE using SAT-algorithms: a super-polynomially faster than 2^n algorithm for the circuit satisfiability problem of a "reasonable" circuit class $\mathcal{C}$ implies either $\mathrm{E}^{\mathrm{NP}} \not\subseteq \mathcal{C}$ or $\mathrm{NE} \not\subseteq \mathcal{C}$, depending on $\mathcal{C}$ and the running time of the algorithm. The approach has been strengthened and simplified by subsequent work [59, 61, 63, 7, 33], see also excellent surveys [52, 45, 62] on this topic.

Williams' result inspired lots of work on satisfiability algorithms for various circuit classes [30, 63, 15, 2, 1, 43, 16, 58]. In addition to satisfiability algorithms, several papers [51, 29, 4, 54, 14, 12, 17, 49] also obtained average-case lower bounds (also known as correlation bounds, see [35, 36, 28]) by investigating the analysis of algorithms instead of just applying Williams' result that yields worst-case lower bounds. In particular, Chen and Kabanets [13] presented algorithms that count the number of satisfying assignments of circuits over U_2 and B_2 and run in time exponentially faster than 2^n if input instances have at most $2.99n$ and $2.49n$ gates, respectively (improving also the previously best known #SAT-algorithm by Nurk [44]). At the same time, they showed that $2.99n$ sized circuits over U_2 and $2.49n$ sized circuits over B_2 have exponentially small correlations with the parity function and affine extractors having "good" parameters, respectively.

To prove a lower bound of ζn on the circuit size, one usually shows that for any circuit there is a substitution $x_i \leftarrow f$ eliminating at least ζ gates from the circuit. For example, to prove a lower bound of $3n - 3$ on the circuit size over U_2 of the parity function, Schnorr [53] shows how to make a bit-fixing substitution (i.e., $f = c$ for $c \in \{0, 1\}$) eliminating at least 3 gates from any U_2-circuit. Demenkov and Kulikov [20] prove a lower bound of $3n - o(n)$ on the circuit size over B_2 of an affine disperser by showing that for any B_2-circuit there is an affine substitution ($f = \oplus_{j \in J} x_j \oplus c$) eliminating at least three gates from the circuit. Chen and Kabanets proved new average case lower bounds and #SAT upper bounds by analyzing what happens in the complementary branch $x_i \leftarrow f \oplus 1$ of proofs by Schnorr and by Demenkov and Kulikov.

1.2 Our Techniques and Results

The main qualitative contribution of this paper is a general framework for proving circuit worst/average case lower bounds and #SAT upper bounds. This framework is separated into conceptual and technical parts. The conceptual part is a proof that for a given circuit complexity measure and a set of allowed substitutions, for any circuit, there is a substitution that reduces the complexity of the circuit by a sufficient amount. This is usually shown by analyzing the structure of the top of a circuit. The technical part is a set of lemmas that allows us to derive worst/average case circuit size lower bounds and #SAT upper bounds as one-line corollaries from the corresponding conceptual part. The technical part can be used in a black-box way: given a proof that reduces the complexity measure of a circuit (conceptual part), the technical part implies circuit lower bounds and #SAT upper bounds. For example, by plugging in the proofs by Schnorr and by Demenkov and Kulikov, one immediately gets the bounds given by Chen and Kabanets. We also give new proofs that lead to the quantitatively better results. The main quantitative contribution of the paper is the following new bounds which are currently the strongest known bounds:

- average case lower bounds of $3.24n$ and $2.59n$ for circuits over U_2 and B_2 (though the lower bound for the basis B_2 is given for a quadratic disperser whose explicit construction is not currently known), respectively, improving upon the bounds of $2.99n$ and $2.49n$ [13];
- faster than 2^n #SAT-algorithms for circuits over U_2 and B_2 of size at most $3.24n$ and $2.99n$, respectively, improving upon the bounds of $2.99n$ and $2.49n$ [13].

These bounds are obtained by using non-standard circuit complexity measures and sets of substitutions. We also show that obtaining non-linear lower bounds through a weak version of this framework is unlikely as it would violate the Exponential Time Hypothesis [31] that states the following: The satisfiability problem of 3-CNF formulas with n variables cannot be solved in time $2^{o(n)}$. ETH is widely used as a hardness assumption to prove the optimality of many algorithms, see, e.g., [41, 42].

1.3 Framework

We prove circuit lower bounds (both in the worst case and in the average case) and upper bounds for #SAT using the following four step framework.

Initial setting We start by specifying the three main parameters: a class of circuits $\mathcal{C}$, a set $\mathcal{S}$ of allowed substitutions, and a circuit complexity measure μ. A set of allowed substitutions naturally defines a class of "sources". For the circuit lower bounds we consider functions that are non-constant (dispersers) or close to uniform (extractors) on corresponding sets of sources. In this paper we focus on the following four sets of substitutions where each set extends the previous one:

1. Bit fixing substitutions, $\{x_i \leftarrow c\}$: substitute variables by constants.
2. Projections, $\{x_i \leftarrow c, x_i \leftarrow x_j \oplus c\}$: substitute variables by constants and other variables and their negations.

3. Affine substitutions, $\{x_i \leftarrow \bigoplus_{j \in J} x_j \oplus c\}$: substitute variables by affine functions of other variables.
4. Quadratic substitutions, $\{x_i \leftarrow p \colon \deg(p) \le 2\}$: substitute variables by degree two polynomials of other variables.

Case analysis We then prove the main technical result stating that for any circuit from the class $\mathcal{C}$ there exists (and can be constructed efficiently) an allowed substitution $x_i \leftarrow f \in \mathcal{S}$ such that the measure μ is reduced by a sufficient amount under both substitutions $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$.

#SAT upper bounds As an immediate consequence, we obtain an upper bound on the running time of an algorithm solving #SAT for circuits from $\mathcal{C}$. The corresponding algorithm takes as input a circuit, branches into two cases $x_i \leftarrow f$ and $x_i \leftarrow f \oplus 1$, and proceeds recursively. When applying a substitution $x_i \leftarrow f \oplus c$, it replaces all occurrences of x_i by a subcircuit computing $f \oplus c$. The case analysis provides an upper bound on the size of the resulting recursion tree.

Circuit size lower bounds Then, by taking a function that survives under sufficiently many allowed substitutions, we obtain lower bounds on the average case and worst case circuit complexity of the function. Below, we describe such functions, i.e., dispersers and extractors for the classes of sources under consideration.

1. The class of bit fixing substitutions generates the class of *bit-fixing sources* [18]. Extractors for bit-fixing sources find many applications in cryptography (see [22] for an excellent survey of the topic). The standard function that is a good disperser and extractor for such sources is the parity function $x_1 \oplus \cdots \oplus x_n$.
2. Projections define the class of *projection sources* [46]. Dispersers for projections are used to prove lower bounds for depth-three circuits [46]. It is shown [46] that a binary BCH code with appropriate parameters is a disperser for $n - o(n)$ substitutions. See [48] for an example of extractor with good parameters for projection sources.
3. Affine substitutions give rise to the class of *affine sources*. Dispersers for affine sources find applications in circuit lower bounds [19, 20, 25]. There are several known constructions of dispersers [6, 55] and extractors [65, 39, 5, 40] that are resistant to $n - o(n)$ substitutions.
4. The class of quadratic substitutions generates a special case of *polynomial sources* [24, 5] and *quadratic varieties sources* [23]. An explicit construction of disperser for quadratic varieties sources would imply new circuit lower bounds [26]. Although an explicit construction of a function resistant to sufficiently many quadratic substitutions is not currently known, it is easy to show that a random function is resistant to any $n - o(n)$ quadratic substitutions.

Due to the page limit of this extended abstract, we have to omit many proofs, which can be found in the full version [27].

2 Preliminaries

2.1 Boolean functions

We denote by B_n the set of all n-variate Boolean functions and define $U_2 = B_2 \setminus \{\oplus, \equiv\}$ as the set of all *binary* Boolean functions except for parity and its complement.

The set of all sixteen binary Boolean functions $f(x, y) \in B_2$ can be classified as follows: 1) two constant functions: 0 and 1; we also call them *trivial*; 2) four functions that depend essentially on one of the arguments only: x, $x \oplus 1$, y, $y \oplus 1$; we call them *degenerate*; 3) eight *and-type* functions: $(x \oplus a) \cdot (y \oplus b) \oplus c$ where $a, b, c \in \{0, 1\}$; 4) two *xor-type* functions: $x \oplus y \oplus a$, where $a \in \{0, 1\}$.

Hence U_2 consists of all binary functions except for xor-type functions. An important property of binary and-type functions $(x \oplus a) \cdot (y \oplus b) \oplus c$, useful for case analyses, is the following: one can turn this function into a constant c by assigning $x \leftarrow a$ or $y \leftarrow b$.

2.2 Dispersers and Extractors

Let $x_1, \ldots, x_n$ be Boolean variables, and $f \in B_{n-1}$ be a function of $n-1$ variables. We say that $x_i \leftarrow f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ is a substitution to the variable x_i.

Let $g \in B_n$ be a function, then the *restriction* of g under the substitution f is a function $h = (g|x_i \leftarrow f)$ of $n-1$ variables, such that $h(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = g(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$, where $x_i = f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Similarly, if $K \subseteq \{0, 1\}^n$ is a subset of the Boolean cube, then the *restriction* of K under this substitution is $K' = (K|x_i \leftarrow f)$, such that $(x_1, \ldots, x_n) \in K'$ if and only if $(x_1, \ldots, x_n) \in K$ and $x_i = f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

For a family of functions $\mathcal{F} = \{f : \{0, 1\}^* \to \{0, 1\}\}$ we define a set of corresponding substitutions $\mathcal{S}(\mathcal{F})$ that contains the following substitutions: for every $1 \le i \le n, c \in \{0, 1\}, f \in \mathcal{F}$, $\mathcal{S}$ contains the substitution $x_i \leftarrow f(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \oplus c$.

Let $\mathcal{S}$ be a set of substitutions. We say that a set $K \subseteq \{0, 1\}^n$ is an $(\mathcal{S}, n, r)$-*source*[1] if it can be obtained from $\{0, 1\}^n$ by applying at most r substitutions from $\mathcal{S}$.

A function $f \in B_n$ is called an $(\mathcal{S}, n, r)$-*disperser*[2] if it is not constant on every $(\mathcal{S}, n, r)$-source. A function $f \in B_n$ is called an $(\mathcal{S}, n, r, \varepsilon)$-*extractor* if $|\Pr_{x \leftarrow K}[f(x) = 1] - 1/2| \le \varepsilon$ for every $(\mathcal{S}, n, r)$-source K.

2.3 Circuits

A circuit over the basis $\Omega \subseteq B_2$ is a directed acyclic graph with the following properties: 1) the indegree of each node is either zero or two; 2) each node of zero indegree is labeled by a variable and is called an *input* or an *input gate*; 3) each node of indegree two is labeled with a binary Boolean function from Ω called an *operation* of this gate; the node itself is called an *internal gate* or just a *gate*; 4) there is a unique node of outdegree zero and it is called an *output*. Such a circuit computes in a natural way a function from B_n, where n is the number of input gates of the circuit. In this paper, we consider circuits over the bases $\Omega = B_2$ and $\Omega = U_2$.

An *xor-gate* (*and-gate*) is a gate computing an xor-type (and-type, respectively) operation. A k-*gate* (k^+-*gate*) is a gate of outdegree exactly k (at least k, respectively).

For a circuit C, by $s(C)$ we denote the *size* of C, that is, the number of internal gates of C. By $i(C)$ and $i_1(C)$ we denote the total number of input gates of C and the number of input 1-gates, respectively. For a function $f \in B_n$, by $C_\Omega(f)$ we denote the minimal size of a circuit over Ω computing f.

For two Boolean functions $f, g \in B_n$, the correlation between them is defined as

$$\mathrm{Cor}(f, g) = \left| \Pr_{x \leftarrow \{0,1\}^n}[f(x) = g(x)] - \Pr_{x \leftarrow \{0,1\}^n}[f(x) \neq g(x)] \right| = 2\left| \frac{1}{2} - \Pr_{x \leftarrow \{0,1\}^n}[f(x) \neq g(x)] \right|.$$

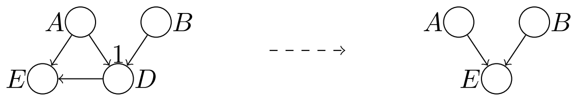
[1] Usually in the literature a source corresponds to a distribution over a subset of $\{0, 1\}^n$. In this paper, we focus only on uniform distributions, so we associate a source with its support.

[2] In this paper, we consider only dispersers and extractors with one bit outputs.

For a function $f \in B_n$, and $0 \leq \varepsilon \leq 1$, by $C_\Omega(f, \varepsilon)$ we denote the minimal size of a circuit over Ω computing function g such that $\operatorname{Cor}(f, g) \geq \varepsilon$.

2.4 Circuit normalization

A gate is called *useless* if it is a 1-gate and is fed by a predecessor of its successor:



In this case E actually computes a binary operation of A and B and this operation can be computed in the gate E directly. This might require to change an operation at E (if this circuit is over U_2 then E still computes an and-type operation of A and B as an xor-type binary function requires three gates in U_2).

By *normalizing* a circuit we mean removing all gates that compute trivial or degenerate operations and removing all useless gates. Note that normalization does not change the function computed by a circuit. It might however change the operations at some gates and outdegrees of some gates (in particular, input gates).

In the proofs of the paper we implicitly assume that if two gates are fed by the same variable then either there is no wire between them or each of the gates feed also some other gate (otherwise, one of the gates would be useless) and hence we do not care about this wire between the gates.

2.5 Circuit complexity measures

A function μ mapping circuits to non-negative real values is called a *circuit complexity measure* if for any circuit C,

- normalization of C does not increase its measure, and
- if $\mu(C) = 0$ then C has no gates.

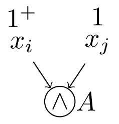
For a fixed circuit complexity measure μ, and function $f \in B_n$, we define $\mu(f)$ to be the minimum value of $\mu(C)$ over circuits C computing f. Similarly, we define $\mu(f, \varepsilon)$ to be the minimum value of $\mu(C)$ over circuits C computing g such that $\operatorname{Cor}(f, g) \geq \varepsilon$.

In this paper, we focus on the following two circuit complexity measures:

- $\mu(C) = s(C) + \alpha \cdot i(C)$ where $\alpha \geq 0$ is a constant;
- $\mu(C) = s(C) + \alpha \cdot i(C) - \sigma \cdot i_1(C)$ where $\alpha \geq 0, \sigma \leq 1$ are constants.

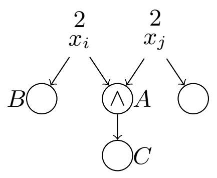
It is not difficult to see that these two functions are indeed circuit complexity measures if $\alpha \geq 0$ and $\sigma \leq 1$. The condition $\sigma \leq 1$ is needed to guarantee that if by removing a degenerate gate we increase the outdegree of a variable, the measure does not increase (an example is given on the next page).

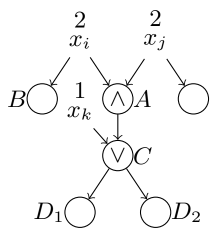
Intuitively we include the term $i(C)$ into the measure to handle cases like the one below (throughout the paper, we use labels above the gates to indicate their outdegree):



In this case, by assigning $x_i \leftarrow 0$ we make the circuit independent of x_j, so the measure is reduced by at least 2α. Usually, our goal is to show that we can find a substitution to a variable that eliminates at least some constant number k of gates, that is, to show a complexity decrease of at least $k + \alpha$. Thus, by choosing a large enough value of α we can always guarantee that $2\alpha \geq \alpha + k$. Thus, in the case above we do not even need to count the number of gates eliminated under the substitution.

The measure $\mu(C) = s(C) + \alpha \cdot i(C) - \sigma \cdot i_1(C)$ allows us to get an advantage of new 1-variables that are introduced during splitting.





For example, by assigning $x_i \leftarrow 0$ in a situation like the one in the left picture we reduce the measure by at least $3 + \alpha + \sigma$. As usual, the advantage comes with a related disadvantage. If, for example, a closer look at the circuit from the left part reveals that it actually looks like as shown on the right, then by assigning $x_i \leftarrow 0$ we introduce a new 1-variable x_j, but also loose one 1-variable (namely, x_k is now a 2-variable). Hence, in this case μ is reduced only by $(3 + \alpha)$ rather than $(3 + \alpha + \sigma)$. That is, our initial estimate was too optimistic. For this reason, when use a measure with $i_1(C)$ we check carefully for each eliminated gate if this elimination increases the degree of a 1-variable.

2.6 Splitting numbers and splitting vectors

Let μ be a circuit complexity measure and C be a circuit. Consider a recursive algorithm solving #SAT on C by repeated substitutions. Assume that at the current step the algorithm chooses k variables $x_1, \dots, x_k$ and k functions $f_1, \dots, f_k$ to substitute these variables and branches into 2^k possible situations: $x_1 \leftarrow f_1 \oplus c_1, \dots, x_k \leftarrow f_k \oplus c_k$ for all possible $c_1, \dots, c_k \in \{0, 1\}$ (in other words, it partitions the Boolean hypercube $\{0, 1\}^n$ into 2^k subsets).[3] For each substitution, we normalize the resulting circuit. Let us call the 2^k circuits $C_1, \dots, C_{2^k}$. We say that the current step has a *splitting vector* $v = (a_1, \dots, a_{2^k})$ w.r.t. μ if for all $i \in [2^k]$, $\mu(C) - \mu(C_i) \geq a_i > 0$. That is, the splitting vector gives a lower bound on the complexity decrease under the considered substitution. The *splitting number* $\tau(v)$ is the unique positive root of the equation $\sum_{i \in [2^k]} x^{-a_i} = 1$.

Splitting vectors and numbers are heavily used to estimate the running time of recursive algorithms. Below we assume that k is bounded by a constant. In all the proofs of this paper either $k = 1$ or $k = 2$, that is, we always estimate the effect of assigning either one or two variables. If an algorithm always splits with a splitting number at most β then its running time is bounded by $O^*(\beta^{\mu(C)})$.[4] To show this one notes that the recursion tree of this algorithm is 2^k-ary and $k = O(1)$ so it suffices to estimate the number of leaves. The

[3] Sometimes it is easier to consider vectors of length that is not a power of 2 too. For example, we can have a branching into three cases: one with one substituted variable, and two with two substituted variables. All the results from this paper can be naturally generalized to this case. For simplicity, we state the results for splitting vectors of length 2^k only.

[4] O^* suppresses factors polynomial in the input length.

number of leaves $T(\mu)$ satisfies the recurrence $T(\mu) \leq \sum_{i\in[2^k]} T(\mu - a_i)$ which implies that $T(\mu) = O(\tau(v)^\mu)$ (we assume also that $T(\mu) = O(1)$ when $\mu = O(1)$). See, e.g., [37] for a formal proof.

For a splitting vector $v = (a_1, \dots, a_{2^k})$ we define the following related quantities:

$$\overline{v}_{\max} = \max_{i\in[2^k]} \left\{\frac{a_i}{k}\right\}, \quad \overline{v}_{\min} = \min_{i\in[2^k]} \left\{\frac{a_i}{k}\right\}, \quad \overline{v}_{\text{avg}} = \frac{\sum_{i\in[2^k]} a_i}{k2^k}.$$

Intuitively, $\overline{v}_{\max}$ ($\overline{v}_{\min}$, $\overline{v}_{\text{avg}}$) is a (lower bound for) the maximum (minimum, average, respectively) complexity decrease per single substitution.

We will need the following estimates for the splitting numbers. It is known that a balanced binary splitting vector is better than an unbalanced one: $2^{1/a} = \tau(a, a) < \tau(a + b, a - b)$ for $0 < b < a$ (see, e.g., [37]). There is a known upper bound on $\tau(a, b)$.

▶ **Lemma 1.** $\tau(a, b) \leq 2^{1/\sqrt{ab}}$.

In the following lemma we provide an asymptotic estimate of their difference.

▶ **Lemma 2** (Gap between $\tau(a_1+b, a_2+b)$ and $\tau((a_1+a_2)/2+b, (a_1+a_2)/2+b) \leq 2^{\frac{1}{(a_1+a_2)/2+b}}$). *Let $a_1 > a_2 > 0$, $a' = (a_1 + a_2)/2$ and $\delta(b) = \tau(a_1 + b, a_2 + b) - 2^{\frac{1}{a'+b}}$. Then, $\delta(b) = O((a_1 - a_2)^2/b^3)$ as $b \to \infty$.*

2.7 Azuma's inequality

Following the approach from [13], we use a variant of Azuma's inequality with one-sided boundedness condition in order to obtain average case lower bounds. The standard version of Azuma's inequality requires the difference between two consecutive variables to be bounded, [13] considers the case when the difference takes on only two values but is bounded only from one side. For our results, we need a slightly more general variant of the inequality: the difference between two consecutive variables takes on up to k values and is bounded from one side.

A sequence $X_0, \dots, X_m$ of random variables is a *supermartingale* if for every $0 \leq i < m$, $\mathrm{E}[X_{i+1}|X_i, \dots, X_0] \leq X_i$.

▶ **Lemma 3.** *Let $X_0, \dots, X_m$ be a supermartingale, let $Y_i = X_i - X_{i-1}$. If $Y_i \leq c$ and for fixed values of $(X_0, \dots, X_{i-1})$, the random variable Y_i is distributed uniformly over at most $k \geq 2$ (not necessarily distinct) values, then for every $\lambda \geq 0$:*

$$\Pr[X_m - X_0 \geq \lambda] \leq \exp\left(\frac{-\lambda^2}{2mc^2(k-1)^2}\right).$$

Note that we have an extra factor of $(k-1)^2$ comparing with the normal form of Azuma's inequality, but we do not assume that $X_i - X_{i-1}$ is bounded from below.

3 Toolkit

3.1 Main theorem

In this subsection we prove the main technical theorem that allows us to get circuit complexity lower bounds and #SAT upper bounds.

▶ **Definition 4.** Let $\{v_1, \dots, v_m\}$ be splitting vectors, and each v_i is a splitting vector of length $2^{t_i} \geq 2$. For a class of circuits Ω (e.g., $\Omega = B_2$ or $\Omega = U_2$), a set of substitutions $\mathcal{S}$, and a circuit complexity measure μ, we write

$$\mathrm{Splitting}(\Omega, \mathcal{S}, \mu) \preceq \{v_1, \dots, v_m\}$$

as a shortcut for the following statement: For any normalized circuit C from the class Ω one can find in time $\mathrm{poly}(|C|)$ either a substitution[5] from $\mathcal{S}$ whose splitting vector with respect to μ belongs to the set $\{v_1, \dots, v_m\}$ or a substitution that trivializes the output gate of C. A substitution always trivializes at least one gate (in particular, when we assign a constant to a variable we trivialize an input gate) and eliminates at least one variable.

▶ **Theorem 5.** *If Splitting$(\Omega, \mathcal{S}, \mu) \preceq \{v_1, \dots, v_m\}$ and the longest splitting vector has length 2^k, then*

1. *There exists an algorithm solving #SAT for circuits over Ω in time $O^*(\gamma^{\mu(C)})$, where*

$$\gamma = \max_{i \in [m]} \{\tau(v_i)\}\,.$$

2. *If $f \in B_n$ is an $(\mathcal{S}, n, r)$-disperser, then*

$$\mu(f) \geq \beta_w \cdot (r - k + 1)\,, \text{ where } \beta_w = \min_{i \in [m]} \{\overline{v_i}_{\max}\}\,.$$

3. *If $f \in B_n$ is an $(\mathcal{S}, n, r, \varepsilon)$-extractor, then for every $\mu < \beta_a \cdot r$,*

$$\mu\,(f, \delta) \geq \mu, \text{ where } \beta_a = \min_{i \in [m]} \left\{\overline{v_i}_{\mathrm{avg}}\right\} \text{ and } \beta_m = \min_{i \in [m]} \{\overline{v_i}_{\min}\}\,,$$

$$\delta = \varepsilon + \exp\left(\frac{-(r \cdot \beta_a - \mu)^2}{2r(\beta_a - \beta_m)^2(2^k - 1)^2}\right)\,.$$

3.2 Discussion

Many known lower bounds for circuits with unrestricted depth can be proved using this framework, in particular, the strongest known lower bounds over B_2 and U_2. Schnorr [53] proved a $2n - \Theta(1)$ on C_{B_2} for a wide class of functions using $\mu(C) = s(C)$ and bit fixing substitutions. Stockmeyer [57] proved a $2.5n - \Theta(1)$ lower bound for symmetric functions using $\mu(C) = s(C)$ and a special case of projections: $\{x_i \leftarrow c, \{x_i \leftarrow f, x_j \leftarrow f \oplus 1\}\}$ (the latter "double" substitution essentially fixes $x_i \oplus x_j$ to 1; by applying such a substitution to, say, the majority function one gets the majority function of fewer inputs). Kojevnikov and Kulikov [34] improved the bound by Schnorr to $7n/3 - \Theta(1)$ using the measure $\mu(C) = 3x(C) + 2a(C)$ assigning different weights to xor-gates and and-gates. Demenkov and Kulikov [20] proved a $3n - o(n)$ lower bound for an affine disperser for sublinear dimension using $\mu(C) = s(C) + i(C)$ and affine substitutions. Recently, Find et al. [25] extended this approach to get a $(3 + 1/86)n$ lower bound for the same function using a few additional tricks (while the measure and the set of allowed substitutions are not easy to describe).

For the basis U_2, Schnorr [53] proved that the circuit size of parity is $3n - 3$ using bit fixing substitutions. Zwick [66] proved a $4n - \Theta(1)$ lower bound for symmetric functions using

[5] Here we assume that the circuit obtained from C by the substitution and normalization belongs to Ω too.

bit-fixing substitutions and $\mu(C) = s(C) - i_1(C)$. His measure was then used by Lachish and Raz [38] and by Iwama and Morizumi [32] to prove $4.5n - o(n)$ and $5n - o(n)$ lower bounds for strongly two-dependent functions. Recently, Demenkov et al. [21] gave a simpler proof of a $5n - o(n)$ lower bound for a linear function with $o(n)$ outputs. All these proofs use bit fixing substitutions only, however the case analysis can be simplified using also projections and a measure of the form $\mu = s + \alpha \cdot i$.

At the same time, there are known lower bound proofs that use additional tricks. E.g., Blum [8] to prove a $3n - o(n)$ lower bound over B_2 first considers a few cases when it is easy to remove three gates, and for all the remaining circuits shows a lower bound directly by counting the number of gates using some particular properties of the function under consideration.

Also, upper bounds for SAT and #SAT for various circuits classes (and for many other NP-hard problems) are proved by making substitutions recursively and using a carefully chosen measure to estimate the complexity decrease after substitutions.

The whole framework is a formalization of the following simple idea. To prove a lower bound ζn on circuit size one usually shows that there always exists a substitution $x_i \leftarrow f$ eliminating at least ζ gates from the circuit. By analysing also the complexity decrease under the substitution $x_i \leftarrow f \oplus 1$ one gets an upper bound for #SAT and an average case lower bound. Below we show an easy consequence of this: if one gets a very strong lower bound via short splitting vectors in this framework, then the corresponding #SAT-algorithm is also quite fast. That is, a superlinear circuit lower bound that uses only short splitting vectors in the framework implies a subexponential time (with respect to the size) algorithm for #SAT, which contradicts the Exponential Time Hypothesis.

▶ **Theorem 6.** *If for some set of substitutions $\mathcal{S}$, Splitting$(\Omega, \mathcal{S}, s + \alpha i) \preceq \{(a_1, b_1), \dots, (a_m, b_m)\}$, such that $\beta_w = \min_{i\in[m]} \max\{a_i, b_i\} = \omega(1)$ then #SAT can be solved in time $O^*(2^{o(s)})$.*

Note that due to the Sparsification Lemma [31] such an algorithm even over the basis U_2 contradicts the Exponential Time Hypothesis.

Although our "positive" results from Theorem 5 hold for splitting vectors of any length, this "negative" result from Theorem 6 holds only for splitting vectors of length 2. The authors do not know how to generalize this result to longer splitting vectors, and leave it as an open question.

4 Bounds for the basis U_2

4.1 Bit fixing substitutions: substituting variables by constants

We start with a well-known case analysis of a $3n - 3$ lower bound for the parity function over U_2 due to Schnorr [53]. Using this case analysis we reprove the bounds given recently by Chen and Kabanets [13] in our framework. The analysis is basically the same though the measure is slightly different. We provide these results mostly as a simple illustration of using the framework.

▶ **Lemma 7.** *Splitting$(U_2, \{x_i \leftarrow c\}, s + \alpha i) \preceq \{(\alpha, 2\alpha), (3 + \alpha, 3 + \alpha), (2 + \alpha, 4 + \alpha)\}$.*

▶ **Corollary 8.** *1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over U_2 of size at most $(3-\epsilon)n$ can be solved in time $(2-\delta)^n$.*

2. $C_{U_2}(x_1 \oplus \cdots \oplus x_n \oplus c) \geq 3n - 6$.

3. $C_{U_2}\left(x_1 \oplus \cdots \oplus x_n \oplus c, \exp\left(\frac{-(t-9)^2}{18(n-1)}\right)\right) \geq 3n - t$. *This, in particular, implies that* $\mathrm{Cor}(x_1 \oplus \cdots \oplus x_n \oplus c, C)$ *is negligible for any circuit C of size $3n - \omega(\sqrt{n}\log n)$.*

4.2 Projections: substituting variables by constants and other variables

In this subsection, we prove new bounds for the basis U_2.

▶ **Lemma 9.** *For $0 \leq \sigma \leq 1/2$,*

$$\textit{Splitting}(U_2, \{x_i \leftarrow c, x_i \leftarrow x_j \oplus c\}, s + \alpha i - \sigma i_1) \preceq \{(\alpha, 2\alpha), (2\alpha, 2\alpha, 2\alpha, 3\alpha), (3+\alpha+\sigma, 3+\alpha), (4+\alpha+\sigma, 2+\alpha)\}.$$

▶ **Corollary 10.** *1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over U_2 of size at most $(3.25-\epsilon)n$ can be solved in time $(2-\delta)^n$.*

2. Let $f \in B_n$ be an $\left(n, r(n) = n - \log^{O(1)}(n)\right)$-projections disperser from [40]. Then $C_{U_2}(f) \geq 3.5n - \log^{O(1)}(n)$.

3. Let $f \in B_n$ be an $\left(n, r(n) = n - \sqrt{n}, \varepsilon(n) = 2^{-n^{\Omega(1)}}\right)$-projections extractor from [48]. Then $C_{U_2}(f, \delta) \geq 3.25n - t$, where $\delta = 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(t-10.25\sqrt{n})^2}{190.125(n-\sqrt{n})}\right)$. This, in particular, implies that $\mathrm{Cor}(f, C)$ is negligible for any circuit C of size $3.25n - \omega(\sqrt{n}\log n)$.

5 Bounds for the basis B_2

5.1 Affine substitutions: substituting variables by linear sums of other variables

Here, we again start by reproving the bounds for B_2 by Chen and Kabanets [13] by using the case analysis by Demenkov and Kulikov [20].

▶ **Lemma 11.** $\textit{Splitting}(B_2, \{x_i \leftarrow \oplus_{j \in J} x_j \oplus c\}, \mu = s + \alpha i) \preceq \{(\alpha, 2\alpha), (2+\alpha, 3+\alpha)\}$.

▶ **Corollary 12.** *1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over B_2 of size at most $(2.5-\epsilon)n$ can be solved in time $(2-\delta)^n$.*

2. Let $f \in B_n$ be an $\left(n, r(n) = n - \log^{O(1)}(n)\right)$-affine disperser from [40]. Then $C_{B_2}(f) \geq 3n - \log^{O(1)}(n)$.

3. Let $f \in B_n$ be an $\left(n, r(n) = n - O(n/\log\log n), \varepsilon(n) = 2^{-n^{\Omega(1)}}\right)$-affine extractor from [39]. Then $C_{B_2}(f, \delta) \geq 2.5n - t$, where $\delta = 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(t-O(n/\log\log n))^2}{O(n)}\right)$. This, in particular, implies that $\mathrm{Cor}(f, C)$ is negligible for any circuit C of size $2.5n - \omega(n/\log\log n)$.

5.2 Quadratic substitutions: substituting variables by degree 2 polynomials of other variables

▶ **Lemma 13.** *For $0 \le \sigma \le 1/5$,*

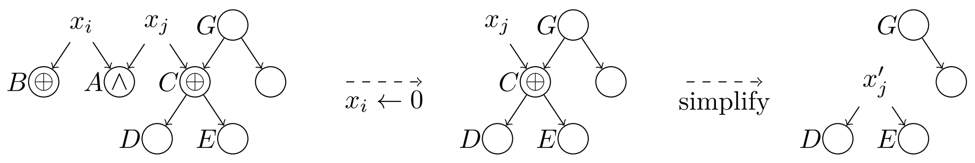
$$\text{Splitting}(B_2, \{x_i \leftarrow p\colon \deg(p) \le 2\}, s + \alpha i - \sigma i_1) \preceq \{(\alpha, 2\alpha), (2\alpha, 2\alpha, 2\alpha, 3\alpha), (3+\alpha-2\sigma, 3+\alpha-2\sigma), (3+\alpha+\sigma, 2+\alpha)\}\,.$$

▶ **Corollary 14.** *1. For any $\epsilon > 0$ there exists $\delta = \delta(\epsilon) > 0$ such that #SAT for circuits over B_2 of size at most $(2.6 - \epsilon)n$ can be solved in time $(2-\delta)^n$.*
2. Let $f \in B_n$ be an $(n, r(n) = n - o(n))$-quadratic disperser. Then $C_{B_2}(f) \ge 3n - o(n)$.
3. Let $f \in B_n$ be an $\left(n, r(n) = n - o(n), \varepsilon(n) = 2^{-\omega(\log n)}\right)$-quadratic extractor. Then $C_{B_2}(f, \delta) \ge 2.6n - t$, where $\delta = 2^{-n^{\Omega(1)}} + \exp\left(\frac{-(t-7.8(n-r(n)))^2}{121.68r(n)}\right)$. This, in particular, implies that $\operatorname{Cor}(f, C)$ is negligible for any circuit C of size $2.6n - g(n)$ for some $g(n) = o(n)$.

▶ Remark 1. Note that it is an open problem to find an explicit construction of quadratic disperser or extractor over $\mathbb{F}_2$ with $r = n - o(n)$. Any disperser for a slightly more general definition of quadratic varieties would also imply a new worst case lower bound [26].

▶ Remark 2. Note that the upper bound for #SAT can be improved using the following "forbidden trick", that is, a simplification rule that reduces the size of a circuit without changing the number of its satisfying assignments, but changes the function computed by the circuit.

In the proof of Lemma 13 set $\sigma = 0$ (that is, do not account for 1-variables). The set of splitting vectors then turn into By inspecting all the cases, we see that the splitting vector $(3 + \alpha, 2 + \alpha)$ only appears in one case. We can handle this case differently: split on x_i. When A is trivialized, x_j becomes a 1-variable feeding an xor-gate. It is not difficult to show that by replacing this gate with a new variable x'_j one gets a circuit with the same number of satisfying assignments.



This additional trick gives us the following set of splitting vectors: $\{(\alpha, 2\alpha), (2\alpha, 2\alpha, 2\alpha, 3\alpha), (3+\alpha, 3+\alpha), (4+\alpha, 2+\alpha)\}$. These splitting numbers give an algorithm solving #SAT in $(2-\delta(\epsilon))^n$ for B_2-circuits of size at most $(3 - \epsilon)n$ for $\epsilon > 0$.

Note that such a simplification rule does not fit into our framework since it *changes the function computed by a circuit.* It would be interesting to adjust the framework to allow such kind of simplifications (probably, by incorporating some new parameter to the measure).

6 Open problems

There are three natural questions left open in this paper.

1. Prove that a superlinear circuit lower bound in this framework violates the Exponential Time Hypothesis.

2. Give an explicit construction of quadratic dispersers (see Remark 1).
3. Adjust the framework to allow using natural simplification rules like replacing an xor gate fed by a 1-variable for both upper bounds for #SAT and lower bounds for circuit size (see Remark 2).

Acknowledgement. We would like to thank the anonymous reviewers for their helpful comments.

References

1. Kazuyuki Amano and Atsushi Saito. A nonuniform circuit class with multilayer of threshold gates having super quasi polynomial size lower bounds against NEXP. In *Proceedings of the 9th International Conference on Language and Automata Theory and Applications (LATA)*, pages 461–472, 2015.
2. Kazuyuki Amano and Atsushi Saito. A satisfiability algorithm for some class of dense depth two threshold circuits. *IEICE Transactions*, 98-D(1):108–118, 2015.
3. Kazuyuki Amano and Jun Tarui. A well-mixed function with circuit complexity $5n$: Tightness of the Lachish-Raz-type bounds. *Theor. Comput. Sci.*, 412(18):1646–1651, 2011.
4. Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating AC^0 by small height decision trees and a deterministic algorithm for $\#AC^0$ SAT. In *Proceedings of the 27th Conference on Computational Complexity (CCC)*, pages 117–125, 2012.
5. Eli Ben-Sasson and Ariel Gabizon. Extractors for polynomial sources over fields of constant order and small characteristic. *Theory of Computing*, 9:665–683, 2013.
6. Eli Ben-Sasson and Swastik Kopparty. Affine dispersers from subspace polynomials. *SIAM J. Comput.*, 41(4):880–914, 2012.
7. Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 163–173, 2014.
8. Norbert Blum. A Boolean function requiring $3n$ network size. *Theor. Comput. Sci.*, 28:337–345, 1984.
9. Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity (CCC)*, pages 8–12, 1998.
10. Venkatesan T. Chakaravarthy and Sambuddha Roy. Oblivious symmetric alternation. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 230–241, 2006.
11. Venkatesan T. Chakaravarthy and Sambuddha Roy. Arthur and Merlin as oracles. *Computational Complexity*, 20(3):505–558, 2011.
12. Ruiwen Chen. Satisfiability algorithms and lower bounds for Boolean formulas over finite bases. In *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II*, pages 223–234, 2015.
13. Ruiwen Chen and Valentine Kabanets. Correlation bounds and #SAT algorithms for small linear-size circuits. In *Proceedings of the 21st International Conference on Computing and Combinatorics (COCOON)*, pages 211–222, 2015.
14. Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015.

15 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #SAT algorithm for small De Morgan formulas. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II*, pages 165–176, 2014.

16 Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse MAX-SAT and MAX-k-CSP. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 33–45, 2015.

17 Ruiwen Chen, Rahul Santhanam, and Srikanth Srinivasan. Average-case lower bounds and satisfiability algorithms for small threshold circuits. In *Proceedings of the 31th Conference on Computational Complexity (CCC)*, pages 1:1–1:35, 2016.

18 Benny Chor, Oded Goldreich, Johan Håstad, Joel Friedman, Steven Rudich, and Roman Smolensky. The bit extraction problem of t-resilient functions (preliminary version). In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 396–407, 1985.

19 Gil Cohen and Igor Shinkar. The complexity of DNF of parities. In *Proceedings of the 7th Innovations in Theoretical Computer Science (ITCS) Conference*, pages 47–58, 2016.

20 Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 256–265, 2011.

21 Evgeny Demenkov, Alexander S. Kulikov, Olga Melanich, and Ivan Mihajlin. New lower bounds on circuit size of multi-output functions. *Theory of Computing Systems*, 56(4):630–642, 2015.

22 Yevgeniy Dodis. *Exposure-resilient cryptography.* PhD thesis, Massachusetts Institute of Technology, 2000.

23 Zeev Dvir. Extractors for varieties. *Computational Complexity*, 21(4):515–572, 2012.

24 Zeev Dvir, Ariel Gabizon, and Avi Wigderson. Extractors and rank extractors for polynomial sources. *Computational Complexity*, 18(1):1–58, 2009.

25 Magnus Gausdal Find, Alexander Golovnev, Edward Hirsch, and Alexander Kulikov. A better-than-$3n$ lower bound for the circuit complexity of an explicit function. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-166, 2015.

26 Alexander Golovnev and Alexander S. Kulikov. Weighted gate elimination: Boolean dispersers for quadratic varieties imply improved circuit lower bounds. In *Proceedings of the 7th Innovations in Theoretical Computer Science (ITCS) Conference*, pages 405–411, 2016.

27 Alexander Golovnev, Alexander S. Kulikov, Alexander Smal, and Suguru Tamaki. Circuit size lower bounds and #sat upper bounds through a general framework. *Electronic Colloquium on Computational Complexity (ECCC)*, TR16-022, 2016.

28 Johan Håstad. On the correlation of parity and small-depth circuits. *SIAM J. Comput.*, 43(5):1699–1708, 2014.

29 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 961–972, 2012.

30 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 479–488, 2013.

31 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

32 Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of $5n - o(n)$ for Boolean circuits. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 353–364, 2002.

33 Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 749–760, 2015.

34 Arist Kojevnikov and Alexander S. Kulikov. Circuit complexity and multiplicative complexity of Boolean functions. In *Proceedings of the 6th Conference on Computability in Europe (CiE)*, pages 239–245, 2010.

35 Ilan Komargodski and Ran Raz. Average-case lower bounds for formula size. In *Proceedings of the 45th Symposium on Theory of Computing (STOC)*, pages 171–180, 2013.

36 Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for De Morgan formula size. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 588–597, 2013.

37 Oliver Kullmann. Fundaments of branching heuristics. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 205–244. IOS Press, 2009.

38 Oded Lachish and Ran Raz. Explicit lower bound of $4.5n - o(n)$ for Boolean circuits. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 399–408, 2001.

39 Xin Li. A new approach to affine extractors and dispersers. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity (CCC)*, pages 137–147, 2011.

40 Xin Li. Improved two-source extractors, and affine extractors for polylogarithmic entropy. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-125, 2015.

41 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

42 Dániel Marx. Consequences of ETH: Tight bounds for various problems. In *Fine-Grained Complexity and Algorithm Design Boot Camp*, 2015. “https://simons.berkeley.edu/talks/daniel-marx-2015-09-03” (abstract, slides and archived video).

43 Atsuki Nagao, Kazuhisa Seto, and Junichi Teruyama. A moderately exponential time algorithm for k-IBDD satisfiability. In *Proceedings of the 14th International Symposium, on Algorithms and Data Structures (WADS)*, pages 554–565, 2015.

44 Sergey Nurk. An $O(2^{0.4058m})$ upper bound for circuit SAT. Technical report, PDMI, 2009.

45 Igor Carboni Oliveira. Algorithms versus circuit lower bounds. *Electronic Colloquium on Computational Complexity (ECCC)*, TR13-117, 2013.

46 Ramamohan Paturi, Michael E. Saks, and Francis Zane. Exponential lower bounds for depth three boolean circuits. *Computational Complexity*, 9(1):1–15, 2000.

47 Wolfgang J. Paul. A $2.5n$-lower bound on the combinational complexity of Boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977.

48 Anup Rao. Extractors for low-weight affine sources. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 95–101, 2009.

49 Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. Bounded depth circuits with weighted symmetric gates: Satisfiability, lower bounds and compression. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2016, to appear.

50 Rahul Santhanam. Circuit lower bounds for Merlin-Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

51 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 183–192, 2010.

52 Rahul Santhanam. Ironic complicity: Satisfiability algorithms and circuit lower bounds. *Bulletin of the EATCS*, 106:31–52, 2012.

53 Claus-Peter Schnorr. Zwei lineare untere schranken für die komplexität boolescher funktionen. *Computing*, 13(2):155–171, 1974.

54 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013.

55 Ronen Shaltiel. Dispersers for affine sources with sub-polynomial entropy. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 247–256, 2011.

56 Claude E. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28(1):59–98, 1949.

57 Larry J. Stockmeyer. On the combinational complexity of certain symmetric Boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.

58 Avishay Tal. #SAT algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-114, 2015.

59 Fengming Wang. NEXP does not have non-uniform quasipolynomial-size ACC circuits of $o(\log\log n)$ depth. In *Proceedings of the 8th Annual Conference on Theory and Applications of Models of Computation (TAMC)*, pages 164–170, 2011.

60 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.

61 Ryan Williams. Natural proofs versus derandomization. In *Proceedings of the 45th ACM Symposium on Theory of Computing Conference (STOC)*, pages 21–30, 2013.

62 Ryan Williams. Algorithms for circuits and circuits for algorithms. In *Proceedings of the 29th Annual IEEE Conference on Computational Complexity (CCC)*, pages 248–261, 2014.

63 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Proceedings of the 46th Symposium on Theory of Computing (STOC)*, pages 194–202, 2014.

64 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2, 2014.

65 Amir Yehudayoff. Affine extractors over prime fields. *Combinatorica*, 31(2):245–256, 2011.

66 Uri Zwick. A $4n$ lower bound on the combinational complexity of certain symmetric Boolean functions over the basis of unate dyadic Boolean functions. *SIAM J. Comput.*, 20(3):499–505, 1991.

On the Limits of Gate Elimination

Alexander Golovnev*[1], Edward A. Hirsch[2], Alexander Knop[3], and Alexander S. Kulikov[4]

1 **New York University, USA**
2 **St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia**
3 **St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia**
4 **St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia**

Abstract

Although a simple counting argument shows the existence of Boolean functions of exponential circuit complexity, proving superlinear circuit lower bounds for *explicit* functions seems to be out of reach of the current techniques. There has been a (very slow) progress in proving linear lower bounds with the latest record of $3\frac{1}{86}n - o(n)$. All known lower bounds are based on the so-called gate elimination technique. A typical gate elimination argument shows that it is possible to eliminate several gates from an optimal circuit by making one or several substitutions to the input variables and repeats this inductively. In this note we prove that this method cannot achieve linear bounds of cn beyond a certain constant c, where c depends only on the number of substitutions made at a single step of the induction.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases circuit complexity, lower bounds, gate elimination

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.46

1 Introduction

One of the most important and at the same time most difficult questions in theoretical computer science is proving circuit lower bounds. A binary Boolean circuit is a directed acyclic graph with nodes of in-degree either 0 or 2. Nodes of in-degree 0 are called inputs and are labeled by variables $x_1, \ldots, x_n$. Nodes of in-degree 2 are called gates and are labeled by binary Boolean functions. One of the nodes is additionally labeled as the output of the circuit. The output gate computes a Boolean function $\{0,1\}^n \to \{0,1\}$ in a natural way. The size of a circuit C is defined as the number of gates in C and is denoted by $\texttt{gates}(C)$. By $\texttt{inputs}(C)$ we denote the number of inputs of C. A circuit complexity measure μ is a function assigning each circuit a non-negative real number. In particular, `gates` and `inputs` are circuit complexity measures.

By B_n we denote the set of all Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$. For a circuit complexity measure μ and a function $f \in B_n$, by $\mu(f)$ we denote the minimum value of $\mu(C)$ over all circuits C computing f. For example, $\texttt{gates}(f)$ is the miniumum size of a circuit computing f.

* This research is partially supported by NSF grant 1319051.

By comparing the number of small size circuits with the total number 2^{2^n} of Boolean functions of n variables, one concludes that almost all such functions have circuit size at least $\Omega(\frac{2^n}{n})$. This was shown by Shannon in 1949 [30]. However we still do not have an example of a function from NP that requires circuits of superlinear size. The currently strongest known lower bound is $(3 + \frac{1}{86})n - o(n)$ [11].

The lack of strong lower bounds is a consequence of the lack of methods for proving lower bounds for general circuits. Practically, the only known method for proving lower bounds is the gate elimination method. We illustrate this method with a simple example. Consider the function $\mathrm{MOD}^n_{3,r}\colon \{0,1\}^n \to \{0,1\}$ which outputs 1 if and only if the sum of n input bits is congruent to r modulo 3. One can prove that $\texttt{gates}(\mathrm{MOD}^n_{3,r}) \geq 2n - 4$ for any $r \in \{0,1,2\}$ by induction on n. The base case $n \leq 2$ clearly holds. Assume that $n \geq 3$ and consider an optimal circuit C computing $\mathrm{MOD}^n_{3,r}$ and its topologically first (with respect to some topological ordering) gate G. This gate is fed by two different variables x_i and x_j (if they were the same variable, the circuit would not be optimal). A crucial observation is that it cannot be the case that the out-degrees of both x_i and x_j are equal to 1. Indeed, in this case the whole circuit would depend on x_i and x_j through the gate G only. In particular, the four ways of fixing the values of x_i and x_j would give at most two different subfunctions (corresponding to $G = 0$ and $G = 1$), while $\mathrm{MOD}^n_{3,r}$ has three such different subfunctions: $\mathrm{MOD}^{n-2}_{3,0}$, $\mathrm{MOD}^{n-2}_{3,1}$, and $\mathrm{MOD}^{n-2}_{3,2}$. Assume, without loss of generality, that x_i has out-degree at least 2. We then substitute $x_i \leftarrow 0$, eliminate the gates fed by x_i from the circuit and proceed by induction. The eliminated gates are those fed by x_i. After the substitution, each such gate computes either a constant or a unary function of the other input of the gate, so can be eliminated. The resulting function computes $\mathrm{MOD}^{n-1}_{3,r}$. Thus we get by induction: $\texttt{gates}(\mathrm{MOD}^n_{3,r}) \geq \texttt{gates}(\mathrm{MOD}^{n-1}_{3,r}) + 2 \geq (2(n-1) - 4) + 2 = 2n - 4$. This proof was given by Schnorr in 1984 [29]. In fact, it works for a wider class of functions $Q^n_{2,3}$ containing functions that have at least three different subfunctions with respect to any two variables.

This example reveals the main idea of the gate elimination process: a lower bound is proved inductively by finding at each step an appropriate substitution that eliminates many gates from the given circuit. At the same time, using just bit-fixing substitutions is not enough for proving even stronger than $2n$ lower bounds: the class $Q^n_{2,3}$ contains, in particular, a function THR^n_2 that outputs 1 iff $\sum_{i=1}^n x_i \geq 2$ whose circuit complexity is known to be at most $2n + o(n)$ [10] (see also Theorem 2.3 in [35]). For this reason, known proofs of stronger lower bounds use various additional tricks.

- One can use amortized analysis of the number of eliminated gates. For example, one can show that at each step one can either find a substitution that eliminates 3 gates *or* a pair of consecutive substitutions, the first one eliminating 2 gates and the next one eliminating 4 gates.
- They also substitute variables not just by constants but by affine functions, quadratic functions, and even arbitrary functions of other variables.
- In order to amortize for steps that eliminate too few gates, they also use more intricate complexity measures that combine the number of gates with the number of variables or other quantities.

We give an overview of known lower bounds and used tricks in Section 2.

One can guess that the gate elimination method changes only the top of a circuit in few places and thus cannot eliminate many gates. In general, this intuition fails (it is easy to present examples where a single substitution greatly simplifies a function, in particular, every substitution to a function of the highest possible complexity $2^n/n$ (see Theorem 2.1

and below in [35]) lowers the complexity of this function almost twice as for a function of $n-1$ variables it cannot exceed $2^{n-1}/(n-1)+o(2^{n-1}/(n-1))$. However, in this paper we manage to make this intuition work for specially designed functions that compose gadgets satisfying certain rather general properties with arbitrary base functions. We show that certain formalizations of the gate elimination method cannot prove superlinear lower bounds. We prove that one cannot reduce the complexity of the designed functions by more than a constant using any constant number of substitutions of any type (that is, we allow to substitute variables by arbitrary functions). The complexity of a function may be counted as any complexity measure (i.e., a nonnegative function of a circuit) varying from the number of gates to any subadditive function. For recently popular measures that combine the number of gates with the number of inputs we prove a stronger result (namely, one cannot prove lower bounds beyond cn for a certain specific constant c; this constant may depend on the number m of consecutive substitutions made in one step of the induction but does not depend on the substitutions themselves, $m = 1$ or 2 in modern proofs).

The paper is organized as follows. In Section 2 we list known proofs based on gate elimination, we discuss their differences and limits. Section 3 presents several examples that lead us to the main questions of this work. This section contains main results of the paper: provable limits of the gate elimination method for various complexity measures. Section 4 contains a brief overview of the known barriers for proving circuit lower bounds. Finally, Section 5 concludes the work with open questions.

2 Known Lower Bounds Proofs

Improving Schnorr's $2n$ lower bound proof mentioned above is already a non-trivial task. It can be the case that all variables in the given circuit feed two parity gates. In this case, substituting any variable by any constant eliminates just two gates from this circuit. In 1977, Stockmeyer [31] used the following clever trick to prove a $2.5n-\Theta(1)$ lower bound for many symmetric functions including all MOD_m^n functions for constant $m \geq 3$. The idea is to eliminate five gates by *two* consecutive substitutions. This time, instead of substituting $x_i \leftarrow c$ where $c \in \{0,1\}$ we substitute $x_i \leftarrow f, x_j \leftarrow f \oplus 1$ where f is an *arbitrary function* that does not depend on x_i and x_j. One should be careful with such substitutions as they potentially might produce a subfunction outside of the class of functions for which we are currently proving a lower bound by induction. At the same time, one can see that, for example, $\mathrm{MOD}_{3,0}^n$ function turns into $\mathrm{MOD}_{3,2}^{n-2}$ function under the substitution $x_i \leftarrow f, x_j \leftarrow f \oplus 1$. Indeed, this substitution just forces the sum of x_i and x_j to be equal to 1 (both over integers and over the field of size two).

In 1984, Blum [5], following the work by Paul [25], proved a $3n-o(n)$ lower bound for an artificially constructed Boolean function of $n+3\log n+3$ variables. The input of this function consists of n variables $X = \{x_1,\ldots,x_n\}$ and $3\log n+3$ variables A. The following "universality" property of this function is essential for Blum's proof: for any two variables $x_i, x_j \in X$ one can assign constants to variables from A to turn the output of the function to be equal to both $x_i \wedge x_j$ and $x_i \oplus x_j$. Blum first applies the standard gate elimination procedure to variables from X using a carefully chosen induction hypothesis that states a circuit size lower bound in terms of the number of variables from X that are still "alive": if there is a substitution $x_i \leftarrow f$ that eliminates at least three gates, perform this substitution and proceed inductively. Note that the used function allows to substitute variables from X by arbitrary functions, but at the same time one is allowed to substitute variables from X, but not from A. In the remaining case, Blum counts the number of gates of out-degree at least 2: he shows that due to the special properties of the function, any circuit computing it must contain many such gates. This gives a lower bound on the size of a circuit.

In 2011, Demenkov and Kulikov [7] presented a different proof of essentially the same $3n - o(n)$ lower bound for a different function. The function they use is an affine disperser for dimension $d = o(n)$, which is by definition non-constant on any affine subspace of dimension at least d. This property allows to make at least $n - o(n)$ affine substitutions (that is, substitutions of the form $x_i \leftarrow \bigoplus_{j \in J} x_j \oplus c$ where $i \notin J \subseteq [n]$ and $c \in \{0, 1\}$) before the function trivializes. The proof also uses a non-standard circuit complexity measure: for a circuit C, $\mu(C) = \texttt{gates}(C) + \texttt{inputs}(C)$. This trick is used to amortize the case when by substituting one variable one also removes the dependence on another variable. One shows that for any circuit there is a substitution that reduces μ by at least 4 (or makes the whole circuit a constant). This implies, by induction, that for any circuit C computing an affine disperser for dimension $o(n)$,

$$\texttt{gates}(C) + \texttt{inputs}(C) \geq 4(n - o(n)), \tag{1}$$

which in turn implies that $\texttt{gates}(C) \geq 3n - o(n)$. To find an appropriate affine substitution, one considers the topologically first gate A that computes a non-linear binary operation. If A is fed by two variables x_i and x_j of out-degree 1, we substitute $x_i \leftarrow c$ to make A constant. This eliminates A and its successor from the circuit as well as the dependence on both x_i and x_j. Hence both $\texttt{gates}$ and $\texttt{inputs}$ are reduced by at least 2, and μ is reduced by at least 4. If, say, x_i has out-degree at least 2, we just substitute x_i by the constant that makes A constant: this eliminates the gates fed by x_i and all successors of A (at least three gates in total) and the dependence on x_i, hence μ is reduced by 4 again. In the remaining case, one of the inputs to A is a gate computing an affine function $\bigoplus_{j \in J} x_j \oplus c$. We make it constant by substituting $x_i \leftarrow \bigoplus_{j \in J \setminus \{i\}} x_j \oplus c'$. This eliminates this gate, the gate A, and the successors of A. Thus, μ is reduced by at least 4 again.

Find et al. [11] pushed the lower bound $3n - o(n)$ for affine dispersers further to $(3 + \frac{1}{86})n - o(n)$ by using several new tricks. They generalize the computational model to allow cycles in circuits, use quadratic substitutions (that are turned into affine substitutions in the end of the gate elimination process), and use a carefully chosen circuit complexity measure which besides the number of gates and inputs also depends on the number of certain local bottleneck configurations and the number of quadratic substitutions.

The first explicit construction of an affine disperser for sublinear dimension ($d = o(n)$) was presented relatively recently by Ben-Sasson and Kopparty [4]. While such constructions of higher degree dispersers for sublinear dimension are not yet known, these dispersers do exist, and a lower bound of $3.1n$ has been shown for them in [13] using the circuit complexity measure $\mu_\alpha(C) = \texttt{gates}(C) + \alpha \cdot \texttt{inputs}(C)$ ($\alpha > 0$ is a constant) and quadratic substitutions.

We summarize the discussed lower bounds proofs in the table below.

Bound	Class of functions	Measure	Substitutions
$2n$ [29]	$Q^n_{2,3}$	$\texttt{gates}$	$x_i \leftarrow c$
$2.5n$ [31]	symmetric	$\texttt{gates}$	$x_i \leftarrow c$, $\{x_i \leftarrow f, x_j \leftarrow f \oplus 1\}$
$3n$ [5]	artificial	$\texttt{gates}$	arbitrary: $x_i \leftarrow f$
$3n$ [7]	affine dispersers	$\texttt{gates} + \texttt{inputs}$	linear: $x_i \leftarrow \bigoplus_{j \in J} x_j \oplus c$
$3.01n$ [11]	affine dispersers	$\texttt{gates} + \alpha\texttt{inputs} + \cdots$	quadratic: $x_i \leftarrow f$, deg ≤ 2
$3.1n$ [13]	quadratic dispersers	$\texttt{gates} + \alpha\texttt{inputs}$	quadratic: $x_i \leftarrow f$, deg ≤ 2

It is also interesting to note that there is a trivial limitation for the first three proofs in the table above: the corresponding classes of functions contain functions of linear circuit complexity. The class $Q^n_{2,3}$ contains the function THR^n_2 (that outputs 1 iff the sum of n input bits is at least 2) of circuit size $2n + o(n)$. The class of symmetric functions used by Stockmeyer contains the function MOD^n_4 whose circuit size is at most $2.5n + \Theta(1)$. The circuit size of Blum's function is upper bounded by $6n + o(n)$. At the same time it is not known whether there are affine dispersers of sublinear dimension that can be computed by linear size circuits.

3 Limits of Gate Elimination

3.1 Notation

Let $X = \{x_1, \dots, x_n\}$ be a set of Boolean variables. A *substitution* ρ of a set of variables $R \subseteq X$ is a set of $|R|$ restrictions of the form

$$r_i = f_i(x_1, \dots, x_n),$$

one restriction for each variable $r_i \in R$, where f_i depends only on variables from $X \setminus R$. The degree of a substitution is the maximum degree of f_i's represented as Boolean polynomials. The size of a substitution is $|R|$. Substitutions of size m are called m-substitutions.

Given an m-substitution ρ and a function f, one can naturally define a new function $f|_\rho$ that has m fewer arguments than f.

A function f *depends* on a variable x if there is a substitution ρ of constants to all other variables such that $f|_\rho(0) \neq f|_\rho(1)$.

As we saw in Section 2, gate elimination proofs sometimes track sophisticated *complexity measure* μ rather than just number of gates, for example, the measure $\mu(f) = \mathtt{gates}(f) + \alpha \cdot \mathtt{inputs}(f)$ for a constant α.

A gate elimination argument uses a certain nonnegative complexity measure μ, a family of substitutions $\mathcal{S}$, a family of functions $\mathcal{F}$, a function $\mathtt{gain}\colon \mathbb{N} \to \mathbb{R}$, and a certain predicate $\mathtt{stop}$, and includes proofs of the following statements:

1. (Measure usefulness.) If $\mu(f)$ is large, then $\mathtt{gates}(f)$ is large.
2. (Invariance.) For every $f \in \mathcal{F}$ and $\rho \in \mathcal{S}$, either $f|_\rho \in \mathcal{F}$ or $\mathtt{stop}(f|_\rho)$.
3. (Induction step.) For every $f \in \mathcal{F}$ with $\mathtt{inputs}(f) = n$, there is a substitution $\rho \in \mathcal{S}$ such that $\mu(f|_\rho) \le \mu(f) - \mathtt{gain}(n)$. (In known proofs, $\mathtt{gain}(n)$ is constant.)

The family must contain functions f such that $\mathtt{stop}(f|_{\rho_1,\dots,\rho_s})$ is not reached for sufficiently many substitutions from $\mathcal{S}$ (for example, for $s = 0.999 \cdot \mathtt{inputs}(f)$ substitutions).

In what follows, we prove that every gate elimination argument fails to prove a strong lower bound, for many functions of (virtually) arbitrarily large complexity.

3.2 Introductory Example

We start by providing an elementary construction of functions that are resistant with respect to any constant number of arbitrary substitutions, i.e., such substitutions eliminate only a constant number of gates. In the next sections, we generalize this construction to capture other complexity measures.

Consider a function $f\colon \{0,1\}^n \to \{0,1\}$ and let $f \diamond \mathrm{MAJ}_3$ be a function of $3n$ variables resulting from f by replacing each of its input variables x_i by the majority function of three fresh variables x_{i1}, x_{i2}, x_{i3}:

$$(f \diamond \mathrm{MAJ}_3)(x_{11}, x_{12}, \dots, x_{n3}) = f(\mathrm{MAJ}_3(x_{11}, x_{12}, x_{13}), \dots, \mathrm{MAJ}_3(x_{n1}, x_{n2}, x_{n3}))\,,$$

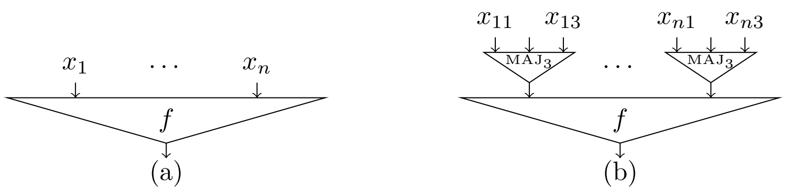


Figure 1 (a) A circuit for f. (b) A circuit for $f \diamond \mathrm{MAJ}_3$.

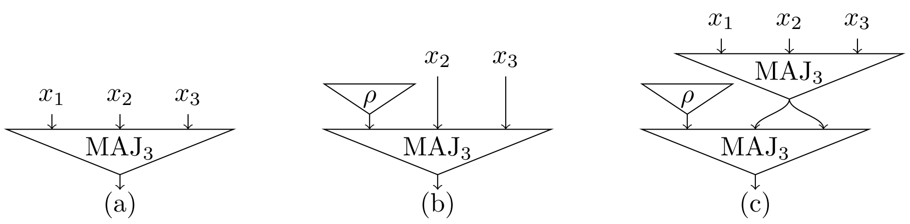


Figure 2 (a) A circuit computing the majority of three bits x_1, x_2, x_3. (b) A circuit resulting from substitution $x_1 \leftarrow \rho$. (c) By adding another gadget to a circuit with x_1 substituted, we force it to compute the majority of x_1, x_2, x_3.

see Fig. 1. Consider a circuit C of the smallest size computing $f \diamond \mathrm{MAJ}_3$. We claim that no substitution $x_{ij} \leftarrow \rho$, where ρ is any function of all the remaining variables, can remove from C more than 5 gates: $\texttt{gates}(C) - \texttt{gates}(C|_{x_{ij} \leftarrow \rho}) \leq 5$. We are going to prove this by showing that one can attach a gadget of size 5 to the circuit $C|_{x_{ij} \leftarrow \rho}$ and obtain a circuit that computes f. This is explained in Fig. 2. Formally, assume, without loss of generality, that the substituted variable is x_{11}. We then take a circuit C' computing $f|_{x_{11} \leftarrow \rho}$ and use the value of a gadget computing $\mathrm{MAJ}_3(x_{11}, x_{12}, x_{13})$ instead of x_{12} and x_{13}. This way we suppress the effect of the substitution $x_{11} \leftarrow \rho$, and the resulting circuit C'' computes the initial function $f \diamond \mathrm{MAJ}_3$. Since the majority of three bits can be computed in five gates, we get:

$$\texttt{gates}(C) \leq \texttt{gates}(C'') \leq \texttt{gates}(C|_{x_{11} \leftarrow \rho}) + 5\,.$$

This trick can be extended from 1-substitution to m-substitutions in a natural way. For this, we use gadgets computing the majority of $2m+1$ bits instead of just three bits. We can then suppress the effect of substituting any m variables by feeding the values to $m+1$ of the remaining variables. Taking into account the fact that the majority of $2m+1$ bits can be computed by a circuit of size $4.5(2m+1)$ [8], we get the following result.

▶ **Lemma 1.** *For any $m > 0$, for any function h of n inputs, there exists a function $f = h \diamond \mathrm{MAJ}_{2m+1}$ of $n(2m+1)$ variables, such that*

- *Circuit complexity of f is close to that of h: $\texttt{gates}(h) \leq \texttt{gates}(f) \leq \texttt{gates}(h) + 4.5(2m+1)n$,*
- *For any m-substitution ρ, $\texttt{gates}(f) - \texttt{gates}(f|_\rho) \leq 4.5(2m+1)m$.*

▶ Remark. Note that from the Circuit Hierarchy Theorem (see, e.g., [18]), one can find h of virtually any circuit complexity from n to $2^n/n$.

3.3 Subadditive Measures

In this section we generalize the result of Lemma 1 to *arbitrary* subadditive measures. A function $\mu\colon B_n \to \mathbf{R}$ is called a *subadditive complexity measure*, if for all functions f and g, $\mu(h) \le \mu(f) + \mu(g)$, where $h(\bar{x}, \bar{y}) = f(g(\bar{x}), \ldots, g(\bar{x}), \bar{y})$. That is, if h can be computed by application some function g to some of the the inputs, and then evaluating f, then the measure of h must not exceed the sum of measures of f and g. Clearly, the measures $\mu(f) = \texttt{gates}(f)$ and $\mu_\alpha(f) = \texttt{gates}(f) + \alpha \cdot \texttt{inputs}(f)$ are subadditive, and so are many other natural measures.

Let $f \in B_n$ and $g \in B_k$. Then by $h = f \diamond g$ we denote the function of nk variables resulting from f by replacing each of its input variables by h applied to k fresh variables.

Our main construction is such a composition of a function f (typically, of large circuit complexity) and a gadget g that is chosen to satisfy certain combinatorial properties. Note that since we show a limitation of the proof method rather than a proof of a lower bound, we do not necessarily need to present explicit functions.

In this section we use gadgets that satisfy the following requirement: For every set of variables Y of size m, we can force the value of the gadget to be 0 and 1 by assigning constants only to the remaining variables.

▶ **Definition 2** (weakly m-stable function)**.** A function $g(X)$ is weakly m-stable if, for every $Y \subseteq X$ of size $|Y| \le m$, there exist two assignments $\tau_0, \tau_1\colon X \setminus Y \to \{0,1\}$ to the remaining variables, such that $g|_{\tau_0}(Y) \equiv 0$ and $g|_{\tau_1}(Y) \equiv 1$. That is, after the assignment τ_0 (τ_1), the function does not depend on the remaining variables Y.

It is easy to see that MAJ_{2m+1} is a weakly m-stable function. In Lemma 6 we show that almost all Boolean functions satisfy an even stronger requirement of stability.

▶ **Theorem 3.** *Let μ be a subadditive measure, f be a Boolean function, g be a weakly m-stable function, and $h = f \diamond g$. Then for every m-substitution ρ, $\mu(h) - \mu(h|_\rho) \le m \cdot \mu(g)$.*

Proof. Similarly to Lemma 1, we use a circuit H for the function $h|_\rho$ to construct a circuit C for h. Let

$$h(x_{11}, x_{12}, \ldots, x_{nk}) = f(g(x_{11}, \ldots, x_{1k}), \ldots, g(x_{n1}, \ldots, x_{nk})).$$

Let us focus on the variables $x_{11}, \ldots, x_{1k}$. Assume, without loss of generality, that the variables $x_{11}, \ldots, x_{1r}$ are substituted by ρ. Since ρ is an m-substitution, $r \le m$. From the definition of weakly m-stable function, there exist substitutions τ_0 and τ_1 to the variables $x_{1r+1}, \ldots, x_{1k}$, such that $g|_{\rho\tau_0} = 0$ and $g|_{\rho\tau_1} = 1$. We take the circuit H and add a circuit computing $g(x_{11}, \ldots, x_{1k})$. Now, for every variable $x \in \{x_{1r+1}, \ldots, x_{1k}\}$ in the circuit H, we wire $g(x_{11}, \ldots, x_{1k}) \oplus \tau_0(x)$ instead of x if $\tau_0(x) \ne \tau_1(x)$, and wire $\tau_0(x)$ otherwise. That is, we set $x_{1r+1}, \ldots, x_{1k}$ in such a way that $g|_\rho(x_{1r+1}, \ldots, x_{1k}) = b = g(x_{11}, \ldots, x_{1k})$. Thus, we added one instance of a circuit computing the gadget g and "repaired" $g(x_{11}, \ldots, x_{1k})$.

Now we repeat this procedure for each of the n inner functions g that have at least one variable substituted by ρ. Since ρ is an m-substitution, there are at most m gadgets we need to repair. Thus, we can compute h using the circuit H and m instances of a circuit computing g. From subadditivity of μ, $\mu(h) - \mu(h|_\rho) \le m \cdot \mu(g)$. ◀

3.4 Measures that count inputs

The results of the previous section prove that no subadditive complexity measure can prove a lower bound of more than $n\mu(g)$, where the gadget g depends only on m. For $g = \mathrm{MAJ}_{2m+1}$

and measure $\mu(g) = \texttt{gates}(g)$ Lemma 1 gives $4.5(2m+1)n$ as a specific linear bound barrier that gate elimination cannot overcome. However, since $\mu(g)$ depends on the measure μ, it does not exclude a possibility that there is a sequence of complexity measures allowing to prove better and better bounds. One such natural sequence is based on the circuit measure $\mu_\alpha(C) = \texttt{gates}(C) + \alpha \cdot \texttt{inputs}(C)$ for a constant $\alpha \geq 0$ (used, for example, in [7, 13]). Indeed, for growing α, the method of the previous section gives growing bounds, and if one proves that it is possible to eliminate, say, $c_1 > 0$ gates and $c_2 > 1$ variables per substitution, then after $n - o(n)$ substitutions that would give us $\mu(C) \geq (n - o(n))(c_1 + \alpha c_2) = n(c_1 + \alpha c_2) - o(n)$. This would imply that $\texttt{gates}(C) \geq n(c_1 + \alpha(c_2 - 1)) - o(n)$, an arbitrary linear lower bound. Note that does not require a sequence of gate elimination proofs, just a single proof and a sequence of complexity measures.

In this section in order to show that such a measure cannot prove growing linear bounds, we construct a function f such that any m-substitution reduces the measure by a constant number c_m of gates and at most m inputs. This prevents anyone from proving a better than $c_m n$ bound with it.

▶ **Definition 4** (m-stable function)**.** A function $g(X)$ is m-stable if, for every $Y \subseteq X$ of size $|Y| \leq m+1$ and every $y \in Y$, there exists an assignment $\tau \colon X \setminus Y \to \{0,1\}$ to the remaining variables such that $g|_\tau(Y) \equiv y$ or $g|_\tau(Y) \equiv \neg y$. That is, after the assignment τ, the function depends only on the variable y.

It is now easy to see that every m-stable function is a weakly m-stable function.

▶ **Theorem 5.** *Let f be a Boolean function, g be an m-stable function, and $h = f \diamond g$. Then for every m-substitution ρ, $\mu_\alpha(h) - \mu_\alpha(h|_\rho) \leq m \cdot (\texttt{gates}(g) + \alpha)$.*

Proof. Since g is m-stable, Theorem 3 implies that $\texttt{gates}(h) - \texttt{gates}(h|_\rho) \leq m \cdot \texttt{gates}(g)$. It remains to show that $\texttt{inputs}(h) - \texttt{inputs}(h|_\rho) \leq m$. Thus, it suffices to prove that if f depends on x_i and ρ does not substitute $x_{i,j}$, then $h|_\rho$ depends on $x_{i,j}$. Let

$$h(x_{11}, x_{12}, \ldots, x_{nk}) = f(g(x_{11}, \ldots, x_{1k}), \ldots, g(x_{n1}, \ldots, x_{nk})).$$

Assume f depends on its first input. Since g is not constant, there exists a substitution η to the variables $\{x_{21}, \ldots, x_{2k}, \ldots, x_{n1}, \ldots, x_{nk}\}$ such that $h|_\eta(x_{11}, \ldots, x_{1k})$ is not constant.

Let us consider the variables $x_{11}, \ldots, x_{1k}$. Assume, without loss of generality, that the variables $x_{11}, \ldots, x_{1r}$ are substituted by ρ. Since ρ is an m-substitution, $r \leq m$. Now we want to show that for every $j > r$, $h|_\rho$ depends on x_{1j}. From the definition of an m-stable function, there exists a substitution τ to $\{x_{1,r+1}, \ldots, x_{1k}\} \setminus \{x_{ij}\}$ such that $g|_{\rho\tau}(x_{1j})$ is not constant ($g|_{\rho\tau} = x_{1j}$ or $g|_{\rho\tau} = \neg x_{1j}$). Now, we compose the substitutions η and τ, which gives us that $h|_{\rho\tau\eta}(x_{1j})$ is not constant. This implies that the function $h|_\rho$ depends on the variable x_{1j}. ◀

Now we show that for a fixed m, almost all Boolean functions are m-stable.

▶ **Lemma 6.** *For $m \geq 1$ and $k = \Omega(2^m)$, a random $f \in B_k$ is m-stable almost surely.*

Proof. Let X denote the set of k input variables. Let us fix a set Y, $|Y| \leq m+1$, and a variable $y \in Y$. Now let us count the number of functions that do not satisfy the definition of m-stable function for this fixed choice of Y and y. Thus, for each assignment to the variables from $X \setminus Y$, the function must not be y nor $\neg y$. There are 2^{k-m-1} assignments to the variables $X \setminus Y$, and at most $(2^{2^{m+1}} - 2)$ functions of $(m+1)$ variables that are not y nor $\neg y$. Thus, there are at most $(2^{2^{m+1}} - 2)^{2^{k-m-1}}$ functions that do not satisfy the definition of

m-stable function for this fixed choice of Y and y. Now, since there are $\binom{k}{m+1} \cdot (m+1)$ ways to choose Y and y, the union bound implies that a random function is not m-stable with probability at most

$$\frac{\binom{k}{m+1}(m+1)(2^{2^{m+1}}-2)^{2^{k-m-1}}}{2^{2^k}} \le k^{m+2} \cdot \left(\frac{2^{2^{m+1}}-2}{2^{2^{m+1}}}\right)^{2^{k-m-1}} \le$$
$$\exp\left((m+2)\ln k - 2^{k-m-2^{m+1}}\right) = o(1)$$

for $k = \Omega(2^m)$. ◀

Lemma 6, together with Theorem 5, provides a class of functions such that any m-substitution decreases the measure μ_α by at most a fixed constant (which may depend on m but not on α).

▶ **Corollary 7.** *For any $m > 0$, there exists $k > 0$ and a function g of k inputs, such that for any function h of n inputs, the function $f = h \diamond g$ of nk inputs satisfies:*

- *Circuit complexity of f is close to that of h: $\mathtt{gates}(h) \le \mathtt{gates}(f) \le \mathtt{gates}(h) + \mathtt{gates}(g) \cdot n$,*
- *For any m-substitution ρ and real $\alpha > 0$, $\mu_\alpha(f) - \mu_\alpha(f|_\rho) \le \mathtt{gates}(g) \cdot m + \alpha m$.*

Thus, for many functions gate elimination with m-substitutions and μ_α measures can prove only $O(n)$ lower bounds.

▶ Remark. Although Lemma 6 proves the existence of m-stable functions, their circuit complexities might be large (though constant). To optimize these constants, one can use explicit constructions of m-stable functions. For example, for $m = 1$ one can use an error correcting code $C\colon \{1, \dots, 7\} \to \{0,1\}^8$ with distance 4. Let us define a function $g_C\colon \{0,1\}^8 \to \{0,1\}$ as follows:

1. $g_C(C(i)) = 0$ and $g_C(C(i)^{\oplus i}) = 0$ for all i, where $x^{\oplus i}$ inverts the i-th coordinate of the vector x;
2. $g_C(C(i)^{\oplus j}) = 1$ and $g_C(C(i)^{\oplus i,j}) = 1$ for all $j \ne i$.

It is easy to see that g_C is 1-stable. This construction can also be easily generalized to larger m.

A computer-assisted search gives a 1-stable function of 5 inputs that can be computed with 11 gates, which means that for 1-substitutions one cannot prove a lower bound stronger than $11n$.

4 Known Limitations for Various Circuit Models

Although there is no known argument limiting the power of gate elimination, there are many known barriers in proving circuit lower bounds. In this section we list some of them. This list does not pretend to cover all known barriers in proving lower bounds, but we try to show both fundamental barriers in proving strong bounds and limits of specific techniques.

Baker, Gill, and Solovay [3, 12] present the *relativization* barrier that shows that any solution to the P versus NP question must be non-relativizing. In particular, they show that the classical diagonalization technique is not powerful enough to resolve this question. Aaronson and Wigderson [1] present the *algebrization* barrier that generalizes relativization. For instance, they show that any proof of superlinear circuit lower bound requires non-algebrizing techniques. The *natural proofs* argument by Razborov and Rudich [28] shows that a "natural" proof of a circuit lower bound would contradict the conjecture that strong one-way functions exist. In particular, this argument shows that the *random restrictions* method [14] is unlikely to prove superpolynomial lower bounds. The natural proofs argument implies the following limitation for the gate elimination method. If subexponentially strong one-way functions exist, then for any large class $\mathcal{P}$ of functions (fraction of elements of $\mathcal{P}$ is greater than $\frac{1}{n}$), for any effective measure (computable in time $2^{O(n)}$) and effective family of substitutions $\mathcal{S}$ (the family of substitutions used by the gate elimination algorithm is enumerable in time $2^{O(n)}$), gate elimination cannot prove lower bounds better than $O(n)$. Note that there are currently no known algorithms computing the measures considered in this paper in time $2^{O(n)}$.

Let $\mathcal{F}$ be a family of Boolean functions of n variables. Let X and Y be disjoint sets of input variables, and $|X| = n$. Then a Boolean function $UF(X, Y)$ is called *universal* for the family $\mathcal{F}$ if for every $f(X) \in \mathcal{F}$, there exists an assignment c of constants to the variables Y, such that $UF(X, c) = f(X)$. For example, it can be shown that the function used by Blum [5] is universal for the family $\mathcal{F} = \{x_i \oplus x_j, x_i \wedge x_j | 1 \le i, j \le n\}$. Nigmatullin [23, 24] shows that many known proofs can be stated as lower bounds for universal functions for families of low-complexity functions. At the same time, Valiant [34] proves a linear upper bound on the circuit complexity of universal functions for these simple families.

Vadhan and Williams [33] note that the inequality (1) is tight for the inner product function. This implies that the approach from [7] described in Section 2 cannot yield stronger bounds.

There are known linear upper bounds on circuit complexity of some specific functions and even classes of functions. For example, Demenkov et al. [6] show that each *symmetric function* (i.e., a function that depends only on the sum of its inputs over the integers) can be computed by a circuit of size $4.5n + o(n)$. This, in turn, implies that no gate elimination argument for a class of functions that contains a symmetric function can lead to a superlinear lower bound.

The basis U_2 is the basis of all binary Boolean functions without parity and its negation. The strongest known lower bound for circuits over the basis U_2 is $5n - o(n)$. This bound is proved by Iwama and Morizumi [17] for $(n - o(n))$-*mixed* functions. Amano and Tarui [2] construct an $(n - o(n))$-mixed function whose circuit complexity over U_2 is $5n + o(n)$.

A formula is a circuit where each gate has out-degree one. The best known lower bound of $n^{2-o(1)}$ on formula size is proved by Nechiporuk [21]. The proof of Nechiporuk is based on counting different *subfunctions* of given function. It is known that this argument cannot lead to a superquadratic lower bound (see, e.g., Section 6.5 in [18]).

A De Morgan formula is a formula with AND and OR gates, whose inputs are variables and their negations. The best known lower bound for De Morgan formulas is $n^{3-o(1)}$ (Håstad [15], Tal [32], Dinur and Meir [9]). The original proof of this lower bound by Håstad is based on showing that the shrinkage exponent Γ is at least 2. This cannot be improved since Γ is also at most 2 as can be shown by analyzing the formula size of the parity function.

Paterson introduces the notion of formal complexity measures for proving De Morgan formula size lower bounds (see, e.g., [35]). A formal complexity measure is a function $\mu\colon B_n \to \mathbb{R}$ that maps Boolean functions to reals, such that

1. for every literal x, $\mu(x) \le 1$;
2. for all Boolean functions f and g, $\mu(f \wedge g) \le \mu(f) + \mu(g)$ and $\mu(f \vee g) \le \mu(f) + \mu(g)$.

It is known that De Morgan formula size is the largest formal complexity measure. Thus, in order to prove a lower bound on the size of De Morgan formula, it suffices to define a formal complexity measure and show that an explicit function has high value of measure. Khrapchenko [19] uses this approach to prove an $n^{2-o(1)}$ lower bound on the size of DeMorgan formulas for parity. Unfortunately, many natural classes of formal complexity measures cannot lead to stronger lower bounds. Hrubes et al. [16] prove that *convex* measures (including the measure used by Khrapchenko) cannot lead to superquadratic bounds. A formula complexity measure μ is called *submodular*, if for all functions f, g it satisfies $\mu(f \vee g) + \mu(f \wedge g) \leq \mu(f) + \mu(g)$. Razborov [26] uses a submodular measure based on matrix parameters to prove superpolynomial lower bounds on the size of monotone formulas. In a subsequent work, Razborov [27] shows that submodular measures cannot yield superlinear lower bounds for non-monotone formulas. The *drag-along principle* [28, 20] shows that no useful formal complexity measure can capture specific properties of a function. Namely, it shows that if a function has measure m, then a random function with probability $1/4$ has measure at least $m/4$. Measures based on graph entropy (Newman and Wigderson [22]) are used to prove a lower bound of $n \log n$ on DeMorgan formula size, but it is proved that these measures cannot lead to stronger bounds.

5 Conclusion and Further Directions

In this paper we have demonstrated that there are functions of virtually arbitrary complexity that even after several substitutions do not allow to reduce their complexity more than by a constant number of gates (and at most one variable they depend upon), or a constant amount of a subadditive complexity measure.

This puts a barrier on gate elimination proofs that do not use specific properties of the functions while analyzing how their circuits degrade after substitutions. Indeed, in most proofs it is usually the case (properties of the function are used for estimating *how many substitutions* can the function withstand).

However, there is one exception: in order to estimate the number of “bad” local situations on the top of a circuit computing the function, [11] uses the fact that the function is an affine disperser. While we believe that in this particular case it can be overcome, there may be new techniques exploiting the function properties. Thus the first open question is:

- *Show that interesting classes of functions contain functions resistant to gate elimination. For example, it would be interesting to show that the class of affine dispersers, or more generally every large enough class of functions, contains a series of functions resistant to gate elimination.*

Another possible direction is to extend the result to other possible complexity measures, because some syntactic measures can lack subadditivity (for example, composition can in principle introduce more “bad” local situations). One can imagine, for example, “local” measures that count specific small patterns in a circuit.

- *Extend the result to local complexity measures or another wide class.*

While the results of this paper capture all types of substitutions, another possible directions is:

- *Allow induction to descend to arbitrary varieties instead of the varieties described by substitutions (for example, allow restrictions of the form $xy = zt$).*

The situation might become much easier if we switch from arbitrary Boolean functions to n-bit linear maps $\{0,1\}^n \to \{0,1\}^n$. They have non-linear complexity in principle but, again, we do not have non-linear lower bounds for explicit functions. Can gate elimination prove non-linear bounds here? What if we restrict ourselves to linear operations in the circuit and linear substitutions? The gadgets used in this paper are non-linear and thus cannot help.

- *Extend the result to linear maps.*

We show that there exist functions such that after a constant number of substitutions the complexity of these functions decreases only by a constant. How far can it be strengthened w.r.t. the number of substitutions?

- *Does there exist a function f of nonlinear complexity such that after $m = \Omega(n)$ substitutions its circuit complexity drops by $O(m)$ gates only?*

References

1 Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Transactions on Computation Theory (TOCT)*, 1(1):2, 2009.

2 Kazuyuki Amano and Jun Tarui. A well-mixed function with circuit complexity $5n \pm o(n)$: Tightness of the Lachish–Raz-type bounds. In *Proceedings of Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 342–350. Springer, 2008.

3 Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} = ?\mathcal{NP}$ question. *SIAM Journal on computing*, 4(4):431–442, 1975.

4 Eli Ben-Sasson and Swastik Kopparty. Affine dispersers from subspace polynomials. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 65–74, 2009.

5 Norbert Blum. A boolean function requiring $3n$ network size. *Theor. Comput. Sci.*, 28:337–345, 1984.

6 Evgeny Demenkov, Arist Kojevnikov, Alexander S. Kulikov, and Grigory Yaroslavtsev. New upper bounds on the Boolean circuit complexity of symmetric functions. *Information Processing Letters*, 110(7):264–267, 2010.

7 Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In *Proceedings of International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 256–265, 2011.

8 Evgeny Demenkov and Alexander S. Kulikov. Computing All MOD-Functions Simultaneously. *Computer Science – Theory and Applications*, pages 81–88, 2012.

9 Irit Dinur and Or Meir. Toward the krw composition conjecture: Cubic formula lower bounds via communication complexity. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 50. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

10 Paul E. Dunne. *Techniques for the analysis of monotone Boolean networks.* PhD thesis, University of Warwick, 1984.

11 Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. A better-than-$3n$ lower bound for the circuit complexity of an explicit function. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:166, 2015.

12 Lance Fortnow. The role of relativization in complexity theory. *Bulletin of the EATCS*, pages 1–15, 1994.

13 Alexander Golovnev and Alexander S. Kulikov. Weighted gate elimination: Boolean dispersers for quadratic varieties imply improved circuit lower bounds. In *Innovations in Theoretical Computer Science, ITCS '16*, pages 405–411, 2016.

14 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

15 Johan Håstad. The shrinkage exponent is 2. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 114–123. IEEE, 1993.

16 Pavel Hrubeš, Stasys Jukna, Alexander Kulikov, and Pavel Pudlak. On convex complexity measures. *Theoretical Computer Science*, 411(16):1842–1854, 2010.

17 Kazuo Iwama and Hiroki Morizumi. An Explicit Lower Bound of $5n - o(n)$ for Boolean Circuits. In *Proceedings of International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2002.

18 Stasys Jukna. *Boolean function complexity: advances and frontiers*, volume 27. Springer Science & Business Media, 2012.

19 Valeriy M. Khrapchenko. Method of determining lower bounds for the complexity of P-schemes. *Mathematical Notes*, 10(1):474–479, 1971.

20 Richard J. Lipton. *The $\mathcal{P} = \mathcal{NP}$ Question and Gödel's Lost Letter.* Springer Science & Business Media, 2010.

21 Edward I. Nechiporuk. On a Boolean function. *Doklady Akademii Nauk. SSSR*, 169(4):765–766, 1966.

22 Ilan Newman and Avi Wigderson. Lower bounds on formula size of boolean functions using hypergraph entropy. *SIAM Journal on Discrete Mathematics*, 8(4):536–542, 1995.

23 Roshal G. Nigmatullin. Are lower bounds on the complexity lower bounds for universal circuits? In *Fundamentals of Computation Theory*, pages 331–340. Springer, 1985.

24 Roshal G. Nigmatullin. *Complexity lower bounds and complexity of universal circuits.* Kazan University, 1990.

25 Wolfgang J. Paul. A $2.5n$-lower bound on the combinational complexity of boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977.

26 Alexander A. Razborov. Applications of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1):81–93, 1990.

27 Alexander A. Razborov. On submodular complexity measures. In *Poceedings of the London Mathematical Society Symposium on Boolean Function Complexity*, pages 76–83, New York, NY, USA, 1992. Cambridge University Press.

28 Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55(1):24–35, 1997.

29 Claus-Peter Schnorr. Zwei lineare untere schranken für die komplexität boolescher funktionen. *Computing*, 13(2):155–171, 1974.

30 Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.

31 Larry J. Stockmeyer. On the combinational complexity of certain symmetric boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.

32 Avishay Tal. Shrinkage of De Morgan formulae by spectral techniques. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 551–560. IEEE, 2014.

33 Salil Vadhan and Ryan Williams. Personal communication, 2013.

34 Leslie G. Valiant. Universal circuits (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 196–203. ACM, 1976.

35 Ingo Wegener. *The complexity of Boolean functions.* Wiley-Teubner, 1987.

Algebraic Problems Equivalent to Beating Exponent 3/2 for Polynomial Factorization over Finite Fields [*]

Zeyu Guo[1], Anand Kumar Narayanan[2], and Chris Umans[3]

1 **Computing and Mathematical Sciences, California Institute of Technology**
`zguo@caltech.edu`
2 **Computing and Mathematical Sciences, California Institute of Technology**
`anandkn@caltech.edu`
3 **Computing and Mathematical Sciences, California Institute of Technology**
`umans@caltech.edu`

Abstract

The fastest known algorithm for factoring univariate polynomials over finite fields is the Kedlaya-Umans [13] (fast modular composition) implementation of the Kaltofen-Shoup algorithm [12, § 2]. It is randomized and takes $\widetilde{O}(n^{3/2}\log q + n\log^2 q)$ time to factor polynomials of degree n over the finite field $\mathbb{F}_q$ with q elements. A significant open problem is if the 3/2 exponent can be improved. We study a collection of algebraic problems and establish a web of reductions between them. A consequence is that an algorithm for any one of these problems with exponent better than 3/2 would yield an algorithm for polynomial factorization with exponent better than 3/2.

1998 ACM Subject Classification F.2.1 Computations in Finite Fields

Keywords and phrases Algorithms, Complexity, Finite Fields, Polynomials, Factorization.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.47

1 Introduction

A recent trend in discrete algorithms has been to establish very efficient reductions between problems with polynomial time algorithms, with the intention of identifying barriers (conceptual or concrete) to improving the polynomial running time of the best known algorithms. A standard example is the problem 3-SUM, which seems to require essentially quadratic time, and which has been reduced to many other problems. More recently, the study of "fine-grained" complexity has broadened, with several connections established between central problems in discrete algorithms, and new conjectures beyond the 3-SUM conjecture entering the picture (see, e.g. [1, 2, 3, 17, 18, 22, 23, 24]).

In this paper we focus on a "barrier" in *algebraic* algorithms, that of improving the exponent 3/2 for univariate polynomial factorization and several other problems. Generally, algebraic problems have two relevant "size" parameters – n, and the field size q. It is typical for the dependence on q to be polylogarithmic (it is for all of the problems we consider), and so we focus on the exponent on n in this work. We find that exponent 3/2 seems to be a barrier for a number of problems. This points to a need to move beyond the so-called "baby steps giant steps" methodology which tends to give rise to the exponent 3/2 behavior.

* The authors were supported by NSF grant CCF 1423544 and a Simons Foundation Investigator grant.

The reductions in this paper can be seen as giving evidence that improving the 3/2 exponent may not be possible for these problems, but we believe that it "merely" gives evidence that this improvement requires a conceptual breakthrough (along the lines of going beyond the baby-steps giant-steps approach). Using the connections established in this paper, such a breakthrough for any one of the problems considered here would improve the exponent for all of them. In the discussion below, we use $\widetilde{O}$ to suppress $n^{o(1)}$ terms and $\log^{o(1)} q$ terms, in order to highlight the exponent on n that is our main object of study. We also use the phrase "nearly linear time reduction" to mean a reduction that runs in time $\widetilde{O}(n \log q)$, and the phrase "3/2 exponent reducible" to mean the weaker connection that shows that beating exponent 3/2 for one problem implies beating exponent 3/2 for the other.

1.1 Algebraic problems with 3/2 exponent algorithms

We investigate the complexity of factoring a univariate polynomial over a finite field into its irreducible factors. The problem formally stated is,

- FACTOR: *Given a monic square free $f(x) \in \mathbb{F}_q[x]$ of degree n, write $f(x)$ as a product of its monic irreducible factors.*

The square free assumption is without loss of generality [14, 25]. FACTOR can be solved in randomized polynomial time [4] and there is an extensive line of research [5, 12, 21] leading to a randomized algorithm [13] with exponent 3/2. Surprisingly, even determining the *degree* of a single irreducible factor rapidly would be sufficient to improve the exponent of this algorithm. We formulate this problem as

- FACTOR DEGREE: *Given a monic square free $f(x) \in \mathbb{F}_q[x]$, find the degree of an irreducible factor of $f(x)$.*

and prove in § 2 that FACTOR is 3/2-exponent reducible to FACTOR DEGREE. That is, an algorithm for FACTOR DEGREE with exponent less than 3/2 yields one for FACTOR. Observe that FACTOR DEGREE merely seeks one, not necessarily all, irreducible factor degrees. We next investigate two linear algebraic problems, both we will demonstrate to be nearly linear time reducible to FACTOR.

- FROBENIUS MIN-POLY: *Given a monic square free $f(x) \in \mathbb{F}_q[x]$, compute the minimal polynomial of the Frobenius endomorphism on $\mathbb{F}_q[x]/(f(x))$ which takes $a(x) \mod f(x)$ to $a(x)^q \mod f(x)$.*
- CARLITZ CHAR-POLY: *Given a monic square free $f(x) \in \mathbb{F}_q[x]$, compute the characteristic polynomial of the Carlitz endomorphism on $\mathbb{F}_q[x]/(f(x))$ which takes $a(x) \mod f(x)$ to $xa(x) + a(x)^q \mod f(x)$.*

In § 4, we prove that FACTOR DEGREE is nearly linear time reducible to CARLITZ CHAR-POLY, under certain restrictions on the characteristic of $\mathbb{F}_q$. These restrictions were removed in [15] by passing from Carlitz to Drinfeld modules. In § 3, through a novel recursive argument, we prove that FACTOR is 3/2-exponent reducible to FROBENIUS MIN-POLY.

FROBENIUS MIN-POLY was known [12, 11] to be nearly linear time reducible to

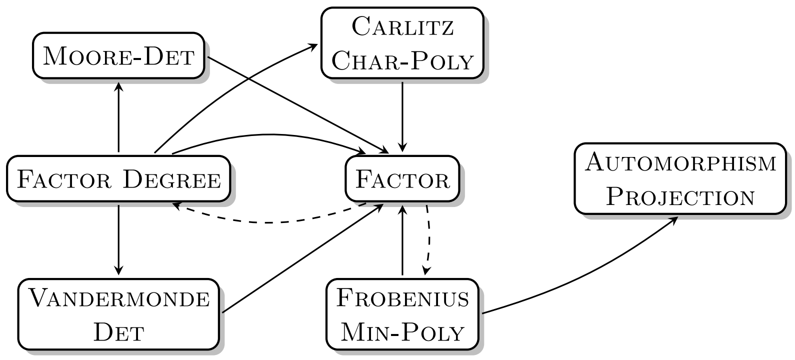
- AUTOMORPHISM PROJECTION: *Given a monic square free $f(x) \in \mathbb{F}_q[x]$, $\alpha \in \mathbb{F}_q[x]/(f(x))$ and an $\mathbb{F}_q$-linear map $u : \mathbb{F}_q[x]/(f(x)) \longrightarrow \mathbb{F}_q$, compute $u(\alpha^{q^i}), \forall i \in \{1, 2, \dots, \deg(f)\}$.*

Thus, as a consequence of the reduction in § 3, we conclude that FACTOR is 3/2-exponent reducible to AUTOMORPHISM PROJECTION. This should be contrasted with the connection established in [12, 11]. They show that FACTOR is nearly linear time reducible to AUTOMORPHISM PROJECTION *assuming an $\mathbb{F}_q$-linear straight line program algorithm* for AUTOMORPHISM PROJECTION. This assumption allows them to use the "transpose" problem AUTOMORPHISM EVALUATION. Our reduction to AUTOMORPHISM PROJECTION is novel, direct, and holds without any assumptions.

The final two problems pertain to zero testing Moore and Vandermonde determinants.

- MOORE-DET: *Given a monic square free $f(x) \in \mathbb{F}_q[x]$ and a positive integer m, decide if the determinant of the m by m square matrix with entries $m_{ij} := x^{jq^i} \mod f(x)$ is zero.*
- VANDERMONDE-DET: *Given a monic square free $f(x) \in \mathbb{F}_q[x]$ and a positive integer $b \leq \sqrt{\deg(f)}$, decide if the determinant of the Vandermonde matrix with first row $(x^{q^i} \mod f(x), i = 0, 1, 2, \ldots, b-1, b, 2b, 3b, \ldots, (b-1)b, b^2)$ is zero.*

In § 5, we prove that FACTOR DEGREE is nearly linear time reducible to each of these problems and that each of these problems is nearly linear time reducible to FACTOR. In summary, we have the following diagram where solid lines denote nearly linear time reductions and dotted lines denote 3/2-exponent reductions.



An interesting open question is if the dotted lines can be made solid. Except for AUTOMORPHISM PROJECTION, every listed problem has a known randomized algorithm with exponent 3/2. If the matrix multiplication exponent is 2, then a randomized algorithm for AUTOMORPHISM PROJECTION with exponent 3/2 is known. Another open problem is if this dependence on the matrix multiplication exponent can be removed – perhaps by reducing AUTOMORPHISM PROJECTION to one of the other problems in the figure. Regardless, an algorithm for any of the problems in the figure with exponent less that 3/2 would yield an algorithm with exponent 3/2 for FACTOR, and this is one of the main points of this paper.

2 Factorization and Finding a Factor Degree

Clearly, if one can solve the problem FACTOR in time $T(n, q)$ then one can solve the problem FACTOR DEGREE in time $T(n, q)$. In this section we show a reduction in the reverse direction, which leads to the surprising conclusion that one *only* needs to compute the *degree* of a single irreducible factor of the polynomial $f(x)$ with exponent better than 3/2 to be able to factor $f(x)$ completely with exponent better than 3/2.

▶ **Theorem 1.** *If there is an algorithm that solves* FACTOR DEGREE *in the time $T(n, q)$ where $T(n, q) = \Omega(n \log^2 q)$* [1]*, then there is an algorithm that solves* FACTOR *in time $\widetilde{O}(n \cdot T(n, q)^{1/3} \log^{4/3} q)$.*

Observe that when FACTOR DEGREE has an exponent 3/2 algorithm (as it does), this reduction recovers a 3/2 exponent algorithm for FACTOR. A sub-3/2 exponent algorithm for FACTOR DEGREE implies a sub-3/2 exponent algorithm for FACTOR, with a nearly-linear time algorithm yielding exponent 4/3 for FACTOR.

[1] The assumption $T(n, q) = \Omega(n \log^2 q)$ is without loss of generality. For otherwise we slow down an algorithm with runtime $T(n, q)$ until it is $\Omega(n \log^2 q)$.

Proof. We are given a monic, square-free polynomial $f(x) \in \mathbb{F}_q[x]$ of degree n. Let $g(x)$ be the product of irreducible factors of $f(x)$ with degrees at most t (for a parameter t to be chosen later). If $s(x)$ is defined as $s(x) = \prod_{i=1}^{t}(x^{q^i} - x)^{a_i}$, for some positive integers $a_1, a_2, \ldots, a_t$, then we have that $g(x) = \gcd(s(x), f(x))$. Using fast modular composition [13] and the method of Kaltofen-Shoup [12], we can compute $s(x) \bmod f(x)$ in time $\widetilde{O}(n\sqrt{t}\log^2 q)$ time. We then proceed to factor $g(x)$ completely, using the Kedlaya-Umans implementation of the Kaltofen-Shoup algorithm. The bottleneck in this algorithm is computing the splitting polynomials, which are all polynomials of the form of $s(x)$, with i ranging from 1 up to $t' \le t$. This portion of the algorithm runs in time $\widetilde{O}(n\sqrt{t}\log^2 q)$ and factors $g(x)$ completely.

Now we invoke the algorithm to solve FACTOR DEGREE, on input $f(x)/g(x)$. Upon finding the degree d of an irreducible factor, we compute $\gcd(x^{q^d} - x \bmod f(x), f(x))$ to split off the factors with that degree. We then repeat. The number of repetitions is bounded by n/t, since each irreducible factor of $f(x)/g(x)$ has degree at least t. Each repetition takes time $T(n,q) + \widetilde{O}(n\log^2 q)$. Thus this portion of the algorithm runs in time $\widetilde{O}(n/t\cdot(T(n,q)+n\log^2 q)) = \widetilde{O}(n/t\cdot T(n,q))$. Finally we factor completely using equal-degree factorization which takes $\widetilde{O}(n\log^2 q)$ time. Optimizing, we set $t = (T(n,q)/\log^2 q)^{2/3}$, and the overall running time becomes $\widetilde{O}(n \cdot T(n,q)^{1/3}\log^{4/3} q)$ for each of the two stages, and hence in total as well. ◀

3 Factoring and Minimal Polynomial of Frobenius

For a monic square free $f(x)$, let $g(\lambda) \in \mathbb{F}_q[\lambda]$ denote the minimal polynomial of the q^{th} power Frobenius endomorphism $\sigma : \mathbb{F}_q[x]/(f(x)) \to \mathbb{F}_q[x]/(f(x))$. That is, $g(\lambda)$ is the unique nonzero monic polynomial of least degree such that the endomorphism $g(\sigma)$ on $\mathbb{F}_q[x]/(f(x))$ is zero. The problem FROBENIUS MIN-POLY is to determine $g(\lambda)$ given $f(x)$. Since $g(\lambda)$ is the least common multiple of $\lambda^d - 1$ as d runs through the degrees of the irreducible factors of $f(x)$, FROBENIUS MIN-POLY is nearly linear time reducible to FACTOR. In this section, we conversely prove that FACTOR is 3/2-exponent reducible to FROBENIUS MIN-POLY.

Let `FrobMinPoly` be an oracle that solves FROBENIUS MIN-POLY. We present an algorithm `Factor` that invokes `FrobMinPoly` and solves FACTOR. For $k \in \mathbb{N}^+$, denote by Φ_k the kth cyclotomic polynomial over $\mathbb{F}_q$. Write $\phi(\cdot)$ for the Euler totient function.

Algorithm 1 `Factor`$(f(x))$

```
Input: Monic square free polynomial $f(x) \in \mathbb{F}_q[x]$ of degree $n$.
Output: Monic irreducible factors of $f(x)$.
Oracle: FrobMinPoly
1: Using [13], output and remove all monic irreducible factors of $f(x)$ of degree at most
   $n^{2/3}$. If at most one irreducible factor of degree greater than $n^{2/3}$ remains, output and
   exit.
2: $g(\lambda) \leftarrow$ FrobMinPoly$(f(x))$.
3: Perform square free factorization on $g(\lambda)$, and then run Factor recursively on the outputs
   to obtain the list of monic irreducible factors $g_1(\lambda), \ldots, g_m(\lambda)$ of $g(\lambda)$.
4: Run FindT$(g_1(\lambda), \ldots, g_m(\lambda))$ computing the set $T := \{k : p \nmid k \text{ and } \Phi_k(\lambda) | g(\lambda)\}$ as well
   as $m_k$, the multiplicity of $\Phi_k(\lambda)$ in $g(\lambda)$, for each $k \in T$.
5: Compute $S := \{kp^e : k \in T, 0 \le e \le \log_p m_k\}$.
6: for each $s \in S$ greater than $n^{2/3}$, set $f_s(x) \leftarrow \gcd(f(x), x^{q^s} - x \bmod f(x))$, $f(x) \leftarrow$
   $f(x)/f_s(x)$ and perform equal-degree factorization on $f_s(x)$.
```

The algorithm begins by extracting all monic irreducible factors of degree at most $n^{2/3}$. After Line 1, $f(x)$ only has large (at least $n^{2/3}$) degree factors. Suppose $d_1, d_2, \ldots, d_m$ are the degrees of the (remaining) monic irreducible factors of $f(x)$. Then the minimal polynomial $g(\lambda) \in \mathbb{F}_q[\lambda]$ of the Frobenius acting on $\mathbb{F}_q[x]/(f(x))$ is

$$g(\lambda) = \operatorname{lcm}\left(\lambda^{d_1} - 1, \ldots, \lambda^{d_m} - 1\right).$$

In particular, the cyclotomic polynomials $\Phi_{d_1}(\lambda), \ldots, \Phi_{d_m}(\lambda)$ divide $g(\lambda)$ and the factorization of $g(\lambda)$ contains information about $d_1, d_2, \ldots, d_m$. We devise a novel procedure to infer $d_1, d_2, \ldots, d_m$ efficiently.

On Line 2, $g(\lambda)$ is computed by invoking `FrobMinPoly`. To infer $d_1, d_2, \ldots, d_m$, we seek the factorization of $g(\lambda)$. To this end, a key idea is to factor $g(\lambda)$ recursively on Line 3 and obtain a list $g_1(\lambda), g_2(\lambda), \ldots, g_m(\lambda)$ of its monic irreducible factors. Since $f(x)$ is not irreducible at this point, $g(\lambda)$ has degree strictly less than $f(x)$ and the algorithm runs to completion.

Then we use a procedure `FindT`$(g_1(\lambda), \ldots, g_m(\lambda))$ to compute the set T and integers m_k as defined on Line 4. This step is the most technical part of the algorithm, and we defer its description and analysis to the next subsection, where we prove the following theorem:

▶ **Theorem 2.** `FindT`$(g_1(\lambda), \ldots, g_m(\lambda))$ *can be implemented to run in* $\widetilde{O}(n \log q)$ *time.*

Once T is known, to compute S on Line 5 is straightforward. The following lemma shows that S indeed contains $d_1, d_2, \ldots, d_m$.

▶ **Lemma 3.** $d_1, d_2, \ldots, d_m \in S$.

Proof. Consider an arbitrary $d \in \{d_1, d_2, \ldots, d_m\}$ and write it as $d = kp^e$ with $p \nmid k$. By definition $(\lambda^d - 1) | g(\lambda)$. Since $\lambda^d - 1 = (\lambda^k - 1)^{p^e}$ and $\lambda^k - 1 = \prod_{k_0 | k} \Phi_{k_0}(\lambda)$, $\Phi_k(\lambda)$ is a factor of $g(\lambda)$ with multiplicity at least p^e. So $k \in T$ and $m_k \geq p^e$, implying $d = kp^e \in S$. ◀

To conclude, by Line 6, all the irreducible factors of $f(x)$ are indeed output.

▶ **Theorem 4.** *Suppose the oracle* `FrobMinPoly` *runs in time* $T(n, q)$ *which is monotone in* n *and* q*. Then* `Factor` *factors a degree-*n *polynomial in* $\widetilde{O}(T(n, q) + n^{4/3} \log^2 q)$ *time.*

Proof. We first analyze the running time of each step except the recursive call. Line 1 can be implemented in $\widetilde{O}(n^{4/3} \log^2 q)$ time using the baby-step-giant-step strategy [12, 13]. The oracle `FrobMinPoly` on Line 2 runs in time $T(n, q)$. The set T on Line 4 could be found in time $\widetilde{O}(n \log q)$ by Theorem 2. Since $\prod_{k \in T} \Phi_k(\lambda)^{m_k}$ divides $g(\lambda)$, we have $\sum_{k \in T} m_k \phi(k) \leq \deg(g(\lambda)) \leq n$. Hence $|T| \leq n$ and $m_k \leq n$ for all $k \in T$, implying S on Line 5 could be computed in time $\widetilde{O}(n)$. Further,

$$\sum_{s \in S} s \leq \sum_{k \in T, 0 \leq e \leq \log_p m_k} kp^e \leq \log n \sum_{k \in T} k m_k \leq O(\log \log n) \cdot \log n \sum_{k \in T} m_k \phi(k) = \widetilde{O}(n)$$

where we use $k/\phi(k) = O(\log \log k)$ [19] and $\sum_{k \in T} m_k \phi(k) \leq n$. Hence the number of $s \in S$ greater than $n^{2/3}$ is at most $(\sum_{s \in S} s)/n^{2/3} = \widetilde{O}(n^{1/3})$. For each $s \in S$, Computing $f_s(x)$ takes $\widetilde{O}(n \log^2 q)$ time for each $s \in S$ [13] and hence $\widetilde{O}(n^{4/3} \log^2 q)$ time in total. Equal degree factorization on Line 6 takes $\widetilde{O}(n \log^2 q)$ time in total.

Let $d_{\max}(f(x))$ denote the maximal degree of the irreducible factors of $f(x)$. We claim that $d_{\max}(f(x))$ shrinks by at least a factor of two every two recursive calls. It implies that the recursive tree has depth no more than $O(\log n)$, so the total running time is bounded by $O(\log n) \cdot (T(n, q) + \widetilde{O}(n^{4/3} \log^2 q)) = \widetilde{O}(T(n, q) + n^{4/3} \log^2 q)$, as desired.

Consider an irreducible factor $g_0(\lambda)$ of $g(\lambda)$. We know $g_0(\lambda)$ divides $\lambda^k - 1 = \prod_{k_0|k} \Phi_{k_0}(\lambda)$ for a positive integer k corresponding to some degree k irreducible factor $f_0(x)$ of $f(x)$. If $g_0(\lambda)$ divides $\Phi_{k_0}(\lambda)$ for some proper divisor k_0 of k, we have $\deg(g_0(\lambda)) \le \phi(k_0) \le k_0 \le k/2$. Likewise, if $g_0(\lambda)$ is a proper irreducible factor of $\Phi_k(\lambda)$, we have $\deg(g_0(\lambda)) \le \phi(k)/2 \le k/2$ as well. So assume $g_0(\lambda) = \Phi_k(\lambda)$. Suppose $k = \prod_\ell \ell^{e_\ell}$, ℓ running over prime divisors of k. Then $\phi(k) = \prod_\ell (\ell - 1)\ell^{e_\ell - 1}$. If k is even, we have $e_2 \ge 1$ implying $\deg(g_0(\lambda)) = \phi(k) \le k/2$ (since for $\ell = 2$, $(\ell - 1)\ell^{e_\ell - 1} = \ell^{e_\ell}/2$). If k is odd, $\deg(g_0(\lambda)) = \phi(k) = \prod_\ell (\ell - 1)\ell^{e_\ell - 1}$ is even. The argument above applied to $g_0(\lambda)$ and $g(\lambda)$ in place of $f_0(x)$ and $f(x)$ shows that the degree shrinks by at least a factor of two in the next recursive call. The claim follows. ◀

▶ Remark. One may easily check that the same algorithm and analysis also work if the polynomial $g(\lambda)$ computed by the oracle is the *characteristic polynomial* of the Frobenius endomorphism instead of the minimal polynomial. The only difference is that $g(\lambda)$ is the product of $\lambda^{d_1} - 1, \ldots, \lambda^{d_m} - 1$ rather than their lcm.

3.1 Computing the Set T

We next devise a nearly linear time procedure to implement `FindT`. It relies on solutions to the following two problems: (1) finding all irreducible factors of $\Phi_k(\lambda)$ over $\mathbb{F}_q$ from a single irreducible factor $g_0(\lambda)$ and (2) finding the corresponding integer k. We deal with these two problems individually before describing `FindT`.

3.1.1 Finding the irreducible factors of $\Phi_k(\lambda)$

Let $k \in [1, n]$ be an integer coprime to p. Our goal is to find all the irreducible factors of $\Phi_k(\lambda)$ over $\mathbb{F}_q$ from a single irreducible factor $g_0(\lambda)|\Phi_k(\lambda)$. To achieve it, we need to know how $\Phi_k(\lambda)$ factorizes over $\mathbb{F}_q$.

3.1.1.1 Factorization of $\Phi_k(\lambda)$ over $\mathbb{F}_q$.

As k is coprime to p, there are $\phi(k)$ distinct primitive kth roots of unity in $\overline{\mathbb{F}}_q$ which are exactly the roots of $\Phi_k(\lambda)$. Denote this set of roots by μ_k. Let G be the abelian group $(\mathbb{Z}/k\mathbb{Z})^\times$ of order $\phi(k)$. For $d \in \mathbb{Z}$, we write $\bar{d}$ for the image of d in $\mathbb{Z}/k\mathbb{Z}$. The group G acts on μ_k such that $\bar{d} \in G$ sends any $\theta \in \mu_k$ to θ^d. This is a regular action, meaning that for fixed $\theta \in \mu_k$, the map $\bar{d} \mapsto \theta^d$ is a bijection between G and μ_k. As p is coprime to k, we have $\bar{q} \in G$. Let $G_0 = \langle \bar{q} \rangle \subseteq G$ and $s = [G : G_0]$. Restrict the G-action on μ_k to a G_0-action. Then μ_k is partitioned into s distinct G_0-orbits represented by $\theta_1, \ldots, \theta_s \in \mu_k$. It is well-known that the factorization of $\Phi_k(\lambda)$ over $\mathbb{F}_q$ is then determined in the following way:

▶ **Lemma 5.** *Under the notations above, $\Phi_k(\lambda)$ has s irreducible factors $g_1(\lambda), \ldots, g_s(\lambda)$ over $\mathbb{F}_q$ corresponding to the G_0-orbits $G_0\theta_1, \ldots, G_0\theta_s$ of μ_k in the sense that the set of roots of $g_i(\lambda)$ is exactly $G_0\theta_i$.*

Proof. Let $g(\lambda)$ be an irreducible factor of $\Phi_k(\lambda)$ over $\mathbb{F}_q$ and $\theta \in \mu_k$ be a root of $g(\lambda)$. Then $\mathbb{F}_q[\theta]$ is Galois over $\mathbb{F}_q$ with the Galois group generated by the Frobenius map $a \mapsto a^q$. So $a \in \mathbb{F}_q[\theta]$ is a root of $g(\lambda)$ if and only if a^q is a root of $g(\lambda)$. Therefore $G_0\theta$ is the set of roots of $g(\lambda)$ and the lemma follows. ◀

From now on we fix a root $\theta \in \mu_k$ of the given irreducible factor g_0 of Φ_k. For any subgroup $H \subseteq G$ containing G_0, the G-action on μ_k restricts to an H-action. The H-orbit $H\theta$ is partitioned into a disjoint union of G_0-orbits and hence corresponds to a subset L of irreducible factors of $\Phi_k(\lambda)$ by Lemma 5. Note that L also determines H: $h \in G$ lies in H if and only if the minimal polynomial of $h\theta$ over $\mathbb{F}_q$ is in L. We say L is *associated with* the subgroup H.

We need an auxiliary procedure `FindOrder`(ℓ, L) to find the order of $H\bar{\ell}$ in G/H.

▶ **Lemma 6.** *There exists a procedure* `FindOrder`(ℓ, L) *that takes an integer ℓ and the set L associated with H, and returns the following result: if $\bar{\ell} \in G = (\mathbb{Z}/k\mathbb{Z})^\times$, it returns the order of $H\bar{\ell}$ in G/H, i.e. the smallest $e > 1$ for which $\bar{\ell}^e \in H$. Otherwise it returns zero. Moreover* `FindOrder`(ℓ, L) *could be implemented in time $\widetilde{O}(\phi(k)\log q)$.*

See the full version of this paper [10] for the proof of Lemma 6 and the description of `FindOrder`(ℓ, L).

We use a randomized procedure `FindCyclotomic`$(g_0(\lambda), n)$ to find all irreducible factors of $\Phi_k(\lambda)$ over $\mathbb{F}_q$. Here $g_0(\lambda)$ is one irreducible factor of $\Phi_k(\lambda)$ and n is the degree of the polynomial $f(x)$.[2]

Algorithm 2 `FindCyclotomic`$(g_0(\lambda), n)$

Input: Irreducible factor $g_0(\lambda)$ of $\Phi_k(\lambda)$ over $\mathbb{F}_q$ and degree n of $f(x)$
Output: The list of irreducible factors of $\Phi_k(\lambda)$ over $\mathbb{F}_q$

```
1: $L \leftarrow \{g_0(\lambda)\}$
2: for $t$ from 1 to $N = \lfloor c \log n \log\log n \rfloor$ do        ▷ $c > 0$ is a large enough constant
3:     Pick an integer $\ell \in [1, n]$ at random
4:     $e \leftarrow$ FindOrder$(\ell, L)$
5:     for each $h(\lambda) \in L$ do
6:         $r_0 \leftarrow \lambda \bmod h(\lambda) \in \mathbb{F}_q[\lambda]/(h(\lambda))$, $r_i \leftarrow r_{i-1}^{\ell}$ for $i = 1, \ldots, e-1$
7:         Let $f_i(\lambda)$ be the minimal polynomial of $r_i$ over $\mathbb{F}_q$ for $i = 1, \ldots, e-1$
8:         Add $f_1(\lambda), \ldots, f_{e-1}(\lambda)$ to $L$
9:     end for
10: end for
11: return $L$
```

The procedure `FindCyclotomic`$(g_0(\lambda), n)$ maintains a subset L of irreducible factors of $\Phi_k(\lambda)$ associated with some subgroup of G containing G_0. Initially $L = \{g_0(\lambda)\}$, associated with $H_0 := G_0$. We claim:

▶ **Lemma 7.** *Suppose L is associated with H_{i-1} at the beginning ith execution of the outer loop of* `FindCyclotomic`$(g_0(\lambda), n)$*. Then at the end of the ith execution, the set L is associated with a subgroup $H_i \supseteq H_{i-1}$. Moreover, $H_i = H_{i-1}$ if $\bar{\ell} \notin G$ in the ith execution of the outer loop. Otherwise $H_i = H_{i-1}\langle\bar{\ell}\rangle$.*

Proof. If $\ell \notin G$ in the ith execution of the outer loop, then e is set to zero by Lemma 6 and the claim is trivial. So assume $\ell \in G$ and let $H = H_{i-1}\langle\bar{\ell}\rangle$. Then e is the order of $H_{i-1}\bar{\ell}$ in H/H_{i-1} by Lemma 6, or $[H : H_{i-1}]$. Suppose the irreducible factors in L at the beginning of the ith execution correspond to distinct G_0-orbits $G_0\theta_1, \ldots, G_0\theta_m$ whose union is the H_{i-1}-orbit $H_{i-1}\theta$, $m = [H_{i-1} : G_0]$. The inner loop enumerates $G_0\theta_j$, and for each of them, adds the irreducible factor corresponding to $G_0\theta_j^{\ell^s}$ to L, $s = 1, \ldots, e-1$. Note that the union of these G_0-orbits $G_0\theta_j^{\ell^s} = G_0\bar{\ell}^s\theta_j = \bar{\ell}^s G_0\theta_j$ where $1 \le j \le m$, $0 \le s \le e-1$ equals

[2] The argument n is only used on Line 2 and 3 to control the number of repetitions and the range of ℓ, which is related to the error probability.

the union of H_{i-1}-orbits $\bar{\ell}^s H_{i-1}\theta_j$, which equals the H-orbit $H\theta$. And these G_0-orbits are all distinct since the number of them is $me = [H : G_0]$. So L is associated with H at the end of the ith execution of the outer loop. ◀

▶ **Lemma 8.** *The procedure* `FindCyclotomic`$(g_0(\lambda), n)$ *returns a set* L *associated with* $H_N \subseteq G$*. And* $H_N = G$ *with probability* $1 - \mathrm{poly}(n)$ *in which case* L *contains all irreducible factors of* $\Phi_k(\lambda)$ *over* $\mathbb{F}_q$*. Moreover* `FindCyclotomic`$(g_0(\lambda), n)$ *could be implemented in time* $\widetilde{O}(\phi(k) \log q)$*.*

Proof. We want to show $H_N = G$ with probability $1 - \mathrm{poly}(n)$. By Lemma 6 and Lemma 7, it is equivalent to showing the set of $\bar{\ell} \in G$ generates G. Identify G with a product of at most $\log |G| \leq \log n$ primary cyclic groups C_i whose orders are coprime to each other. We only need to show the the set of holomorphic images of $\bar{\ell} \in G$ generates C_i for each i with probability $1 - \mathrm{poly}(n)$ and then apply the union bound.

So fix one such C_i and let $m = |C_i|$. Then $\phi(m)$ out of the m elements in C_i are generators of C_i. Let α be the probability that the holomorphic image of $\bar{\ell}$ is among these $\phi(m)$ elements, where ℓ is randomly sampled from $[1, n]$ as on Line 3. As m is a prime power, we have $\phi(m) \geq m/2$. Therefore

$$\alpha \geq \frac{\lfloor n/k \rfloor}{n} \cdot \frac{\phi(m)}{m} \cdot |G| = \Omega(\phi(k)/k) = \Omega(1/\log\log k)$$

where we use $k/\phi(k) = O(\log\log k)$ [19]. So for sufficiently large $N = \lfloor c \log n \log\log n \rfloor$, the claim holds with probability $1 - \mathrm{poly}(n)$.

Then we analyze the running time: Line 4 runs in time $\widetilde{O}(\phi(k) \log q)$ by Lemma 6. Line 7 could be implemented in time $\widetilde{O}(|G_0| \log q)$ [13, 20]. And Line 3–9 runs in time $|L| \cdot \max\{e, 1\} \cdot \widetilde{O}(|G_0| \log q)$. This is bounded by $\widetilde{O}(\phi(k) \log q)$ since $|L| = [H_{i-1} : G_0]$, $\max\{e, 1\} = [H_i : H_{i-1}]$, and $|G| = \phi(k)$. As $N = \Theta(\log n \log\log n)$, the total running time is bounded by $\widetilde{O}(\phi(k) \log q)$. ◀

3.1.2 Finding the integer k

Another problem we need to solve is finding the integer k given an irreducible factor $g_0(\lambda)$ of $\Phi_k(\lambda)$ over $\mathbb{F}_q$. Using the procedure `FindCyclotomic`$(g_0(\lambda))$, we could find all the irreducible factors of $\Phi_k(\lambda)$ and hence $\Phi_k(\lambda)$ itself. The degree $d := \deg(\Phi_k(\lambda)) = \phi(k)$ is hence also known. If $|\phi^{-1}(d)|$ is small, we could find k by enumerating $k_0 \in \phi^{-1}(d)$ and checking if $\Phi_{k_0}(\lambda) = \Phi_k(\lambda)$. However, Erdős [9] showed that for some constant $c > 0$, there are infinitely many integers d for which $|\phi^{-1}(d)| \geq d^c$. So this approach is not affordable in general. Based on more sophisticated ideas, we show that it is possible to find k efficiently:

▶ **Lemma 9.** *There exists a procedure* `Findk`$(d, g_0(\lambda))$ *that takes an integer* $d > 0$ *dividing* $\phi(k)$ *and an irreducible factor* $g_0(\lambda)$ *of* $\Phi_k(\lambda)$*, and returns a positive integer* $k_0 | k$ *in time* $\widetilde{O}(\phi(k) \log q)$*. Moreover* $k_0 = k$ *if* $d = \phi(k)$*.*

See the full version of this paper [10] for the proof of Lemma 9 and the description of `Findk`$(d, g_0(\lambda))$.

3.1.3 Finding the set T

Now we are ready to describe the procedure `FindT`$(g_1(\lambda), \ldots, g_m(\lambda))$:

Algorithm 3 `FindT`$(g_1(\lambda), \ldots, g_m(\lambda))$

Input: The irreducible factors $g_1(\lambda), \ldots, g_m(\lambda)$ of $g(\lambda)$ over $\mathbb{F}_q$
Output: The set T and multiplicities m_k for each $k \in T$

```
1:  $L_0 \leftarrow \{g_1(\lambda), \ldots, g_m(\lambda)\}$ as a multi-set and $T \leftarrow \emptyset$
2:  repeat
3:      Pick an arbitrary element $g_0(\lambda) \in L_0$
4:      $L \leftarrow$ FindCyclotomic$(g_0(\lambda))$
5:      $h(\lambda) \leftarrow \prod_{f_i(\lambda) \in L} f_i(\lambda)$, $d \leftarrow \deg(h(\lambda))$
6:      $k_0 \leftarrow$ Findk$(d, g_0)$
7:      if $h(\lambda) | \lambda^{k_0} - 1$ then
8:          if $k_0 \notin T$ then $m_{k_0} \leftarrow 0$
9:          $T \leftarrow T \cup \{k_0\}$, $m_{k_0} \leftarrow m_{k_0} + 1$
10:         $L_0 \leftarrow L_0 - L$
11:     end if
12: until $L_0 = \emptyset$
13: return $T$
```

▶ **Theorem 10** (Theorem 2 restated). `FindT`$(g_1(\lambda), \ldots, g_m(\lambda))$ *computes the set* T *and multiplicities* m_k *as defined in Algorithm 1, Line* 4*. Moreover it halts in time* $\widetilde{O}(n \log q)$ *with probability* $1 - 1/\mathrm{poly}(n)$.

Proof. The algorithm picks $g_0(\lambda)$ from L_0, calls `FindCyclotomic` to find a list $L \subseteq L_0$ that almost surely contains all the irreducible factors of $\Phi_k(\lambda)$, and remove these factors from L_0. It repeats these steps until L_0 is empty. Each time it also determines the integer k using `Findk`, adds it to T and updates m_k.

Note that with small probability, the list L returned by `FindCyclotomic` may not contain all the irreducible factors, in which case it is associated with a proper subgroup $H_N \subseteq G$ (c.f. Lemma 8). In any case we have $\deg(h(\lambda))|\phi(k)$ and therefore by Lemma 9, the integer k_0 returned by `Findk` divides k. We verify that $k = k_0$ on Line 7: $h(\lambda)|(\lambda^{k_0} - 1)$ if and only if $k|k_0$ if and only if $k = k_0$ since we know $k_0|k$. And if we find $k \neq k_0$ we do nothing in that round. The correctness of the algorithm is then straightforward.

For the running time, note that each round runs in time $\widetilde{O}(\phi(k) \log q)$ by Lemma 8 and Lemma 9, and then factors of total degree $\phi(k)$ are removed from L_0 with probability $1 - 1/\mathrm{poly}(n)$. So with probability $1 - 1/\mathrm{poly}(n)$, the total running time is bounded by $\sum_{i=1}^{m} \widetilde{O}(\deg(g_i(\lambda)) \log q) = \widetilde{O}(n \log q)$. ◀

4 Polynomial Factorization Using Carlitz Modules

We next establish connections between polynomial factorization and the Carlitz action. We prove two nearly linear reductions, namely Factor Degree to Carlitz Char-Poly and Carlitz Char-Poly to Factor. The former reduction requires that the characteristic p of $\mathbb{F}_q$ is larger than the number of irreducible factors.

4.1 Carlitz Modules

Let A be an $\mathbb{F}_q[x]$-algebra. For $f(x) \in \mathbb{F}_q[x]$ and $\alpha \in A$, $f(x)\alpha$ is understood to be the result of the $\mathbb{F}_q[x]$ action of $f(x)$ on α in A. Let $\sigma : A \longrightarrow A$ and $\tau : A \longrightarrow A$ denote the q^{th} power Frobenius endomorphism and the multiplication by x endomorphism respectively. That is,

$\forall \alpha \in A$, $\sigma(\alpha) = \alpha^q$ and $\tau(\alpha) = x\alpha$. In [6, 7], Carlitz endowed a new $\mathbb{F}_q[x]$-module structure on A by defining $m(x) = \sum_i m_i x^i \in \mathbb{F}_q[x]$ to act on $\alpha \in A$ as

$$\rho_m(\alpha) := (m(\sigma+\tau))(\alpha) = \left(\sum_i m_i(\sigma+\tau)^i\right)(\alpha).$$

In particular, $\forall \alpha \in A$, $\rho_x(\alpha) = \alpha^q + x\alpha$ and $\forall u \in \mathbb{F}_q$, $\rho_u(\alpha) = u\alpha$. Let $\rho(A)$ denote the $\mathbb{F}_q[x]$-module structure thus endowed to A by the Carlitz action. To factor a monic square free polynomial $f(x)$, we will concern ourselves with $\rho(\mathbb{F}_q[x]/(f(x)))$. Let $\chi_f(x) \in \mathbb{F}_q[x]$ denote the characteristic polynomial of the $\mathbb{F}_q$ linear transformation on $\mathbb{F}_q[x]/(f(x))$ that takes $\alpha \in \mathbb{F}_q[x]/(f(x))$ to $\rho_x(\alpha)$. Hence Carlitz Char-Poly may be restated as

▶ **Problem 11.** Given a monic square free $f(x) \in \mathbb{F}_q[x]$, compute $\chi_f(x)$.

By Lemma 12, knowledge of factorization of $f(x)$ immediately yields $\chi_f(x)$ in $\widetilde{O}(n \log q)$ time. Thus Problem Carlitz Char-Poly is linear time reducible to Factor. We next reduce Factor Degree to Carlitz Char-Poly.

4.2 Factor Degree Estimation using Carlitz Modules

▶ **Lemma 12.** *Let $f(x) = \prod_i f_i(x)$ be a factorization of a monic square free $f(x) \in \mathbb{F}_q[x]$ into monic irreducible polynomials . Then $\rho(\mathbb{F}_q[x]/(f(x))) \cong \bigoplus_i \mathbb{F}_q[x]/(f_i(x) - 1)$. In particular, $\chi_f(x) = \prod_i (f_i(x) - 1)$.*

Proof. By the Chinese remainder theorem, $\mathbb{F}_q[x]/(f(x)) \cong \prod_i \mathbb{F}_q[x]/(f_i(x))$

$$\Rightarrow \rho(\mathbb{F}_q[x]/(f(x))) \cong \rho\left(\prod_i \mathbb{F}_q[x]/(f_i(x))\right) \cong \bigoplus_i \rho\left(\mathbb{F}_q[x]/(f_i(x))\right). \tag{4.1}$$

The final congruence holds since for every direct product $C \cong A \times B$ of $\mathbb{F}_q[x]$-algebras, we have the corresponding direct sum $\rho(C) \cong \rho(A) \oplus \rho(B)$ of $\mathbb{F}_q[x]$-modules [8]. For a monic irreducible $g(x)$ ([8]),

$$\rho(\mathbb{F}_q[x]/(g(x))) \cong \mathbb{F}_q[x]/(g(x) - 1). \tag{4.2}$$

Equation 4.1 and 4.2 together prove the lemma. ◀

▶ **Lemma 13.** *If p does not divide the number of smallest degree factors of a monic square free $f(x) \in \mathbb{F}_q[x]$, then the smallest irreducible factor degree of $f(x)$ is* $\deg(f(x)) - \deg(f(x) - \chi_f(x))$.

Proof. Let $f(x) = \prod_i f_i(x)$ be a factorization of a monic square free $f(x) \in \mathbb{F}_q[x]$ into monic irreducible polynomials. Let d be the smallest degree of factors of $f(x)$. Then

$$f(x) - \chi_f(x) = f(x) - \prod_i (f_i(x) - 1) = \sum_i \frac{f(x)}{f_i(x)} + (\text{terms of degree less than } \deg(f(x)) - d).$$

The first equality is from Lemma 12. Since $f(x)$ and $f_i(x)$ are all monic and p does not divide the number of $f_i(x)$ of degree d, the leading term of $\sum_i (f(x)/f_i(x))$ is of degree $\deg(f(x)) - d$. Therefore $\deg(f(x) - \chi_f(x)) = \deg(f(x)) - d$ and the lemma follows. ◀

Lemma 13 reduces in nearly linear time Factor Degree (when restricted to p greater than the number of factors of $f(x)$) to Carlitz Char-Poly. To see this, given $f(x)$, we may call an algorithm that solves Problem 11 to obtain $\chi_f(x)$ and output $\deg(f(x)) - \deg(f(x) - \chi_f(x))$.

5 Moore and Vandermonde Determinants

5.1 Moore Determinants and Carlitz Factorials

Let A be a finitely generated $\mathbb{F}_q$ algebra and n a positive integer. The Moore matrix M_w with first row $w = (w_1, w_2, \dots, w_n) \in A^n$ is defined as

$$M_w := \begin{bmatrix} w_1 & w_2 & w_3 & \dots & w_n \\ w_1^q & w_2^q & w_3^q & \dots & w_n^q \\ w_1^{q^2} & w_2^{q^2} & w_3^{q^2} & \dots & w_n^{q^2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_1^{q^{n-1}} & w_2^{q^{n-1}} & w_3^{q^{n-1}} & \dots & w_n^{q^{n-1}} \end{bmatrix}$$

and its determinant $det(M_w)$ is denoted by $\Delta(w_1, w_2, \dots, w_n)$. For a positive integer m, the m^{th} Carlitz factorial

$$\prod_{0 \le i < j \le m} \left(x^{q^{j-i}} - x\right)^{q^i},$$

is the product of all polynomials over $\mathbb{F}_q$ of degree at most m [6]. We next recall Carlitz's identity and from it reduce FACTOR DEGREE to computing certain Moore determinants.

▶ **Lemma 14.** *(Carlitz [6]) For every positive integer m,*

$$\Delta(1, x, x^2, \dots, x^m) = \prod_{0 \le i < j \le m} \left(x^{q^{j-i}} - x\right)^{q^i},$$

Proof. The Moore matrix with first row $(1, x, x^2, \dots, x^m)$, when viewed column-wise is Vandermonde. By the Vandermonde determinant formula,

$$det\left(\begin{bmatrix} 1 & x & x^2 & \dots & x^m \\ 1 & x^q & x^{2q} & \dots & x^{mq} \\ 1 & x^{q^2} & x^{2q^2} & \dots & x^{mq^2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x^{q^{n-1}} & x^{2q^{n-1}} & \dots & x^{mq^{n-1}} \end{bmatrix}\right) = \prod_{0 \le i < j \le m} (x^{q^j} - x^{q^i}) = \prod_{0 \le i < j \le m} \left(x^{q^{j-i}} - x\right)^{q^i}$$

◀

MOORE-DET may be restated as

▶ **Problem 15.** Given a square free monic polynomial $f(x) \in \mathbb{F}_q[x]$ of degree n and a positive integer $m \le n$, decide if $\Delta(1, x, \dots, x^m) \mod f(x)$ is 0.

Problem 15 can be solved in $\widetilde{O}(n^{3/2} \log q + n \log^2 q)$ time [13, Lemma 8.4].

▶ **Theorem 16.** *If there is a $T(n, m, \log q)$ time algorithm for Problem 15, then* FACTOR DEGREE *can be solved in $O(T(n, \lceil n/2 \rceil, \log q) \log n)$ time. That is,* FACTOR DEGREE *is nearly linear time reducible to* MOORE-DET.

Proof. By Lemma 14, for a monic square free $f(x) \in \mathbb{F}_q[x]$ and $m \le \deg(f(x))$, we have $\Delta(1, x, \dots, x^m) \mod f(x) = 0$ if and only if

$$\prod_{0 \le i < j \le m} \left(x^{q^{j-i}} - x\right)^{q^i} = 0 \mod f(x). \tag{5.1}$$

Since $f(x)$ is square free, Equation 5.1 holds if and only if every irreducible factor of $f(x)$ has degree at most m. Given oracle access to an algorithm for Problem 15, a binary search leads to the determination of the largest irreducible factor degree of $f(x)$. ◀

5.2 Vandermonde Determinants

The determinants involved in the previous subsection were both Moore and Vandermonde. Here we study determinants that are Vandermonde but not Moore. Further, the matrices involved are of dimension significantly smaller than the degree of the polynomial factored.

For a positive integer m, let

$$S_m := \{0, 1, 2, \dots, \lfloor\sqrt{m}\rfloor - 1, \lfloor\sqrt{m}\rfloor, 2\lfloor\sqrt{m}\rfloor, 3\lfloor\sqrt{m}\rfloor, \dots, (\lfloor\sqrt{m}\rfloor - 1)\lfloor\sqrt{m}\rfloor, \lfloor\sqrt{m}\rfloor^2, m\}.$$

This ensures that $|S_m| \leq 2\lfloor\sqrt{m}\rfloor + 1$ and $\{j - i | i, j \in S_m, i < j\} = \{1, 2, \dots, m-1, m\}$.

For a positive integer m, let $V_m(x) \in \mathbb{F}_q[x]$ denote the determinant of the Vandermonde matrix with first row $\{x^{q^i}, i \in S_m\}$.

▶ **Lemma 17.** *For every monic square free $f(x) \in \mathbb{F}_q[x]$ and every positive integer m,*

$$\gcd(V_m(x), f(x)) = \gcd\left(\prod_{0\leq i\leq m}\left(x^{q^i} - x\right), f(x)\right).$$

Proof. By the Vandermonde determinant formula,

$$V_m(x) = \prod_{i,j\in S_m|i<j}\left(x^{q^j} - x^{q^i}\right) = \prod_{i,j\in S_m|i<j}\left(x^{q^{j-i}} - x\right)^{q^i}. \tag{5.2}$$

Since $f(x)$ is square free and $\{j - i | i, j \in S_m, i < j\} = \{1, 2, \dots, m-1, m\}$,

$$\gcd\left(\prod_{i,j\in S_m|i<j}\left(x^{q^{j-i}} - x\right)^{q^i}, f(x)\right) = \gcd\left(\prod_{0\leq i\leq m}\left(x^{q^i} - x\right), f(x)\right). \tag{5.3}$$

By Equations 5.2 and 5.3, the lemma follows. ◀

Vandermonde Det may be restated as

▶ **Problem 18.** Given a square free monic polynomial $f(x) \in \mathbb{F}_q[x]$ of degree n and a positive integer $m \leq n$, decide if $V_m(x) \mod f(x)$ is 0.

We next sketch a fast algorithm for Problem 18. Since $|S_m| \leq \sqrt{n}$, the first row $\{x^{q^i} \mod f(x), i \in S_m\}$ can be computed in $\widetilde{O}(n^{3/2}\log q + n\log^2 q)$ time using iterated Frobenius algorithm [21] implemented using fast modular composition [13]. Given the first row of a Vandermonde matrix over a commutative ring, the square of its determinant can be computed with nearly linearly many operations over the ring [16]. Hence, $V_m(x) \mod f(x)$ can be zero tested in $\widetilde{O}(n^{3/2}\log q + n\log^2 q)$ time.

▶ **Theorem 19.** *If there is a $T(n, m, \log q)$ time algorithm for Problem 18, then* Factor Degree *can be solved in $O(T(n, \lceil\sqrt{n}\rceil, \log q)\log n)$ time. That is,* Factor Degree *is nearly linear time reducible to* Vandermonde Det.

Proof. By Lemma 14 and Lemma 17, for every monic square free $f(x) \in \mathbb{F}_q[x]$ and positive integer $m \leq \deg(f(x))$,

$$\gcd(V_m(x), f(x)) = \gcd(\Delta(1, x, \dots, x^m), f(x)).$$

Hence Problems 15 and 18 are identical and our theorem follows from Theorem 16. ◀

References

1 A. Abboud, F. Grandoni, and V. V. Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1681–1697, 2015.

2 A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th Annual Symposium on Foundations of Computer Science*, pages 434–443, 2014.

3 A. Abboud, V. V. Williams, and O. Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming*, pages 39–51, 2014.

4 E. R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46(8):1853–1859, 1967.

5 D. G. Cantor and H. Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

6 L. Carlitz. On certain functions connected with polynomials in a Galois field. *Duke Math. J.*, 1(2):137–168, 06 1935.

7 L. Carlitz. A class of polynomials. *Transactions of the American Mathematical Society*, 43(2):167–182, 1938.

8 K. Conrad. Carlitz extensions, available online at. `http://www.math.uconn.edu/~kconrad/blurbs/gradnumthy/carlitz.pdf`.

9 P. Erdős. On the normal number of prime factors of $p-1$ and some related problems concerning Euler's ϕ-function. *Quart. J. Math*, 6:205–213, 1935.

10 Z. Guo, A. K. Narayanan, and C. Umans. Algebraic problems equivalent to beating exponent 3/2 for polynomial factorization over finite fields. *arXiv preprint arXiv:1606.04592*, 2016.

11 E. Kaltofen and A. Lobo. Factoring high-degree polynomials by the black box berlekamp algorithm. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 90–98, 1994.

12 E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Mathematics of computation*, 67(223):1179–1197, 1998.

13 K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

14 D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms.* Addison-Wesley, 1997.

15 A. K. Narayanan. Polynomial factorization over finite fields by computing Euler-Poincare characteristics of Drinfeld modules. *arXiv preprint arXiv:1504.07697*, 2015.

16 V. Pan. On computations with dense structured matrices. *Mathematics of Computation*, 55(191):179–190, 1990.

17 M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 603–610, 2010.

18 L. Roditty and U. Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33:1–33:11, 2012.

19 J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, 6(1):64–94, 1962.

20 V. Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation*, pages 53–58, 1999.

21 J. Von Zur Gathen and V. Shoup. Computing frobenius maps and factoring polynomials. *Computational complexity*, 2(3):187–224, 1992.

22 O. Weimann and R. Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, 2013.

23 V. V. Williams. Faster replacement paths. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1337–1346, 2011.

24 V. V. Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, pages 645–654, 2010.

25 D. Y.Y. Yun. On square-free decomposition algorithms. In *Proceedings of the 3rd ACM Symposium on Symbolic and Algebraic Computation*, pages 26–35, 1976.

On Synchronizing Colorings and the Eigenvectors of Digraphs[*]

Vladimir V. Gusev[1] and Elena V. Pribavkina[1]

1 **Institute of Mathematics and Computer Science, Ural Federal University Ekaterinburg, Russia and ICTEAM Institute, Université catholique de Louvain Louvain-la-Neuve, Belgium**
`vl.gusev@gmail.com`

2 **ICTEAM Institute, Université catholique de Louvain Louvain-la-Neuve, Belgium**
`vl.gusev@gmail.com`

Abstract

An automaton is synchronizing if there exists a word that sends all states of the automaton to a single state. A coloring of a digraph with a fixed out-degree k is a distribution of k labels over the edges resulting in a deterministic finite automaton. The famous road coloring theorem states that every primitive digraph has a synchronizing coloring. We study recent conjectures claiming that the number of synchronizing colorings is large in the worst and average cases.

Our approach is based on the spectral properties of the adjacency matrix $\mathcal{A}(G)$ of a digraph G. Namely, we study the relation between the number of synchronizing colorings of G and the structure of the dominant eigenvector $\vec{v}$ of $\mathcal{A}(G)$. We show that a vector $\vec{v}$ has no partition of coordinates into blocks of equal sum if and only if all colorings of the digraphs associated with $\vec{v}$ are synchronizing. Furthermore, if for each b there exists at most one partition of the coordinates of $\vec{v}$ into blocks summing up to b, and the total number of partitions is equal to s, then the fraction of synchronizing colorings among all colorings of G is at least $\frac{k-s}{k}$. We also give a combinatorial interpretation of some known results concerning an upper bound on the minimal length of synchronizing words in terms of $\vec{v}$.

1998 ACM Subject Classification F.1.1 Models of Computation, G.2.2 Graph Theory

Keywords and phrases The road coloring problem, synchronizing automata, edge-colorings of digraphs, Perron-Frobenius eigenvector, primitive digraphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.48

1 Introduction

Let $\mathscr{A} = (Q, \Sigma, \delta)$ be a finite deterministic complete automaton with an alphabet Σ, a set of states Q and a transition function δ. The automaton $\mathscr{A}$ is *synchronizing* if there exist a word u and a state p such that for every state $q \in Q$ we have $q \cdot u = p$, where $q \cdot u$ denotes the image of q under the action of u. Any such word u is called *synchronizing* (or *reset*) word for $\mathscr{A}$. The length of the shortest synchronizing word $\mathrm{rt}(\mathscr{A})$ is called *the reset threshold* of $\mathscr{A}$. Synchronizing automata naturally appear in algebra, coding theory, industrial automation,

* The authors benefited from the Russian foundation for basic research (grant 16-01-00795), Ministry of Education and Science of the Russian Federation (project no. 1.1999.2014/K), and the Competitiveness Program of Ural Federal University. The first author was also supported by Interuniversity Attraction Poles (IAP) Programme, and by the ARC grant 13/18-054 (Communauté française de Belgique).

discrete dynamical systems, etc. A brief survey of the theory of synchronizing automata may be found in [19].

Two fundamental problems about synchronizing automata that were intensively investigated in the last decades are the Černý conjecture and the road coloring problem. The former states that the reset threshold of an n-state automaton is at most $(n-1)^2$ [9]. Despite intensive research efforts it remains open for already half a century. The latter problem states a certain connection between primitive digraphs and synchronizing automata, which we will explain shortly, and was recently resolved by Trakhtman [18] after crucial insight by Culik, Karhumäki, and Kari [10]. Our paper is devoted to the generalizations of the road coloring theorem.

1.1 The road coloring theorem

The underlying digraph $\mathcal{G}(\mathscr{A})$ of an automaton $\mathscr{A}$ is a digraph with Q as a set of vertices, and for each $u \in Q, x \in \Sigma$ there is an edge $(u, u \cdot x)$. We allow loops and multiple edges, thus $\mathcal{G}(\mathscr{A})$ has a fixed out-degree equal to the cardinality of the alphabet Σ, i.e., $\mathcal{G}(\mathscr{A})$ is a $|\Sigma|$-out-regular digraph.

Vice versa, given a digraph G with a fixed out-degree k and a finite alphabet Σ with k letters, we can obtain a deterministic finite automaton by distributing the letters of Σ over the edges of G. Any automaton obtained in this way is called a *coloring* of G. A digraph is *primitive* if there exists a number t such that for any two vertices u and v there exists a path from u to v of length exactly t. An automaton is *strongly connected* if its underlying digraph is strongly connected.

▶ **Theorem 1** (Road coloring theorem). *A strongly connected digraph G with a fixed out-degree k has a synchronizing coloring if and only if it is primitive.*

This theorem was stated as a conjecture in 1977 [1]. The authors' original motivation comes from symbolic dynamics. Namely, synchronizing coloring defines a morphism from a shift of finite type given by G to a full shift over Σ with special properties, see [3].

The origin of the terminology is as follows. A digraph G represents a network of one-way roads. A coloring of G defines labels of the roads that can be perceived by drivers. If the coloring is synchronizing then the drivers who are unaware of their current location have the following strategy to relocate themselves: they can simply follow roads labelled by a synchronizing word and their final position will be well defined.

Although the road coloring theorem gives an answer for a principal connection between digraphs and synchronizing automata, there are still basic quantitative questions that remain unanswered. Namely, how many synchronizing colorings a primitive digraph G can have and what is the number of synchronizing colorings of an average (or random) digraph? These questions were addressed in [13] and two conjectures were formulated as a result of extensive computational experiments. In order to state them, we will need some definitions.

The *synchronizing ratio* of a digraph G is the number of synchronizing colorings divided by the total number of colorings. Note, that a coloring is a mapping from the set of edges to Σ with parallel edges being distinguished. Thus, the total number of colorings of a k-out-regular digraph with n states is always k^n.

▶ **Conjecture 2.** *The minimum value of the synchronizing ratio among all k-out-regular primitive digraphs with n vertices is equal to $\frac{k-1}{k}$, except for the case $k = 2$ and $n = 6$ when it is equal to $\frac{30}{64}$.*

We say that the digraph is *totally synchronizing* if its synchronizing ratio is equal to 1, i.e., every coloroing is synchronizing.

▶ **Conjecture 3.** *For every $k \geq 2$, the fraction of totally synchronizing digraphs among all k-out-regular primitive digraphs with n vertices tends to 1 as n goes to infinity.*

If both conjectures are true, then the road coloring theorem is a relatively weak statement that gives us just the first step towards satisfactory understanding of the synchronizing properties of automata and digraphs.

We want to mention another direction to strengthen the road coloring theorem.

▶ **Conjecture 4** (Hybrid Černý–Road Coloring Problem)**.** *Every primitive k-out-regular digraph with n vertices has a synchronizing coloring with the reset threshold at most $n^2 - 3n + 3$.*

This conjecture was made by M. V. Volkov and partial results were obtained in [17, 8].

1.2 Our contributions

One of the major obstacles in approaching conjectures 2 and 3 comes from the difficulty of proving that a coloring under consideration is synchronizing. A simple and straightforward proof of this fact tends to be tedious and technical even for relatively simple automata, see for example [12]. In order to overcome this difficulty we rely on spectral properties of the adjacency matrix $\mathcal{A}(G)$ of a primitive k-out-regular digraph G.

More precisely, Perron-Frobenius theorem [15, Chapter 8] implies existence of entrywise positive eigenvector $\vec{v}$ of $\mathcal{A}(G)$ associated with the unique largest eigenvalue, which we will simply call *the eigenvector* of G. The vector $\vec{v}$ can also be seen as the unique stationary distribution of the Markov chain associated with G by assigning the probability $\frac{1}{k}$ for each of the outgoing edges.

The importance of the eigenvector of G in the context of synchronizing automata was demonstrated by Friedman [11]. We will require a few definitions to state his result. Let $Q = \{1, \ldots, n\}$, and $\vec{v}[i]$ be the ith entry of $\vec{v}$. The *weight* of a subset $S \subseteq Q$ is given by $\mathrm{wg}(S) = \sum_{i \in S} \vec{v}[i]$. The subset $S \subseteq Q$ is *synchronizing* if there exists a word u and a state p such that for every $q \in S$ we have $q \cdot u = p$.

▶ **Theorem 5.** *Every coloring $\mathscr{A}$ of G has a partition of vertices into synchronizing subsets $Q_1, \ldots, Q_\ell$ such that $\mathrm{wg}(Q_1) = \ldots = \mathrm{wg}(Q_\ell)$ and for any other synchronizing subset S we have $\mathrm{wg}(S) \leq \mathrm{wg}(Q_1)$.*

A simple corollary of this statement allows us to easily identify a relatively large class of totally synchronizing digraphs. We will say that a vector $\vec{v}$ is *partitionable* if there exists a partition of $\vec{v}$ into blocks of equal weight b, i.e., a partition $Q_1, \ldots, Q_\ell$ of Q with $\ell > 1$ such that $\sum_{i \in Q_1} \vec{v}[i] = \ldots = \sum_{i \in Q_\ell} \vec{v}[i] = b$. Clearly, a digraph with non-partitionable eigenvector is totally synchronizing, otherwise maximal synchronizing subsets, i.e. synchronizing subsets with the largest weight, give rise to a partition by theorem 5. Our first contribution is the converse (in some sense) of this statement. Namely, let $\mathcal{G}(\vec{v})$ be the class of primitive digraphs of fixed out-degree with the eigenvector $\vec{v}$. We show that all digraphs in $\mathcal{G}(\vec{v})$ are totally synchronizing if and only if $\vec{v}$ is non-partitionable. We also formulate an algebraic conjecture that implies conjecture 3. These results are given in section 3.

Our second contribution is a lower bound on the synchronizing ratio of G depending on the structure of $\vec{v}$. We say that the partition $Q_1, \ldots, Q_\ell$ of $\vec{v}$ into blocks of weight b is *unique* if for every partition $Q'_1, \ldots, Q'_\ell$ of weight b there exists a permutation of $1, \ldots, \ell$ such that $Q_i = Q'_{\sigma(i)}$ for all i. In section 4 we show that if all partitions of $\vec{v}$ into blocks of equal weight are unique and their number is bounded by s, then the synchronizing ratio of G is at least $\frac{k-s}{k}$. Note, that for $s = 1$ we obtain the bound of conjecture 2. To the best of our knowledge it is the first result that shows validity of the conjecture on a relatively large class of digraphs, e.g., this class contains all primitive Eulerian[1] digraphs with a prime number of states.

Let $\mathscr{A}$ be a coloring of G. We can consider an arbitrary probability distribution on the letters of $\mathscr{A}$ turning it into a Markov chain. Similarly to the previous uniform case we obtain an eigenvector $\vec{v'}$ corresponding to the unique stationary distribution. The vector $\vec{v'}$ played an important role in various proofs of the Černý conjecture in the special classes of automata, see e.g. [16, 5, 6]. Our third contribution is related to such approaches. First, in section 2 we generalize theorem 5 to the case of arbitrary probability distributions on the alphabet. Secondly, in section 5 we present a combinatorial reduction from an arbitrary synchronizing automaton $\mathscr{A}$ to an Eulerian automaton with possibly larger number of states, which has the same reset threshold as $\mathscr{A}$. This reduction gives a combinatorial view of results by Berlinkov [4] and Steinberg [16].

2 Partitions into synchronizing subsets

In the present section we will prove a generalization of Theorem 5. Let $\mathscr{A}$ be a strongly connected automaton with the set of states $\{1, 2, \ldots, n\}$. Let $A_1, A_2, \ldots, A_k$ be the adjacency matrices of the letters of $\mathscr{A}$, i.e., $A_\ell[i, j] = 1$ if i is mapped to j under the action of the ℓth letter, and $A_\ell[i, j] = 0$ otherwise.

Consider the matrix $A = \sum_{i=0}^{k} p_i A_i$, where $p_i > 0$ are rational for all i and $\sum_{i=0}^{k} p_i = 1$. Since the matrix A is row-stochastic the largest eigenvalue of A is equal to 1. By the Perron-Frobenius theorem [15, Chapter 8] there exists a positive left eigenvector $\vec{u}$ such that $\vec{u}A = \vec{u}$. Since the entries of A are rational, so are the entries of $\vec{u}$. Let $\vec{w} = \ell\vec{u}$, where ℓ is the least common multiple of the denominators of entries of $\vec{u}$. We will call the vector $\vec{w}$ *the eigenvector* of $\mathscr{A}$ in accordance with the distribution $p_1, \ldots, p_k$. If the distribution is uniform, i.e., $p_1 = p_2 = \ldots = p_k = \frac{1}{k}$, then we will usually omit its description. Since all colorings of a digraph G have the same eigenvector $\vec{w}$ in accordance with the uniform distribution we will call $\vec{w}$ obtained in this way *the eigenvector* of G.

The *kernel* of a word x with respect to an automaton $\mathscr{A}$ is an equivalence relation ρ on the set of states Q such that $i\rho j$ if and only if $i \cdot x = j \cdot x$. A subset S is *synchronizing* if there exists a word x such that the cardinality of $S \cdot x = \{q \cdot x \mid q \in S\}$ is equal to 1. By $S \cdot x^{-1}$ we denote the full preimage of the set S under the action of a word x, i.e., $S \cdot x^{-1} = \{q \in Q \mid q \cdot x \in S\}$. Let $\vec{w}$ be the eigenvector of the automaton $\mathscr{A}$. We define *the weight* $\mathrm{wg}(i)$ of a state i as $\vec{w}[i]$. The weight of a set S is defined as $\mathrm{wg}(S) = \sum_{i \in S} \mathrm{wg}(i)$.

▶ **Theorem 6.** *Let $\vec{w}$ be the eigenvector of a strongly connected automaton $\mathscr{A}$ in accordance with a distribution $p_1, p_2, \ldots, p_k$. There exists a partition of the states of $\mathscr{A}$ into synchronizing subsets of maximal weight. Furthermore, this partition is equal to the kernel of some word x.*

[1] A digraph is *Eulerian* if the outdegree and indegree of each vertex is equal to k for some constant k. The eigenvector of such digraph is equal to $(1, 1, \ldots, 1)$.

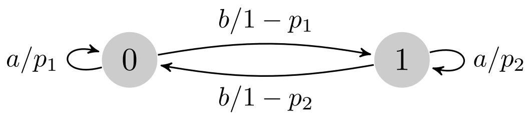


Figure 1 Automaton $\mathscr{F}$.

Proof. Let $\Sigma = \{a_1, a_2, \dots, a_k\}$, and let S be an arbitrary subset of Q. Note the following equality:

$$\sum_{i=1}^{k} p_i \operatorname{wg}(S \cdot a_i^{-1}) = \operatorname{wg}(S)$$

(the incoming edges to S in total bring the weight equal to $\operatorname{wg}(S)$, and each preimage brings $p_i \operatorname{wg}(Sa_i^{-1})$; the weights are equal, since $\vec{w}$ is the eigenvector of $\mathscr{A}$). If S is a synchronizing subset of maximal weight, then the weights of preimages are bounded by $\operatorname{wg}(S)$, since every preimage is also a synchronizing subset. Moreover, every preimage has the weight equal to $\operatorname{wg}(S)$, otherwise the left-hand side would be strictly less than the right-hand side. Therefore, if S is a synchronizing subset of maximal weight, then every preimage of S is a synchronizing subset of maximal weight.

We will iteratively construct a partition of the set of states of $\mathscr{A}$ into synchronizing subsets of maximal weight. Let S_0 be a synchronizing subset of maximal weight. Let u be a word synchronizing S_0 to some state q: $S_0 \cdot u = q$. If $S_0 = Q$, then the automaton is synchronizing, and the proof is complete. Otherwise, let p be a state that doesn't belong to S_0. Since the automaton $\mathscr{A}$ is strongly connected, there exists a word v such that $q \cdot v = p$. Consider now the sets $S_1 = S_0 \cdot (uv)^{-1}$ and S_0. Note, that S_1 is also a maximal synchronizing subset by the preceding paragraph. Furthermore, both sets are synchronized by uvu. But their images are different, since q is not equal to $p \cdot u$ due to maximality of S_0. Continuing in the same manner we will eventually construct the desired partition of Q. ◀

From the matrix theory point of view, independent assignment of probabilities to the edges of the underlying digraph of $\mathscr{A}$ is more natural than the assignment of probabilities to the letters. Unfortunately, Theorem 6 does not hold in this case. Let $\mathscr{F}$ be the automaton depicted in Fig. 1. The notation ℓ/p means that the edge is labelled by ℓ and has the probability p. Note, that the eigenvector of $\mathscr{F}$ is equal to $(1-p_2, 1-p_1)$. Since every letter acts as a permutation, the automaton $\mathscr{F}$ is not synchronizing. Therefore, the partition of the states into synchronizing subsets should be of the form $\{\{0\},\{1\}\}$, but for $p_2 = \frac{1}{3}$ and $p_1 = \frac{1}{2}$ these subsets have different weight.

▶ **Corollary 7.** *Let $\vec{w}$ be the eigenvector of an automaton $\mathscr{A}$ in accordance with a distribution $p_1, p_2, \dots, p_k$, and the weight of a subset of states S is given by $\sum_{i \in S} \vec{w}[i]$. If there is no partition of the states into subsets of equal weight, then the automaton $\mathscr{A}$ is synchronizing.*

Unfortunately, the converse of this corollary does not hold. Let $\mathscr{B}$ be an automaton depicted in Fig. 2. It is synchronized by the word *bbaab* to the state 1. If p and $1-p$ are the probabilities of the letters a and b respectively, then the eigenvector of $\mathscr{B}$ is equal to $(1, 1, p, p)$. Thus, the subsets $\{0, 2\}$ and $\{1, 3\}$ form a partition of the states of $\mathscr{B}$ for any p, in other words, there is no witness of the fact that $\mathscr{B}$ is synchronizing.

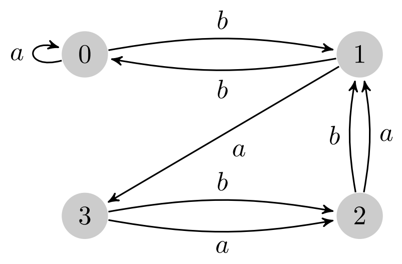


Figure 2 Automaton $\mathscr{B}$.

3 The eigenvectors of totally synchronizing digraphs

Let $\vec{w}$ be an entrywise positive integer vector. We denote by $\mathcal{G}(\vec{w})$ the class of primitive digraphs with the eigenvector $\vec{w}$ such that every digraph in this class has a fixed out-degree (which can be different for two different digraphs from the class). In this section we will characterize in terms of $\vec{w}$ the classes $\mathcal{G}(\vec{w})$ consisting of only totally synchronizing digraphs.

Let $\mathscr{A}$ be an automaton with the set of states Q and an alphabet Σ. Recall that an equivalence relation $\sim$ on Q is a *congruence* if $i \sim j$ implies $i \cdot x \sim j \cdot x$ for all $i, j \in Q$ and $x \in \Sigma$. The *factor automaton* $\mathscr{A}/\sim$ of $\mathscr{A}$ with respect to $\sim$ is defined as follows. The set of states of $\mathscr{A}/\sim$ is equal to the equivalence classes of $\sim$, and its alphabet is equal to Σ. The action of a letter x on an equivalence class C defined in accordance with the representative $c \in C$, i.e., $C \cdot x$ is equal to the class of $c \cdot x$ in $\mathscr{A}$. Since $\sim$ is a congruence, this definition is correct and does not depend on the representative c.

We will call an equivalence relation β on the coordinates of $\vec{w}$ a *partition* if it has at least two classes and satisfies the following property: there exists a constant b such that for every class B of β we have $\sum_{i \in B} \vec{w}[i] = b$. We will refer to the classes of partition β as *blocks*. If $\vec{w}$ is the eigenvector of an automaton $\mathscr{A}$, then every coordinate corresponds to a state of $\mathscr{A}$. Thus, we can naturally obtain an equivalence relation β' on the states of $\mathscr{A}$ from the partition β. Abusing notation, we will refer to β' as β. A vector $\vec{w}$ is called *partitionable* if it possesses a partition.

▶ **Theorem 8.** *An entrywise positive integer vector $\vec{w}$ is not partitionable if and only if all digraphs from $\mathcal{G}(\vec{w})$ are totally synchronizing.*

Proof. Let G be a digraph from $\mathcal{G}(\vec{w})$. If G has a non-synchronizing coloring, then by Theorem 6 it admits a partition of the states into synchronizing subsets of equal weight. Since the coloring is not synchronizing such partition has at least two blocks. Thus, the vector $\vec{w}$ is also partitionable.

Assume now that $\vec{w}$ is partitionable, i.e., there are sets $B_1, B_2, \ldots, B_\ell$ such that for every i we have $\sum_{j \in B_i} \vec{w}[j] = b$. Let n be the number of entries of $\vec{w}$. We will construct a digraph G belonging to $\mathcal{G}(\vec{w})$ on the set of vertices $V = \{0, 1, \ldots, n-1\}$ that is not totally synchronizing as follows: for every pair of vertices i and j there is an edge (i, j) of multiplicity $\vec{w}[j]$.

First, let us show that $G \in \mathcal{G}(\vec{w})$. Note, that the out-degree of every vertex is equal to the sum of entries of $\vec{w}$, i.e., $b\ell$. Furthermore, the digraph G is primitive since there is a path of length 1 between every two vertices. It remains to show that $\vec{w}$ is the eigenvector of G corresponding to the eigenvalue 1. Let $c_{ij} = \vec{w}[j]$ be the multiplicity of the edge from i to j, and $c = b\ell$ be the out-degree of G. We have

$$\sum_{i\in V} \frac{c_{ij}}{c}\vec{w}[i] = \sum_{i\in V} \frac{\vec{w}[j]}{c}\vec{w}[i] = \vec{w}[j]\sum_{i\in V} \frac{\vec{w}[i]}{c} = \vec{w}[j]$$

(the incoming and the outgoing weights are equal). Therefore, $\vec{w}$ is the eigenvector of G.

Now we are going to construct a non-synchronizing coloring of G. We will write $i\,\beta\,j$ for $i, j \in V$ if both i and j belong to B_s for some s. Let $\mathcal{A}$ be the set of colorings of G that have β as a congruence, i.e., for every letter x and for every pair of states i, j such that $i\,\beta\,j$ we necessarily have that $(i \cdot x)\,\beta\,(j \cdot x)$. The set $\mathcal{A}$ is not empty, since it contains the following coloring: the action of the first $\vec{w}[1]$ letters brings all states to the state 1, the action of the next $\vec{w}[2]$ letters brings all states to the state 2, and so on.

Let us fix some automaton $\mathscr{A} \in \mathcal{A}$. Recall that an automaton over k-letter alphabet is Eulerian if the indegree (and the out-degree) of every state is equal to k. Clearly, an automaton is Eulearian if and only if its eigenvector is equal to $(1, 1, \ldots, 1)$. We will show now that the factor automaton $\mathscr{A}'$ of $\mathscr{A}$ with respect to β is Eulerian. Let Σ be the alphabet of $\mathscr{A}$ and $\mathscr{A}'$. Relying on the fact that $\vec{w}$ is the eigenvector of $\mathscr{A}$ we have the following equalities for every block B_t:

$$\sum_{\substack{i\in Q, x\in\Sigma\\ i\cdot x = j}} \frac{1}{c}\vec{w}[i] = \vec{w}[j] \quad\Rightarrow \sum_{\substack{i\in Q, x\in\Sigma\\ i\cdot x \in B_t}} \frac{1}{c}\vec{w}[i] = b \quad\Rightarrow \sum_{\substack{B_s, x\in\Sigma\\ B_s\cdot x = B_t}} \frac{1}{c} b = b$$

The last equality ensures that $(b, b, \ldots, b)$ is the eigenvector of $\mathscr{A}'$, thus, it is Eulerian.

Lemma 1 from [14] states that every Eulerian automaton has a non-synchronizing coloring[2]. Thus, we can recolor an automaton $\mathscr{A}'$ into a non-synchronizing automaton $\mathscr{B}'$. Such recoloring procedure can be seen as a sequence of basic flips, i.e., for a fixed B_t, $x_1, x_2 \in \Sigma$ we change the label from x_1 to x_2 and vice versa on the outgoing edges of B_t. Therefore, this recoloring can be applied to $\mathscr{A}$ leading to an automaton $\mathscr{B}$ in the following manner: a basic flip is applied simultaneously to all states of B_t. The latter ensures that β is a congruence of $\mathscr{B}$ and $\mathscr{B}'$ is the factor automaton of $\mathscr{B}$ with respect to β. Note that the automaton $\mathscr{B}$ is not synchronizing, since any synchronizing word of the automaton $\mathscr{B}$ will synchronize the automaton $\mathscr{B}'$ leading to a contradiction. ◀

Theorem 8 allows us to obtain very simple proofs for otherwise non-obvious statements. Recall that the Černý automaton $\mathscr{C}_n$ [9] can be defined as $\langle\{0, \ldots, n-1\}, \{a, b\}, \delta\rangle$, where $\delta(i, a) = i + 1$ for $i < n - 1$, $\delta(n - 1, a) = 0$, $\delta(n - 1, b) = 0$, and $\delta(i, b) = i$ for $i < n - 1$.

▶ **Proposition 9.** *[13, Proposition 2] The underlying digraph of the Černý automaton $\mathscr{C}_n$ is totally synchronizing.*

Proof. It is easy to verify that the eigenvector $\vec{w}$ of the underlying digraph of the n-state Černý automaton is equal to $(2, 2, \ldots, 2, 1)$. Since in every partition exactly one block will have an odd sum, we conclude that $\vec{w}$ is not partitionable. Thus, the digraph is totally synchronizing. ◀

A similar proof can be presented for many other examples in [2].

Another application of theorem 8 is related to conjecture 3. We believe that significant progress on this conjecture can be made through the study of the eigenvectors of digraphs.

[2] It is also a relatively simple corollary of the Birkhoff-von Neumann theorem

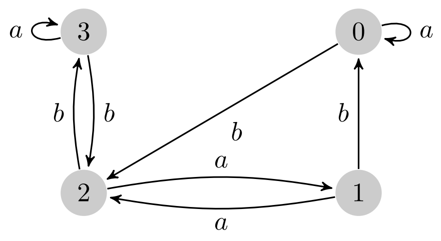


Figure 3 Automaton $\mathscr{D}$.

Despite the fact that the statement of Theorem 8 gives only a necessary condition for a digraph to be totally synchronizing we expect it to hold in most cases. More formally, we state the following conjecture:

▶ **Conjecture 10.** *The eigenvector of a random primitive k-out-regular digraph with n vertices has no partition into blocks of equal sum with probability 1 as n goes to infinity.*

This conjecture has the following interpretation in terms of Markov chains theory. A primitive k-out-regular digraph G correspond to Markov chain via the distribution of the probability $\frac{1}{k}$ for each edge. Furthermore, this chain is mixing, i.e., irreducible and aperiodic, and its stationary distribution is equal to the eigenvector of the the digraph. Informally speaking, a partition of the eigenvector corresponds to a partition of states of the Markov chain into classes such that an infinitely long random walk will spend equal amount of time in each of the classes. Conjecture 10 states that the fraction of Markov chains with this property goes to 0 as the number of states grows.

▶ **Corollary 11.** *If the eigenvector of G is not partitionable, then G is totally synchronizing.*

There are classes of digraphs $\mathcal{G}(\vec{w})$ that contain both totally synchronizing and not totally synchronizing digraphs. Let $\vec{w}$ be $(1, 1, 2, 2)$. The underlying digraph of the automaton $\mathscr{D}$, see fig. 3, belongs to $\mathcal{G}(\vec{w})$. It is not totally synchronizing, since the pair $\{2, 3\}$ is not synchronizable in the coloring $\mathscr{D}$. At the same time, it is easy to see that the underlying digraph of the automaton $\mathscr{B}$, see fig. 2, belongs to $\mathcal{G}(\vec{w})$ and it is totally synchronizing.

There are also classes of digraphs $\mathcal{G}(\vec{w})$ which do not contain totally synchronizing digraphs at all. Namely, if $\vec{w} = (1, 1, \ldots, 1)$ then every digraph in $\mathcal{G}(\vec{w})$ is Eulerian, thus it possesses a non-synchronizing coloring [14, Lemma 1].

4 Partitions of the eigenvectors and the synchronizing ratios

In this section we will present a bound on the synchronizing ratio of a digraph G depending on the structure of its eigenvector. It can be seen as the the first theoretical statement supporting conjecture 2. In order to obtain our result we will rely on the following key lemma:

▶ **Lemma 12.** *Let $\mathscr{A}$ be a non-synchronizing automaton with the eigenvector $\vec{w}$. A partition into maximal synchronizing subsets is unique if and only if it is a congruence.*

Proof. Assume first that the partition into maximal synchronizing subsets is unique. We will denote the block containing a state p by $[p]$. If the partition is not a congruence, then there exists a letter ℓ such that $[p] = [q]$ and $[p\ell] \neq [q\ell]$ for some states p and q. Note, that the preimage of a maximal synchronizing subset by any letter is also a maximal synchronizing subset (see the proof of Theorem 6). Hence, the preimage of a partition into maximal synchronizing subsets is also a partition into maximal synchronizing subsets. Thus, $[p\ell]\ell^{-1}$ is a maximal synchronizing subset and $[p\ell]\ell^{-1} \cap [p] \neq \varnothing$. We also have $[p\ell]\ell^{-1} \neq [p]$, otherwise we get $[p\ell] = [p]\ell$ which implies $[p\ell] = [q\ell]$. Therefore, the preimage of the partition by the letter ℓ is a different partition into maximal synchronizing subsets. A contradiction.

Let τ be a partition into maximal synchronizing subsets. Let us assume that the partition τ is a congruence. Assume to the contrary that there is another partition σ into synchronizing subsets of maximal weight. Note, that there are states p and q such that $p \sim_\sigma q$ and $p \nsim_\tau q$, otherwise σ is a refinement of τ, and σ is not a partition into synchronizing subsets of maximal weight. Since $p \sim_\sigma q$ there exists a word u such that $pu = qu$. Let $[p]$ and $[q]$ be the blocks of the partition τ of p and q respectively. Since τ is a congruence both $[p]u$ and $[q]u$ are subsets of the same block $[r]$ for some state r. The subset $[r]$ is synchronizing. Therefore, the subset $[p] \cup [q]$ is also synchronizing, which contradicts maximality of $[p]$ and $[q]$. ◀

▶ **Corollary 13.** *A digraph G with the eigenvector $\vec{w}$ is totally synchronizing if the following conditions hold:*
1. *if there exists a partition of $\vec{w}$ into blocks of weight b, then it is unique;*
2. *every partition of $\vec{w}$ is not a congruence for every coloring.*

Let Q be a set of states of an automaton $\mathscr{A}$ with the eigenvector $\vec{w}$. A partition of $\vec{w}$ into blocks of weight b is a partition $Q_1, \ldots, Q_\ell$ of Q with $\ell > 1$ such that $\sum_{i \in Q_1} \vec{w}[i] = \ldots = \sum_{i \in Q_\ell} \vec{w}[i] = b$. For simplicity in this section we will sometimes say "a partition" meaning a partition into blocks of equal weight. A partition $Q_1, \ldots, Q_\ell$ of $\vec{w}$ into blocks of weight b is unique if for every partition $Q'_1, \ldots, Q'_\ell$ of weight b there exists a permutation of $1, \ldots, \ell$ such that $Q_i = Q'_{\sigma(i)}$ for all i.

▶ **Theorem 14.** *If all partitions of the eigenvector $\vec{w}$ are unique and their number is equal to s, then the synchronizing ratio of every k-out-regular digraph in $\mathcal{G}(\vec{w})$ is at least $\frac{k-s}{k}$.*

Proof. Every non-synchronizing coloring is associated with a partition of $\vec{w}$ according to theorem 6. We will show that with every partition at most $\frac{1}{k} \cdot k^n$ such colorings can be associated. Thus, the total number of non-synchronizing colorings will be bounded by $\frac{s}{k} \cdot k^n$, and the theorem will follow.

Let G be a digraph in $\mathcal{G}(\vec{w})$, and let β be one of the partitions of $\vec{w}$. In order to show that the fraction of non-synchronizing colorings associated with β is at most $\frac{1}{k}$ we will consider two cases depending on the structure of G and β.

Case I: there are two distinct vetrices q, p belonging to the same block B of β with the following property: there are edges (q, q') and (p, p') such that q' and p' belong to different blocks of β. Let $\mathscr{A}$ be a non-synchronizing coloring associated with β (if there is no such coloring, then the proof is complete). By lemma 12 the partition β is a congruence for $\mathscr{A}$. Thus, for every block B' there is the same number of letters (and edges) going from q to B' and from p to B'. Let k_1 be the number of edges going from q to B_1, k_2 be the number of edges going from q to B_2, …, k_ℓ be the number of edges going from q to B_ℓ, where $B_1, \ldots, B_\ell$ are blocks of β and $k_1, \ldots, k_\ell$ are positive integers such that $\sum_{i=1}^{\ell} k_i = k$.

Now we will divide all colorings of G into classes and show that the fraction of non-synchronizing colorings in each class is at most $\frac{1}{k}$. Let us fix a coloring C of all edges, except for the outgoing edges of q and p. Let $\mathcal{A}(C)$ be the set of automata obtainable from C by all possible colorings of the remaining edges. We will show that the fraction of non-synchronizing automata in $\mathcal{A}(C)$ is at most $\frac{1}{k}$. By lemma 12 every non-synchronizing coloring of G associated with β must be a congruence. Note, that there are at most

$\binom{k}{k_1}(k_1!)^2\binom{k-k_1}{k_2}(k_2!)^2\dots\binom{k_\ell}{k_\ell}(k_\ell!)^2$ automata in $\mathcal{A}(C)$ that have β as a congruence. Whereas the total number of automata in $\mathcal{A}(C)$ is $(k!)^2$. Thus, the fraction of non-synchronizing automata is at most $\frac{k_1!k_2!\dots k_\ell!}{k!}$. It is not hard to see, that this value is bounded by $\frac{1}{k}$. Since every coloring of G belongs to $\mathcal{A}(C)$ for some C, the result will follow.

Case II: for all **distinct** vetrices q, p belonging to the same block B of β and for all edges (q, q') and (p, p') we have that q' and p' belong to the same block of β. Thus, there is at least one singleton, i.e. a block of β consisting of a single vertex. Indeed, if it is not the case, then each of the blocks has a unique successor. It implies that there are no paths of the same length leading from vertices belonging to different blocks to some fixed vertex, so G is not primitive. Note, that β is a congruence for every coloring of G.

First, we will show that a coloring $\mathscr{A}$ of G is synchronizing if and only if the factor automaton $\mathscr{A}/\beta$ is synchronizing. Clearly, if $\mathscr{A}$ is synchronizing, then $\mathscr{A}/\beta$ is synchronizing too. Assume now that $\mathscr{A}/\beta$ has a synchronizing word u that brings it to a state i. Let j be the state belonging to a singleton block of β. Since G is primitive, there is a word v that brings i to j. It is not hard to see that the word uv is synchronizing for the automaton $\mathscr{A}$.

Secondly, we note that the factor automaton $\mathscr{A}/\beta$ is Eulerian with a prime number of states. Indeed, since β is a partition into blocks of equal weight, we conclude that $\mathscr{A}/\beta$ is Eulerian (see the proof of Theorem 8). If the number of states of $\mathscr{A}/\beta$ is not prime, then the partition β will not be unique. Now it remains to show that the synchronizing ratio of G/β is at least $\frac{k-1}{k}$. The proof of this fact is reminiscent of case I.

Note, that there exist a vertex q and edges $(q, r), (q, s)$ for $r \neq s$, otherwise G is not primitive. Let all the k outgoing edges of q be (q, p_1) of multiplicity k_1, (q, p_2) of multiplicity k_2, …, (q, p_ℓ) of multiplicity k_ℓ. Let us fix a coloring C of all edges, except for the outgoing edges of q. Let $\mathcal{A}(C)$ be the set of automata obtainable from C by all possible colorings of the remaining edges. In order to show that the synchronizing ratio of G is at least $\frac{k-1}{k}$ we will demonstrate that the fraction of non-synchronizing automata in $\mathcal{A}(C)$ is at most $\frac{1}{k}$.

If all automata in $\mathcal{A}(C)$ are synchronizing, then the statement holds true. Otherwise, let $\mathscr{A} \in \mathcal{A}(C)$ be a non-synchronizing automaton. Since the number of states is prime and the eigenvector of $\mathscr{A}$ is equal to $(1, 1, \dots, 1)$, by Theorem 6 we conclude that every letter of $\mathscr{A}$ acts as a permutation on the set of states. Note, that if edges (q, p_1) and (q, p_2) are labelled by x and y respectively, then the automaton $\mathscr{A}' \in \mathcal{A}(C)$ obtained by flipping the labels on these edges, i.e., assigning letter y to (q, p_1) and letter x to (q, p_2), is synchronizing. Indeed, either p_1 or p_2 is not equal to q. Without loss of generality we will assume that $p_1 \neq q$. Since every letter in $\mathscr{A}$ acts as a permutation, there exists a state r such that $r \cdot y = p_1$. Thus, $r \cdot y = q \cdot y$ for the automaton $\mathscr{A}'$ and it is synchronizing by Theorem 6. More generally, there are at most $k_1!k_2!\dots k_\ell!$ permutations of labels on the outgoing edges of q that keep the resulting automaton non-synchronizing. Since the value of the fraction $\frac{k_1!k_2!\dots k_\ell!}{k!}$ is bounded by $\frac{1}{k}$ we obtain the desired statement. ◀

5 The eigenvectors and the reset thresholds

The structure of the eigenvector an automaton $\mathscr{A}$ in accordance with some distribution can be utilized to bound the reset threshold of $\mathscr{A}$. To the best of our knowledge, the first such result was obtained by Kari [14]. He bounded the reset threshold of automata with the eigenvector $(1, 1, \dots, 1)$ in accordance with the uniform distribution, i.e., Eulerian automata.

▶ **Theorem 15.** *The reset threshold of an Eulerian automaton with n states is at most $n^2 - 3n + 3$.*

Afterwards, Steinberg noticed that the same bound holds true for automata with the eigenvector $(1, 1, \dots, 1)$ in accordance with *some* distribution [16]. Both of these results were later subsumed by the following theorem[3] of Berlinkov [4, Corollary 1].

▶ **Theorem 16.** *Let* $\mathbf{w}$ *be the sum of the coordinates of the integer eigenvector* $\vec{w}$ *of a strongly connected automaton* $\mathscr{A}$ *in accordance with some distribution. If* $\mathscr{A}$ *is synchronizing, then the reset threshold of* $\mathscr{A}$ *is at most* $1 + (n-1)(\mathbf{w} - 2)$.

Note, that the eigenvector of $\mathscr{A}$ in accordance with the uniform distribution depends only on $\mathcal{G}(\mathscr{A})$. Therefore, the bound given in this theorem will be valid for every recoloring of $\mathscr{A}$.

In this section we will present a simple reduction from an automaton $\mathscr{A}$ with the eigenvector $\vec{w}$ to an Eulerian automaton $\mathscr{B}$ with $\mathbf{w} = \sum_i \vec{w}[i]$ states such that $\mathrm{rt}(\mathscr{A}) \le \mathrm{rt}(\mathscr{B}) \le \mathrm{rt}(\mathscr{A}) + 1$. Thus, we will be able to utilize results of Kari about Eulerian automata to analyze $\mathscr{A}$. This reduction also gives a combinatorial interpretation of the aforementioned results by Steinberg and, to some extent, of Berlinkov.

▶ **Theorem 17.** *Let* $\mathbf{w}$ *be the sum of the coordinates of the integer eigenvector* $\vec{w}$ *of a strongly connected automaton* $\mathscr{A}$ *in accordance with a distribution* $p_1, p_2, \dots, p_k$*. If* $\mathscr{A}$ *is synchronizing, then there exists a synchronizing Eulerian automaton* $\mathscr{B}$ *with* $\mathbf{w}$ *states such that* $\mathscr{A}$ *is the factor automaton of* $\mathscr{B}$ *and* $\mathrm{rt}(\mathscr{A}) \le \mathrm{rt}(\mathscr{B}) \le \mathrm{rt}(\mathscr{A}) + 1$.

Proof. Let $\Sigma = \{a_1, a_2, \dots, a_k\}$ and $p_i = \frac{m_i}{\ell}$ for $1 \le i \le k$, where m_i, ℓ are positive integers. If there exists p_i such that $p_i \ne \frac{1}{k}$, then we will perform the next step, otherwise we proceed to step II.

Step I. We are going to duplicate certain letters of $\mathscr{A}$ in order to obtain an automaton $\mathscr{A}'$ such that its eigenvector in accordance with the uniform distribution is equal to $\vec{w}$. The alphabet of $\mathscr{A}'$ is equal to $\Sigma' = \{a_1^1, a_1^2, \dots, a_1^{m_1}, a_2^1, a_2^2, \dots, a_2^{m_2}, \dots, a_k^1, a_k^2, \dots, a_k^{m_k}\}$. The actions of these letters are as follows: for every i and j the action of the letter a_i^j in $\mathscr{A}'$ coincides with the action of the letter a_i in $\mathscr{A}$. It is easy to see that $\mathscr{A}'$ is synchronizing and $\mathrm{rt}(\mathscr{A}') = \mathrm{rt}(\mathscr{A})$. Furthermore, the eigenvector of $\mathscr{A}'$ in accordance with the uniform distribution coincides with $\vec{w}$. Thus, if $\vec{w} = (1, 1, \dots, 1)$, then $\mathrm{rt}(\mathscr{A}) = \mathrm{rt}(\mathscr{A}') \le n^2 - 3n + 3$ by Theorem 15. Therefore, this simple reduction gives an alternative way to obtain the result of Steinberg.

Step II. Now we are going to construct an Eulerian automaton $\mathscr{B}$ on a larger set of states and on a larger alphabet such that $\mathrm{rt}(\mathscr{A}) \le \mathrm{rt}(\mathscr{B}) \le \mathrm{rt}(\mathscr{A}) + 1$. Let $Q = \{1, \dots, n\}$ be the set of states of $\mathscr{A}'$ and Σ' be the alphabet of $\mathscr{A}'$. The set of states of $\mathscr{B}$ is equal to $\{(i, j) \mid i \in Q, 1 \le j \le \vec{w}[i]\}$. The alphabet of $\mathscr{B}$ is equal to $\Sigma' \cup \Lambda$, where Λ is a set of letters that we will define shortly. We will denote the set $\{(i, j) \mid 1 \le j \le \vec{w}[i]\}$ by S_i. Note that the number of states in $\mathscr{B}$ is equal to $\mathbf{w}$. Our construction of $\mathscr{B}$ will ensure the following properties:

1. $\mathscr{B}$ is Eulerian, and for every $i \in Q, x \in \Lambda$ we have $S_i \cdot x \subseteq S_i$;
2. for every i and $x \in \Sigma'$ we have $S_i \cdot x \subseteq S_{i \cdot x}$, where $i \cdot x$ is an image of i under the action of x in $\mathscr{A}'$.

Note, that these conditions imply that the partition $S_1, \dots, S_n$ is a congruence for the automaton $\mathscr{B}$, and the factor automaton with respect to this congruence is equal to the automaton $\mathscr{A}'$.

[3] In the original formulation of the theorem the bound is given in terms of the least common multiple L of the coordinates' denominators of the eigenvector $\vec{v}$ associated with the eigenvalue 1 such that $\sum_i \vec{v}[i] = 1$. Clearly, $\mathbf{w} = L$.

Let $k' = |\Sigma'|$ be the cardinality of Σ' and c_{ij} be the number of letters in Σ' that bring i to j in the automaton $\mathscr{A}'$. By the definition of $\vec{w}$ we have $\sum_{i \in Q} c_{ij}\vec{w}[i] = k'\vec{w}[j]$ for every j. Since $\vec{w}[i] = |S_i|$ we derive the equality $\sum_{i \in Q} c_{ij}|S_i| = k'|S_j|$ for every j. Due to the second property, $c_{ij}|S_i|$ is the total number of edges labelled by Σ' going from S_i to S_j in $\mathscr{B}$. Since the total number of incoming edges to S_j labelled by Σ' is equal to $k'|S_j|$, we can arrange them in such a way that every state of S_j has exactly k' incoming edges labelled by Σ'. We fix any such arrangement to define the action of Σ' on $\mathscr{B}$. Note, that the automaton $\mathscr{B}$ restricted to the alphabet Σ' is Eulerian.

The additional set of letters Λ is defined as follows. For every $i \in Q$ and every $j \in S_i$ we add a letter u_i^j. The action of u_i^j brings all states from S_i to j, and all the remaining states are fixed. Note, that the automaton $\mathscr{B}$ restricted to the alphabet Λ is Eulerian. Therefore, the first property is satisfied.

Step III. We will show now that the automaton $\mathscr{B}$ is synchronizing and $\mathrm{rt}(\mathscr{A}) \leq \mathrm{rt}(\mathscr{B}) \leq \mathrm{rt}(\mathscr{A}) + 1$. Let u be the shortest synchronizing word of $\mathscr{A}'$, and the action of u brings it to a state i. Since $\mathscr{A}'$ is the factor automaton of $\mathscr{B}$, we conclude that the automaton $\mathscr{B}$ is brought to S_i under the action of u. Thus, uu_i^i is a synchronizing word of $\mathscr{B}$ and $\mathrm{rt}(\mathscr{B}) \leq \mathrm{rt}(\mathscr{A}) + 1$.

Let u be a synchronizing word of $\mathscr{B}$, and the action of u brings it to a state in S_i for some i. Let v be a word over the alphabet Σ' obtained from u by removing all the letters from Λ. Since the action of every letter x from Λ of the automaton $\mathscr{B}$ satisfies $S_i \cdot x \subseteq S_i$ and $\mathscr{A}'$ is the factor automaton of $\mathscr{B}$, we conclude that u is a synchronizing word for the automaton $\mathscr{A}'$ and $\mathrm{rt}(\mathscr{A}) \leq \mathrm{rt}(\mathscr{B})$. ◀

▶ **Corollary 18.** *Let* $\mathbf{w}$ *be the sum of the coordinates of the integer eigenvector* $\vec{w}$ *of a strongly connected automaton* $\mathscr{A}$ *in accordance with a distribution* $p_1, p_2, \ldots, p_k$*. If* $\mathscr{A}$ *is synchronizing, then* $\mathrm{rt}(\mathscr{A}) \leq \mathbf{w}^2 - 3\mathbf{w} + 3$.

Clearly, this corollary is much weaker than Theorem 16. Nevertheless, both of these statements give $O(n^2)$ bound when $\mathbf{w} = O(n)$. This case is the most typical application of Theorem 16. At the same time, we believe that such simple reduction to an Eulerian automaton is of interest by itself.

To conclude this section we will estimate entries of the eigenvector of an arbitrary digraph. In general, they can be exponential in terms of the number of vertices. Consider the following k-out-regular digraph $U_{n,k}$. The set of vertices is equal to $\{0, 1, 2, \ldots, n-1\}$. For each $0 \leq i \leq n-1$ there is an edge $(i, 0)$ of multiplicity $k-1$ and an edge $(i, i+1 \mod n)$. It is easy to verify that the integer eigenvector of $U_{n,k}$ is $(k^{n-1}, k^{n-2}, \ldots, k, 1)$. Thus, the upper bound given by Theorem 16 can be exponential in n.

▶ **Proposition 19.** *Let* G *be a primitive* k*-out-regular digraph with* n *vertices. The entries of the eigenvector* $\vec{w}$ *are at most* $(2k^2)^{\frac{n-1}{2}}$.

Proof. Let A be the adjacency matrix of G, i.e., $A[i,j] = 1$ if there exists an edge going from i to j and $A[i,j] = 0$ otherwise. According to the definition of the eigenvector $\vec{w}$ in Section 2 we have the equality $\vec{w}(\frac{1}{k}A) = \vec{w}$. After rearrangement we get $\vec{w}(A - kI) = 0$, where I is the identity matrix. By Perron-Frobenius theorem we conclude that the rank of $A - kI$ is equal to $n-1$, since the eigenspace associated with $\vec{w}$ is one-dimensional. The main result of [7] states that for every integer matrix M of rank r if a system of linear equations $Mx = 0$ admits a nontrivial non-negative integer solution, then there exists such solution with entries bounded by the maximum of the absolute values of the $r \times r$ minors of M.

Thus, we conclude that there exists a non-negative integer vector $\vec{w'}$ such that $\vec{w'}(A - kI) = 0$ and entries of $\vec{w'}$ are bounded by the maximum of the absolute values of the $(n-1) \times (n-1)$

minors of $A - kI$. Note, that the Eucledean norm of each row of every minor is at most $\sqrt{2k^2}$, since the absolute of the entries is at most k and their sum is at most $2k$. Thus, by the Hadamard's inequality for the determinant we obtain an upper bound $(2k^2)^{\frac{n-1}{2}}$ on the minors. Since the non-negative vector $\vec{w'}$ is an eigenvector of A associated with the largest eigenvalue, we immediately conclude that $\vec{w'}$ is positive by the Perron-Frobenius theorem. ◀

Acknowledgements. We want to sincerely thank François Gonze and Jarkko Kari for the helpful discussions.

References

1 R. L. Adler, L. W. Goodwyn, and B. Weiss. Equivalence of topological Markov shifts. *Israel Journal of Mathematics*, 27(1):49–63, 1977.

2 D. S. Ananichev, M. V. Volkov, and V. V. Gusev. Primitive digraphs with large exponents and slowly synchronizing automata. *Journal of Mathematical Sciences*, 192(3):263–278, 2013.

3 M.-P. Béal and D. Perrin. *Handbook of Formal Languages: Volume 2. Linear Modeling: Background and Application*, chapter Symbolic Dynamics and Finite Automata, pages 463–506. Springer Berlin Heidelberg, 1997.

4 M. V. Berlinkov. Synchronizing Automata on Quasi-Eulerian Digraph. In *Implementation and Application of Automata*, volume 7381 of *LNCS*, pages 90–100. Springer, 2012.

5 M. V. Berlinkov. Synchronizing Quasi-Eulerian and Quasi-one-cluster Automata. *International Journal of Foundations of Computer Science*, 24(6):729–745, 2013.

6 M. V. Berlinkov and M. Szykuła. Algebraic synchronization criterion and computing reset words. In *Mathematical Foundations of Computer Science*, volume 9234 of *LNCS*, pages 103–115. Springer, 2015.

7 I. Borosh. A sharp bound for positive solutions of homogeneous linear Diophantine equations. *Proc. Amer. Math. Soc.*, 60:19–21 (1977), 1976.

8 A. Carpi and F. D'Alessandro. Independent sets of words and the synchronization problem. *Advances in Applied Mathematics*, 50(3):339–355, 2013.

9 J. Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied*, 14(3):208–216, 1964. In Slovak.

10 K. Culik, J. Karhumäki, and J. Kari. A note on synchronized automata and road coloring problem. In *Developments in Language Theory*, volume 2295 of *LNCS*, pages 175–185. Springer, 2002.

11 J. Friedman. On the Road Coloring Problem. *Proceedings of the American Mathematical Society*, 110(4):1133–1135, 1990.

12 V. V. Gusev and E. V. Pribavkina. Reset thresholds of automata with two cycle lengths. *International Journal of Foundations of Computer Science*, 26(07):953–966, 2015. `doi:10.1142/S0129054115400080`.

13 V. V. Gusev and M. Szykuła. On the number of synchronizing colorings of digraphs. In *Implementation and Application of Automata*, volume 9223 of *LNCS*, pages 127–139. Springer, 2015.

14 J. Kari. Synchronizing finite automata on Eulerian digraphs. *Theoretical Computer Science*, 295(1-3):223–232, 2003.

15 C. D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

16 B. Steinberg. The averaging trick and the Černý conjecture. *International Journal of Foundations of Computer Science*, 22(7):1697–1706, 2011.

17 B. Steinberg. The Černý conjecture for one-cluster automata with prime length cycle. *Theoretical Computer Science*, 412(39):5487–5491, 2011.

18 A. N. Trahtman. The Road Coloring Problem. *Israel Journal of Mathematics*, 172(1):51–60, 2009.

19 M. V. Volkov. Synchronizing automata and the Černý conjecture. In *Language and Automata Theory and Applications*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.

Competitive Packet Routing with Priority Lists

Tobias Harks[1], Britta Peis[2], Daniel Schmand[3], and Laura Vargas Koch[4]

1 **Department of Mathematics, University of Augsburg, Germany**
`tobias.harks@math.uni-augsburg.de`
2 **School of Business and Economics, RWTH Aachen University, Germany**
`britta.peis@oms.rwth-aachen.de`
3 **School of Business and Economics, RWTH Aachen University, Germany**
`daniel.schmand@oms.rwth-aachen.de`
4 **School of Business and Economics, RWTH Aachen University, Germany**
`laura.vargas@oms.rwth-aachen.de`

Abstract

In competitive packet routing games, packets are routed selfishly through a network and scheduling policies at edges determine which packages are forwarded first if there is not enough capacity on an edge to forward all packages at once. We analyze the impact of *priority lists* on the worst-case quality of pure Nash equilibria. A priority list is an ordered list of players that may or may not depend on the edge. Whenever the number of packets entering an edge exceeds the inflow capacity, packets are processed in list order. We derive several new bounds on the price of anarchy and stability for global and local priority policies. We also consider the question of the complexity of computing an optimal priority list. It turns out that even for very restricted cases, i.e., for routing on a tree, the computation of an optimal priority list is APX-hard.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, G.1.6 Optimization, G.2.2 Graph Theory

Keywords and phrases Packet Routing, Nash equilibrium, Price of Anarchy, Priority Policy, Complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.49

1 Introduction

A fundamental combinatorial optimization problem that has received considerable attention in the past (cf. [4, 14, 17, 18, 21, 24]) is *packet routing* in graphs. We are given a set of packets, which may, for example, correspond to unit-sized messages/bits in a communication network. Originated at possibly different start vertices, the goal is to transfer all packets as fast as possible to their respective destination vertices. It is assumed that each edge is equipped with a capacity (or bandwidth) and a travel time. A prominent variant is *discrete store-and-forward packet routing*, where every vertex can store arbitrarily many packets, but only a limited number can enter an edge simultaneously at each discrete time step, see [17]. Applications can be found in routing models used in synchronized systems with a given clock-rate. Imagine, for example, a chip with components and wires corresponding to the nodes and edges, respectively, of the associated graph, and with a centralized clock rate for the chip given by a crystal oscillator.

In this work, we focus on *selfish* or *competitive* packet routing using the discrete store-and-forward packet routing model. We are given a multi- or single-commodity network, where the commodities are specified by a source and a sink vertex and represent the players that route rational and selfishly one packet from their source to their sink through the network. Each edge of the network is endowed with an integral travel time and an integral capacity. The capacity of the edge defines the number of players that may enter the edge simultaneously. For each edge, we are given a priority list (i.e., an ordered list of the players) to resolve conflicts whenever more than capacity many players seek to enter that edge at the same point in time. All players are ready to start right from the beginning (i.e., there are no release dates) and aim to minimize their respective arrival time at the sink. Since the outcome of this competitive situation intrinsically depends on the priority lists employed on the edges, the problem of finding *good* priority lists renders into a coordination mechanism design problem. See [6] for the first landmark paper and several follow ups [1, 5, 7, 12].

1.1 Our Contribution

In this paper, we further explore properties of selfish discrete store-and-forward packet routing with *priority based* scheduling policies. We consider local priority lists and global priority lists (that may or may not depend on the edge). We obtain the following results.

Price of Anarchy/Stability

For global priority lists and multiple source-sink instances we show that the price of stability (PoS for short) behaves as $PoS \in \Omega(\sqrt{n})$ and the price of anarchy (PoA) is $PoA \in \mathcal{O}(n^3)$, where n denotes the number of players. For global priority lists and symmetric games (that is, all packets travel from a common source to a common sink) the PoS is one, while the PoA for these games is exactly $^{(n+1)}/_2$ and this bound holds even for multiple sources and a single sink.

For general local priority lists (that is, the predefined order may be different among edges) and asymmetric multi-commodity games we derive that the PoA is in between $^T/_4$ and $4T^2$, where T is a kind of *dilation* of the graph, i.e., the maximal length of a path, where edges with travel time 0 contribute 1 to the path. This result is obtained via adapting the primal-dual technique introduced by Kulkarni and Mirrokni [16]. Note that the network model in [16] is different to ours in the sense that it allows for different weights and sizes of the packets. See Section 1.2 for comparison. While the result in [16] holds for a very specific local scheduling policy only, namely, *Highest Density First*, our result even applies to *arbitrary* local priority lists, due to the special network structure in our model. As a byproduct of applying the primal-dual technique, we obtain the same bounds even for correlated equilibria which are guaranteed to exist.

Computational Complexity

We turn to the question of computing *optimal* priority lists, that is, priority lists that induce best possible social optima or Nash equilibria. We show that even for the special case of tree graphs, the resulting problem is APX-hard. Note that this is the first hardness result for the underlying coordination mechanism design problem and complements several approximability results for the tree case recently derived by Bhattacharya et al. [2]. Technically, we adapt a construction of Peis et al. [21], where it is shown that the problem to compute a schedule minimizing the makespan (the latest arrival of any packet) is APX-hard. Our result implies that the problem of defining global as well as local priority lists for minimizing the cost of any Nash equilibrium or of any social optimum is APX-hard.

We finally derive several further hardness results for our model: In multi-commodity games with local priority lists it is NP-hard to compute a pure Nash equilibrium. Moreover, it is NP-hard to compute a best response in symmetric games with local priority lists. These results are obtained by adapting the reduction described in Hoefer et al. [11] used to prove hardness in a more general setting. For global priority lists, we get an efficient Dijkstra-type algorithm for computing a best response and thus a pure Nash equilibrium.

1.2 Related work

Competitive Routing over Time

Hoefer et al. [11] considered weighted network congestion games in the continuous-time setting. In their model, the edges represent machines with predefined speed. Each job has a weight and the time needed to traverse an edge is given by the product of speed and weight. In contrast to our model, traversal times in [11] might be rational, but zero travel times are not allowed. Furthermore, the type of capacity constraints is different: While [11] capacitates the total weight of players using an edge at the same point in time, our model capacitates the number of players that may enter an edge simultaneously. Hoefer et al. analyzed four different scheduling policies: *FIFO*, *non-preemptive global ranking*, *preemptive global ranking* and *fair Time-Sharing*. They showed that in the case of a global priority list at least one equilibrium exists and it can be computed efficiently by iteratively and greedily routing the players with respect to the global order. Further results include the non-existence of equilibria for the FIFO scheduling policy and the complexity of computing equilibria and best responses.

The model of Kulkarni and Mirrokni [16] is a generalization of the model of Hoefer et al. [11] with two main differences. First, each packet has a size as well as a weight. The size determines how long it takes to traverse an edge and the weight denotes the contribution of this packet to the social cost. Observe that the authors also exclude edges with traversal time zero for a packet by their definition of the processing time as size divided by speed of an edge. The second difference to our model is that Kulkarni and Mirrokni assume that the strategy space of a packet is a subset of the simple paths from its source to its sink, whereas it is the set of all simple source-sink paths in [11] and in our model. Kulkarni and Mirrokni consider a variant of the robust price of anarchy, which is the worst-case ratio of the social cost of a coarse-correlated equilibrium and that of a social optimum [16]. A general framework to bound the robust price of anarchy via LP-Duality or Fenchel Duality was introduced by [16]. For the *Highest Density First* scheduling policy, they derive an upper bound of $4D^2$ for the robust price of anarchy, where D is the dilation of the graph. They also show a lower bound of $D/16$.

Flows over Time

Non-competitive packet routing can be interpreted as a special variant of *flows over time* (also known under the name *dynamic flows*) as introduced by Ford and Fulkerson in their seminal paper [10]. In fact, (non-competitive) packet routing is exactly the problem to find an integral multi-commodity flow satisfying unit demands of minimum time horizon. For an introduction to flows over time we refer to Skutella [23].

Koch and Skutella introduced in [15] a game-theoretic variant of flows over time. In their model, a continuum of players routes selfishly from a source to a sink through a network and flow enters an edge in a continuous fashion. They showed the existence of equilibria and analyzed the price of anarchy for their model, see also Cominetti et al. for a constructive proof for the existence and uniqueness of equilibria in [8]. Koch et al. [13] introduced discrete time steps for a single-commodity model. One could see our work as an extension to unsplittable flow particles and multi-commodity networks.

2 Preliminaries

An instance of a competitive packet routing game is a tuple (N, G, u, τ, π) consisting of a directed graph $G = (V, E)$ with integral travel times $\tau_e \in \mathbb{Z}_0^+$ denoting the time needed to traverse an edge $e \in E$. Additionally, each edge $e \in E$ is endowed with a capacity $u_e \in \mathbb{Z}^+$ denoting the number of players that can enter an edge e simultaneously at *each* integral time step. Note that this is independent of the travel times, even for $\tau_e = 0$. The set of players is denoted by $N = \{1, \ldots n\}$. Each player $i \in N$ is associated with a source s_i and sink t_i, inducing a strategy space $\mathcal{P}_i \subseteq 2^E$ consisting of all possible simple paths in G linking the respective source and sink. We call an instance *symmetric* if all players start at the same node and have the same sink, i.e. $s_i = s_j$ and $t_i = t_j$ for all players $i, j \in N$.

Depending on the chosen strategies there might be more than u_e-many players seeking to enter an edge at the same integral time step $\theta \in \mathbb{Z}_0^+$. In our model, we are given priority lists $\pi_e : N \to \{1, \ldots, n\}$ on each $e \in E$ to resolve such conflicts: among those players seeking to traverse edge e at time $\theta \in \mathbb{Z}_0^+$, the u_e players of highest priority according to list π_e may enter and travel along edge e, while the remaining players need to wait (at least) one time step. A priority list $\pi = (\pi_e)_{e \in E}$ is called *global* if $\pi_e = \pi_{e'}$ for all edges e and e', otherwise it is called local.

Given an instance (N, G, u, τ, π), each player $i \in N$ selects one path P_i from $\mathcal{P}_i$ with the goal to minimize the time when its packet entirely reaches its sink t_i. This time not only depends on the length $\tau(P_i) = \sum_{e \in P_i} \tau_e$ of path P_i, but also on the time the packet needs to wait at intermediate nodes due to interferences with players of higher priority. Given a strategy profile or state $P = (P_1, \ldots, P_n)$, we denote by $C_i(P)$ [or C_i if the context is clear] the time needed for player i's packet to entirely reach sink t_i. We define

$$C_i(P) = \left(\sum_{e \in P_i} \tau_e + w_{i,e}(P) \right), \tag{1}$$

where $w_{i,e}(P)$ is the waiting time for player i on the entry of edge e under profile P. The *social cost* of state $P = (P_1, \ldots, P_n)$ is the sum of all players' costs, i.e., $C(P) = \sum_{i \in N} C_i(P)$. We call a profile P *(socially) optimal* if it minimizes the social cost $C(P)$ over the set of all possible profiles. State P is a *pure Nash equilibrium (PNE)* if $C_i(P) \leq C_i(P_{-i}, P_i')$ holds for each player $i \in N$ and each alternative strategy $P_i' \in \mathcal{P}_i$. Here, as usual, state (P_{-i}, P_i') is obtained from P by replacing strategy P_i with P_i'.

In the definition of the arrival time of player i (cf. (1)), we implicitly assumed that the values $C_i(P), i \in N$ for a given state $P = (P_1, \ldots, P_n)$ are actually well-defined. We show in the following example that directed cycles of length 0 might be harmful:

▶ **Example 2.1.** Consider the graph depicted in Figure 1. We are given 3 players and player i travels from s_i to t_i for $i \in \{1, 2, 3\}$. In this example, each player has exactly one strategy and the priority lists on the edges e_1, e_2 and e_3 are given as $\pi_{e_1} = \{2, 3\}$, $\pi_{e_2} = \{3, 1\}$, $\pi_{e_3} = \{1, 2\}$. The travel times on the edges e_1, e_2, e_3 are equal to zero, where all other edges have travel time 1. The capacities of all edges are equal to 1. Now, there is no feasible integral flow over time respecting both the capacity constraints as well as the priority lists. Therefore, it is not possible to map each player $i \in N$ to a real-valued arrival time $C_i(P)$.

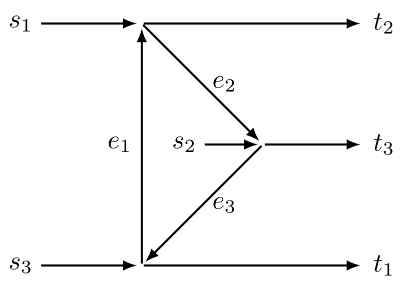


Figure 1 A graph showing that the mapping of paths to arrival times is not necessarily well-defined.

This observation motivates the exclusion of directed 0-cycles, that is, cycles C of length $\tau(C) = \sum_{e \in C} \tau_e = 0$. We assume in the following that there are no paths $P \in \mathcal{P}_i$ with total travel time equal to 0. Under this assumption, the following proposition shows that, given any strategy profile P, the embedding of players to arrival times $C_i(P), i \in N$, is well-defined, as long as directed 0-cycles are excluded.

▶ **Proposition 2.2.** *Given an instance without directed 0-cycles, we can use a Dijkstra-like algorithm to map given paths $P = (P_1, \dots, P_n)$ to a flow over time, thus to arrival times $C_i(P)$ in polynomial time.*

Proof. The idea is to adapt Dijkstra's algorithm [9] as follows. For each vertex v, we additionally define a list containing the arrival times of the players at v by the following procedure: initialize all source nodes s_i with arrival time 0 for every player starting at s_i. Use a priority queue of vertices sorted by the earliest arrival time of any player at that vertex. In each step, extract all vertices of minimal arrival time from the queue. Consider the graph H induced by these vertices and the corresponding edges of length $\tau_e = 0$. Repeatedly choose a vertex without incoming edges. Note that such a vertex needs to exist, since graph G is assumed to be free of 0-cycles . For each chosen vertex, route all players being able to depart according to the priority lists of the outgoing edges and delete the arrival time of the routed players. Add the arrival time of the routed players to the next vertex on their paths and reintroduce the vertex into the priority queue, if necessary. If the vertex is already in the priority queue, we possibly change its order in the queue. Now, delete the current vertex from H and go on with the next vertex without incoming edge. If H is empty continue with the next vertices from the queue. For a formal description of the algorithm, we refer to the full version. ◀

Using the relationship between packet routing and integral flows over time [23], we show that at least in the symmetric setting, a social optimum can be computed via Earliest Arrival Flows.

▶ **Definition 2.3.** Let $s, t \in V$. An *integral s-t-flow over time* is a set of functions $f_e : \mathbb{N} \to \mathbb{N}$ for all $e \in E$ satisfying the following two constraints:

$$f_e(\theta) \leq u_e \qquad \forall e \in E, \theta \in \mathbb{N} \tag{2}$$

$$\sum_{e \in \delta^-(v)} \sum_{\theta=0}^{\xi - \tau_e} f_e(\theta) \geq \sum_{e \in \delta^+(v)} \sum_{\theta=0}^{\xi} f_e(\theta) \qquad \forall \xi \in \mathbb{N}, v \in V \setminus \{s, t\} \tag{3}$$

The first inequality constrains the capacity and the second one represents weak flow conservation. If (3) is fulfilled with equality, the flow is said to satisfy strong flow conservation. An

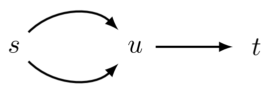


Figure 2 A network without pure Nash equilibrium.

integral s-t-flow over time fulfills the *earliest arrival property* if it maximizes the amount of flow arriving at the sink t for every integral time step. An integral s-t-flow over time that fulfills the earliest arrival property is called an *integral earliest arrival s-t-flow.*

▶ **Lemma 2.4** (Wilkinson [25]). *An integral earliest arrival s-t-flow with strong flow conservation can be computed in pseudo-polynomial time by an adapted successive shortest path algorithm.*

Wilkinson's algorithm combines the ideas of the push-and-relabel algorithm and the successive shortest path algorithm. During the execution of the algorithm one can easily transfer an integral earliest arrival s-t flow x into a strategy profile P with the following properties. The paths P_i are cycle-free, there are no waiting times at intermediate nodes and the demands are fulfilled. We call such a path profile an *earliest arrival state.*

▶ **Lemma 2.5.** *An earliest arrival state in a symmetric network corresponds to a social optimum, and vice versa.*

Proof. Let P be a social optimum and P' be an earliest arrival state. Let $N_\theta^P = |\{i : C_i(P) \leq \theta\}|$ denote the number of players with arrival time less than or equal to θ. Note that inequality $\sum_{i\in N} C_i(P) \leq \sum_{i\in N} C_i(P')$ can be equivalently written as $\sum_{\theta\in\mathbb{Z}_+}\left(N_\theta^P - N_{\theta-1}^P\right)\theta \leq \sum_{\theta\in\mathbb{Z}_+}\left(N_\theta^{P'} - N_{\theta-1}^{P'}\right)\theta$. Adding $(N_{\theta-1}^P - N_\theta^{P'})\theta$ on both sides of the inequality yields

$$\sum_{\theta\in\mathbb{Z}_+}\left(N_\theta^P - N_\theta^{P'}\right)\theta \leq \sum_{\theta\in\mathbb{Z}_+}\left(N_{\theta-1}^P - N_{\theta-1}^{P'}\right)\theta = \sum_{\theta\in\mathbb{Z}_+}\left(N_\theta^P - N_\theta^{P'}\right)\theta + \sum_{\theta\in\mathbb{Z}_+}\left(N_{\theta-1}^P - N_{\theta-1}^{P'}\right).$$

It follows that $\sum_{\theta\in\mathbb{Z}_+}\left(N_{\theta-1}^P - N_{\theta-1}^{P'}\right) \geq 0$. On the other hand, $N_{\theta-1}^P - N_{\theta-1}^{P'} \leq 0$ for all θ due to the earliest arrival property of P'. As a consequence, $N_\theta^P = N_\theta^{P'}$ for all θ. This completes the proof. ◀

We conclude that we can compute a social optimum via an adapted successive shortest path algorithm.

▶ **Proposition 2.6.** *A social optimum of a symmetric competitive packet routing instance is computable via an adapted version of the successive shortest path algorithm even for local priority lists.*

While this proposition shows that a socially optimal profile can be computed via a successive shortest path algorithm in symmetric games, we prove in Section 4 that the computational complexity of computing the social optimum or the socially optimal Nash equilibrium becomes APX-hard in non-symmetric games, even when restricted to global priority lists.

The following example shows that PNE do not necessarily exist even in very simple two-player games:

Table 1 The inefficiency of packet routing games with a global scheduling policy.

	Price of stability		Price of anarchy	
symmetric	1	Thm. 3.1	$\frac{n+1}{2}$	Thm. 3.2
multi-commodity	$\Omega(\sqrt{n})$	Prop. 3.3	$O(n^3)$	Prop. 3.4

▶ **Example 2.7.** Consider the network shown in Figure 2 with three vertices $V = \{s, u, t\}$, two parallel edges e_1, e_2 linking s and u, and one edge e_3 linking u and t, all edges of unit capacity and unit travel time. Suppose we are given two players, both with source s and sink t. Now, if the priorities are chosen such that player 1 has priority on the two s-leaving edges, i.e., $\pi_{e_1} = \pi_{e_2} = (1, 2)$, and player 2 has priority on the t-entering edge, i.e., $\pi_{e_3} = (2, 1)$, then the resulting packet routing game does not admit an equilibrium. In this game player 1 tries to choose the same path as player 2 and player 2 always chooses the free path. It follows that already this very simple two-player symmetric (i.e., single-source-single-sink) game does not admit an equilibrium.

We have seen that not all priority lists guarantee pure Nash equilibria in competitive packet routing games. However, for games with global priority lists a pure Nash equilibrium can be guaranteed to exist by adding players one by one according to the priority list, see Hoefer et al. [11].

Certainly, the social cost of a profile highly depends on the chosen priority lists $\pi_e, e \in E$. In the full version we show that the restriction to global priority lists might in fact lead to higher social cost and that this gap might be arbitrarily large.

3 Inefficiency of Nash equilibria

In this section, we examine the inefficiency of Nash equilibria. For the case of global priority lists, we find tight bounds for the PoS and PoA for symmetric games, a multi-commodity game with PoS in the order of $\sqrt{n}$ and an upper bound for the PoA in the multi-commodity case. An overview of the results for global priority lists is depicted in Table 1.

For local priority lists it turns out that it is much harder to find bounds on the price of anarchy and price of stability. We use a technique introduced in [3] and [16] to prove that the PoA is between $T/4$ and $4T^2$, where T is some kind of dilation of the graph.

3.1 Global Priority Lists

▶ **Theorem 3.1.** *In symmetric competitive packet routing games with global priorities, the price of stability is equal to 1.*

Proof. Consider a symmetric instance with global priority list π. Note that the choice of the priority list does not matter since all players have the same start and target node. Up to renaming, suppose $\pi = \{1, ..., n\}$. Observe that a social optimum P fulfills the earliest arrival property due to Lemma 2.5. Note that $C_i(P) \leq C_j(P)$ whenever $i \leq j$. For the sake of contradiction, suppose there exists at least one player with an improving move. Among all players with an improving move, we choose one of smallest index, say k. If k improves her strategy by switching from path P_k to P'_k, the arrival time of player k decreases, while the arrival time of players $\{1, ..., k-1\}$ stays the same. However, this is a contradiction to the fact that a socially optimal profile admits the earliest arrival property. ◀

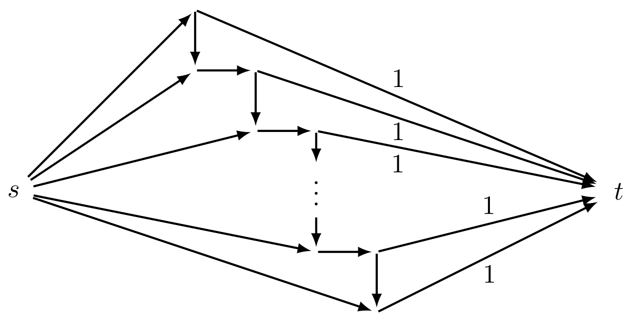


Figure 3 A game with price of anarchy equal to $(n+1)/2$.

Theorem 3.1 shows that there always exists a socially optimal Nash equilibrium, as long as we restrict to global priority lists and symmetric games. In the following, we show that the price of anarchy for this case is exactly $(n+1)/2$.

▶ **Theorem 3.2.** *For all symmetric competitive packet routing games with global priority lists and n players, the price of anarchy is upper bounded by $1 + (n-1)/2k$, this bound is tight for $k = 1$.*

Proof. We first prove the upper bound. Let $S = (S_1, \dots, S_n)$ be any Nash equilibrium. We start by showing that the cost of the Nash equilibrium S is at most

$$C(S) \le n \cdot k + \frac{n^2}{2} - \frac{n}{2},$$

for k being the length of a shortest $s-t$-path with respect to the travel times τ_e. Since $\sum_{i=1}^{n} (k + i - 1) = n \cdot k + n^2/2 - n/2$, it suffices to show that the arrival time of the i-th player in the priority list does not exceed $k + i - 1$ time units. Suppose this is not the case, i.e. there are players arriving later. Clearly, the first player in the priority list needs k time units. Now, let j be the player who is the first one arriving late in the order of the priority list. It follows that player j has the following improvement move: she can start at source s and follow player 1. If she has to wait at the entry of any edge, she follows the player entering the edge directly ahead of her. Player j can only be delayed by players arriving on time. By using this strategy she arrives at sink t at the latest one time unit after the last player being able to delay her. Hence, she can guarantee to reach sink t by time $k + i - 1$.

Observe that no player can arrive before time unit k. So, the cost of the optimal solution can be lower bounded by nk. This yields the following bound.

$$\text{PoA} \le \frac{C(S)}{C(\text{OPT})} \le \frac{n \cdot k + \frac{n^2}{2} - \frac{n}{2}}{nk} = \frac{k - \frac{1}{2} + \frac{n}{2}}{k} = 1 + \frac{(n-1)}{2k},$$

This completes the proof for the upper bound on the PoA and is well defined since $k \ge 1$.

For showing the tightness of the result, note that $1 + (n-1)/2k = (n+1)/2$ for $k = 1$. Now, consider the example depicted by the graph of Figure 3. This Braess-like graph topology has been used before to show lower bounds on the PoA in other settings, e.g. see [16]. The travel times τ_e are depicted next to the edges, where edges without label have $\tau_e = 0$. We define $u_e = 1$ for all edges $e \in E$. Note that there are n direct paths (i.e., those without vertical edges), each with travel time 1. Thus, in an optimal solution, all players pick a direct path, resulting in an arrival time of 1 for each player, and an optimal social cost of n. However, the profile in which all players use the path containing all vertical edges is a Nash equilibrium as

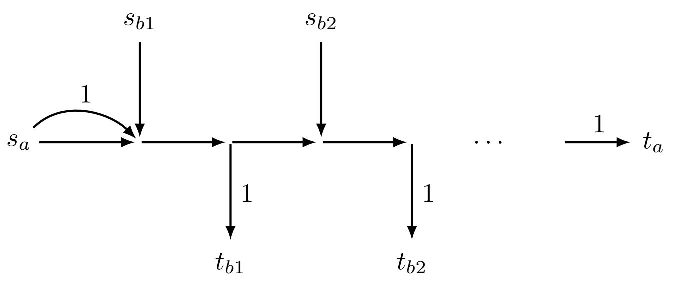


Figure 4 A game with price of stability in $\theta(\sqrt{n})$.

well. The i-th player in the order of the priority list arrives at time i and has no incentive to deviate, since there is no possible path she can use without waiting for all players of higher priority. Thus, there is a Nash equilibrium of social cost equal to $1 + 2 + \cdots + n = {n(n+1)}/{2}$. As a consequence, we obtain PoA $= \frac{n(n+1)}{2}/n = {(n+1)}/{2}$, which completes the proof. ◀

▶ Remark. Theorem 3.2 can be extended to networks with multiple sources and one sink.

For multi-commodity competitive packet routing games, earliest arrival flows do not necessarily exist [23]. It turns out that the price of stability might not even be constant in this more general setting. We provide an example where the price of stability is in $\theta(\sqrt{n})$.

▶ **Proposition 3.3.** *There is a multi-commodity competitive packet routing game with a global priority list and price of stability in $\theta(\sqrt{n})$.*

Proof. Consider the game illustrated by the graph in Figure 4. There are a players with source s_a and sink t_a, and b players each with individual source s_{bi} and sink t_{bi}, $i \in \{1, \dots, b\}$. All edges have unit capacity. The travel time is 0 if not depicted otherwise in Figure 4. We consider a global priority list in which all horizontal players have priority over the vertical players.

In the only Nash equilibrium, all horizontal players choose the direct path. Thus, their arrival times sum up to $\sum_{i=1}^{a} i$. The vertical players need to wait until all horizontal players have passed. Thus, their arrival times sum up to $b \cdot (a + 1)$. In an optimal solution, all horizontal players choose the longer path with the first edge of travel time 1. Therefore, their cost is $\sum_{i=1}^{a} i + 1$, while the cost of the vertical players decreases to b. Due to these considerations, the price of stability of this game can be expressed by

$$\text{PoS} = \frac{\frac{a^2}{2} + ba + b + \frac{a}{2}}{b + \frac{a^2}{2} + \frac{3a}{2}}.$$

If we replace b by $n - a$, we can differentiate the expression with respect to a. Taking the root we get $-2 + \sqrt{2n - 3}$. By substitution we get a price of stability which is in $\theta(\sqrt{n})$. ◀

In the following proposition, we give a bound on the price of anarchy by estimating the waiting times of the players. The proof can be found in the full version.

▶ **Proposition 3.4.** *The price of anarchy in a multi-commodity competitive packet routing game with a global priority list is upper bounded by $1 + {n^3}/{2} \in O(n^3)$.*

3.2 Local Priority Lists

All results presented so far deal with global priority lists. For local priority lists, we derive the following upper bound on the price of anarchy.

▶ **Theorem 3.5.** *In instances in which Nash equilibria exist, the price of anarchy is upper bounded by $4T^2$ for all priority lists, where $T = \max_{i\in N} \max_{P_j \in \mathcal{P}_i} \sum_{e\in P_j} \max\{\tau_e, 1\}$.*

In a related model of competitive routing games, Kulkarni and Mirrokni [16] prove a bound on the robust price of anarchy of $4D^2$, where D denotes the number of edges of the longest feasible path of any player. For the differences in the two models, see Section 1.2. Note that an instance of a competitive packet routing game, in the special case where no 0-travel times exist, can be fit into the model defined by Kulkarni and Mirrokni by replacing any edge of length τ_e by τ_e edges of length 1. After this procedure, the values T and D coincide.

In contrast to [16], our result holds for *arbitrary* local priority lists, whereas the result of [16] applies only to the *Highest-Density-First*-rule. Besides, our analysis turns out to be much simpler.

Proof. We start with the following preprocessing in order to ensure binary travel times and unit-capacities. For a given instance, we substitute each edge $e \in E$ of travel time $\tau_e > 1$ by τ_e-many edges of length one, each with the same priority lists $\pi(e)$. This results in an instance with travel times $\tau_e \in \{0,1\}$. Similar, we replace each edge of capacity $u_e > 1$ by u_e-many parallel edges of unit capacity. Now, consider the following linear program.

$$\begin{array}{lrcll}
\min & \sum_{i\in N}\sum_{P_j\in\mathcal{P}_i}\sum_{e\in P_j}\sum_{\theta\in\mathbb{Z}_+} x_{eij\theta}\cdot(\theta+\tau_e) & & & \\
\text{s.t.} & \sum_{P_j\in\mathcal{P}_i} x_{ij} & \geq & 1 & \forall i \in N\\
 & \sum_{\theta\in\mathbb{Z}_+} x_{eij\theta} & \geq & x_{ij} & \forall e\in E, i\in N, j: P_j\in\mathcal{P}_i\\
 & \sum_{i\in N}\sum_{P_j\in\mathcal{P}_i} x_{eij\theta} & \leq & 1 & \forall e\in E, \theta\in\mathbb{Z}_+\\
 & x_{ij}, x_{eij\theta} & \geq & 0 & \forall e\in E, \theta\in\mathbb{Z}_+, i\in N, j: P_j\in\mathcal{P}_i
\end{array}$$

We claim that the optimal LP-solution value is a T-approximation on the cost of a social optimum for the following reason. Given a socially optimal profile $P = (P_1, \dots, P_n)$ with completion times $C_i(P), i \in N$, assign $x_{ij} = 1$ if player i chooses path $P_j \in \mathcal{P}_i$, and $x_{ij} = 0$ otherwise. Furthermore, assign $x_{eij\theta} = 1$ if player i enters edge e on her selected path P_j at time step θ, and $x_{eij\theta} = 0$ otherwise. It is easy to check that this is a feasible solution for the LP. For bounding the objective function we consider $x_{eij\theta}\cdot\tau_e$ and $x_{eij\theta}\cdot\theta$ separately. Clearly, $x_{eij\theta}\cdot\theta \leq C_i(P)$ since a player has not yet arrived at her sink if she uses edge e. By definition of T, there are at most T edges on every path P_j. Additionally we consider $\min_{e,j,\theta}\{\theta | x_{eij\theta} = 1\}$ as the starting time of player i and get

$$\sum_{P_j\in\mathcal{P}_i}\sum_{e\in P_j}\sum_{\theta\in\mathbb{Z}_+} x_{eij\theta}\cdot\tau_e + \min_{e,j,\theta}\{\theta | x_{eij\theta}=1\} \leq C_i(P) \quad \text{and}$$

$$\sum_{P_j\in\mathcal{P}_i}\sum_{e\in P_j}\sum_{\theta\in\mathbb{Z}_+} x_{eij\theta}\cdot\theta - \min_{e,j,\theta}\{\theta | x_{eij\theta}=1\} \leq (T-1)C_i(P).$$

Thus, the optimal solution value of the LP is upper bounded by $\sum_{i\in N}(T-1)C_i(P)+C_i(P) = T\cdot\text{OPT}$, where $\text{OPT} = \sum_{i\in N} C_i(P)$ denotes the optimal social value of the competitive routing game.

Let us now modify the LP by dividing the right-hand side of the third constraint by $2T$. That is, we replace constraint $\sum_{i\in N}\sum_{P_j\in\mathcal{P}_i} x_{eij\theta} \leq 1$ by $\sum_{i\in N}\sum_{P_j\in\mathcal{P}_i} x_{eij\theta} \leq 1/2T$ for all $e \in E, \theta \in \mathbb{Z}_+$. Let LP^* denote the optimal objective value of the modified LP. By scaling the $x_{eij\theta}$ variables of the LP-solution as described above by $1/2T$ and repeating the original solution $2T$ times, we achieve a feasible solution for the modified LP which loses a factor of at most $2T$ in the objective compared to the prior solution. Hence, the modified LP is a $2T^2$-approximation on the cost of a socially optimal profile. That is, $LP^* \leq 2T^2 \cdot \mathrm{OPT}$.

In the remainder of the proof, we show that even the worst PNE has a social cost of at most $2 \cdot LP^*$ yielding the desired bound of $4T^2$ on the price of anarchy. Consider the dual of the modified LP:

$$\begin{array}{rrcll}
\max & \displaystyle\sum_{i\in N} \alpha_i - \frac{1}{2T}\sum_{e\in E}\sum_{\theta\in\mathbb{Z}_+}\beta_{e\theta} & & & \\
\text{s.t.} & \displaystyle\alpha_i - \sum_{e\in P_j}\nu_{eij} & \leq & 0 & \forall i \in N, j : P_j \in \mathcal{P}_i \\
& \nu_{eij} - \beta_{e\theta} & \leq & \theta + \tau_e & \forall e \in E, i \in N, j : P_j \in \mathcal{P}_i, \theta \in \mathbb{Z}_+ \\
& \alpha_i, \beta_{e\theta}, \nu_{eij} & \geq & 0 & \forall e \in E, i \in N, j : P_j \in \mathcal{P}_i, \theta \in \mathbb{Z}_+
\end{array}$$

By weak linear programming duality, we know that any feasible dual solution has objective value at most LP^*. It suffices to show that any pure Nash equilibrium $\bar{P}$ induces a feasible dual solution of objective value $1/2 \sum_{i\in N} C_i(\bar{P})$.

Let $\bar{P} = (\bar{P}_1, \dots, \bar{P}_n)$ be any PNE. Define a dual solution $\bar{\alpha}, \bar{\beta}, \bar{\nu}$ as follows. For each $i \in N$ let $\bar{\alpha}_i = C_i(\bar{P})$. Furthermore, for each $\theta \in \mathbb{Z}_+$ and $e \in E$, let $\bar{\beta}_{e\theta} = |\{i \in N \mid e \in \bar{P}_i,\ \theta \leq C_i(\bar{P})\}|$ denote the number of players who have not yet arrived at their sink at time θ under profile $\bar{P}$. Finally, let $\bar{\nu}_{eij} = \tau_e + w_{ie}(P_j, \bar{P}_{-i})$ denote the waiting time at the entry of e plus the traversing time τ_e in case player i switches from strategy $\bar{P}_i$ to P_j. Note that the first dual constraint is easily seen to be satisfied by this definition of the dual variables, since $\bar{\alpha}_i - \sum_{e\in P_j}\bar{\nu}_{eij} \leq 0$ is equivalent to $C_i(\bar{P}) \leq \sum_{e\in P_j}(\tau_e + w_{ie}(P_j, \bar{P}_{-i})) = C_i(P_j, \bar{P}_{-i})$, which follows by the definition of PNE. The second constraint $\bar{\nu}_{eij} - \bar{\beta}_{e\theta} \leq \theta + \tau_e$ can equivalently be written as $w_{ie}(P_j, \bar{P}_{-i}) \leq \theta + \bar{\beta}_{e\theta}$ which is certainly satisfied, since at each time step θ, any player i, after switching from $\bar{P}_i$ to path P_j, will never wait at the entry of any edge e longer than θ plus the total number of players which have not yet arrived at their sink at time θ. This concludes the proof. ◀

▶ Remark. We get an example with price of anarchy $T/4$ from the proof of Theorem 3.2.

▶ Remark. Since our proof goes along the same lines as the primal-dual proof technique of Kulkarni and Mirrokni [16], it is not hard to verify that our results also hold for coarse-correlated equilibria, which are guaranteed to exist. This includes the case of correlated equilibria and mixed Nash equilibria, see [22].

4 Computational Complexity

The problem to design either local or global priority lists to minimize the cost of a social optimum, or the cost of any Nash equilibrium, turns out to be APX-hard.

▶ **Theorem 4.1.** *Even in graphs that form a tree, i.e. every player has a pre-defined strategy, designing priority lists that minimize the social cost or the cost of a Nash equilibrium is APX-hard both in the case of global or local priority lists.*

Proof (Sketch). We describe the main ideas of the proof. More details can be found in the full version. Assume we are given an instance of 3-OCCURRENCE-MAX-3-SAT, i.e. a maximum satisfiability problem with n variables, each of which occuring at most 3 times, together with m clauses where each clause contains exactly 3 variables. Inspired by [21], we define an instance of a competitive packet routing game as follows. We design a tree graph with 8 players per variable and 3 players per clause such that the sum of the arrival times is equal to $34n + 6m + \#(\text{unsatisfied clauses})$. For each variable we introduce 4 variable-players and 4 dummy-players. The variable-players correspond to either the first or second occurrence of this variable, as a positive or negative literal, respectively. These players share an edge with one of the corresponding clause-players if the variable is the first or second occurrence of this variable as a positive or negative literal in the clause. The scheduling policies of the (fixed) starting edges of these players decode the variable assignment of the satisfiability problem. The 4 dummy-players per variable guarantee that the two players corresponding to the positive or negative occurrences leave at the same time, respectively. With this idea, we can decode the variable assignment in the satisfiability problem. We design the network and the travel times in such a way, that exactly one clause player and variable player meet, i.e. impose a waiting time of 1, if and only if the clause is not fulfilled by any variable. Thus maximizing the number of satisfied clauses in the satisfiability problem is equivalent to minimizing the social cost in the competitive routing instance. Since 3-OCCURRENCE-MAX-3-SAT is APX-hard (see [19, 20]), we derive APX-hardness for our problem. The variable-players and the topology of the graph are the same as in [21]. Due to the different objective function, we need to introduce the dummy players with their paths and conduct a different analysis. ◀

For computing best responses and Nash equilibria, the complexity status highly depends on the chosen priority list.

▶ **Proposition 4.2.** *In competitive packet routing games with local priority lists it is NP-hard to compute a best response for a player, even in the symmetric setting. Moreover, given a game with local priority lists, it is NP-hard to compute a Nash equilibrium if one exists.*

The proof of this proposition is based on a proof of [11]. The difference is that Hoefer et al. use the FIFO policy with a global tie breaking rule, where we use local priority policies. Their model allows to schedule a player with lower priority before another player by slight perturbation of the travel times. This does not work in our model due to the integral time steps and integral travel times, so we use local priority policies. In order to show the hardness for computing Nash equilibria we had to modify the graph and introduce an exclusive start-node for every player.

The following observation shows that we can compute best-responses and a Nash equilibrium in games with global priority policies. The main idea is to use a Dijkstra-like algorithm and the fact, that a player is never influenced by players with a lower priority. This idea has also been used in [11]. The proofs are obtained by extending the proofs of [11] step-by-step to edges with 0-travel times. We refer to the full version for further details.

▶ **Observation 4.3.** *For games with global priority policies there is a polynomial time algorithm to compute a Nash equilibrium and a best response of any given player.*

References

1 Yossi Azar, Lisa Fleischer, Kamal Jain, Vahab S. Mirrokni, and Zoya Svitkina. Optimal coordination mechanisms for unrelated machine scheduling. *Operations Research*, 63(3):489–500, 2015.

2 Sayan Bhattacharya, Janardhan Kulkarni, and Vahab S Mirrokni. Coordination mechanisms for selfish routing over time on a tree. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 186–197, 2014.

3 Vittorio Bilò. A unifying tool for bounding the quality of non-cooperative solutions in weighted congestion games. In *Approximation and Online Algorithms – 10th International Workshop, WAOA 2012, Ljubljana, Slovenia, September 13-14, 2012, Revised Selected Papers*, pages 215–228, 2012.

4 Costas Busch, Malik Magdon-Ismail, Marios Mavronicolas, and Paul Spirakis. Direct routing: Algorithms and complexity. *Algorithmica*, 45(1):45–68, 2006.

5 Ioannis Caragiannis. Efficient coordination mechanisms for unrelated machine scheduling. *Algorithmica*, 66(3):512–540, 2013.

6 George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. *Theoretical Computer Science*, 410(36):3327–3336, 2009.

7 Richard Cole, José R. Correa, Vasilis Gkatzelis, Vahab S. Mirrokni, and Neil Olver. Inner product spaces for minsum coordination mechanisms. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 539–548, 2011.

8 Roberto Cominetti, José R. Correa, and Omar Larré. Existence and uniqueness of equilibria for flows over time. In *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, pages 552–563, 2011.

9 Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

10 Lester R Ford Jr and Delbert Ray Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958.

11 Martin Hoefer, Vahab S. Mirrokni, Heiko Röglin, and Shang-Hua Teng. Competitive routing over time. *Theoretical Computer Science*, 412(39):5420–5432, 2011.

12 Nicole Immorlica, Li Erran Li, Vahab S Mirrokni, and Andreas S Schulz. Coordination mechanisms for selfish scheduling. *Theoretical Computer Science*, 410(17):1589–1598, 2009.

13 Ronald Koch, Ebrahim Nasrabadi, and Martin Skutella. Continuous and discrete flows over time. *Mathematical Methods of Operations Research*, 73(3):301–337, 2011.

14 Ronald Koch, Britta Peis, Martin Skutella, and Andreas Wiese. Real-time message routing and scheduling. In *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*, pages 217–230, 2009.

15 Ronald Koch and Martin Skutella. Nash equilibria and the price of anarchy for flows over time. *Theory of Computing Systems*, 49(1):71–97, 2011.

16 Janardhan Kulkarni and Vahab S. Mirrokni. Robust price of anarchy bounds via LP and fenchel duality. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1030–1049, 2015.

17 Frank Thomson Leighton, Bruce Maggs, and Satish Rao. Packet routing and job-shop scheduling in $\mathcal{O}$(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

18 Frank Thomson Leighton, Bruce Maggs, and Andrea W Richa. Fast algorithms for finding $\mathcal{O}$(congestion + dilation) packet routing schedules. *Combinatorica*, 19(3):375–401, 1999.

19 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

20 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.

21 Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In *Approximation and Online Algorithms, 7th International Workshop, WAOA 2009, Copenhagen, Denmark, September 10-11, 2009. Revised Papers*, pages 217–228, 2009.

22 Tim Roughgarden. Intrinsic robustness of the price of anarchy. *Journal of the ACM*, 62(5):32, 2015.

23 Martin Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization, Bonn Workshop on Combinatorial Optimization, November 3-7, 2008, Bonn, Germany*, pages 451–482, 2008.

24 Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing*, 30(6):2051–2068, 2001.

25 William L Wilkinson. An algorithm for universal maximal dynamic flows in a network. *Operations Research*, 19(7):1602–1612, 1971.

The Ground-Set-Cost Budgeted Maximum Coverage Problem

Irving van Heuven van Staereling[1], Bart de Keijzer[2], and Guido Schäfer[3]

1 **Centrum Wiskunde & Informatica (CWI), Networks and Optimization Group, Amsterdam, The Netherlands**
heuven@cwi.nl
2 **Centrum Wiskunde & Informatica (CWI), Networks and Optimization Group, Amsterdam, The Netherlands**
keijzer@cwi.nl
3 **Centrum Wiskunde & Informatica (CWI), Networks and Optimization Group, Amsterdam, The Netherlands; and Vrije Universiteit Amsterdam, Department of Econometrics and Operations Research, Amsterdam, The Netherlands**
schaefer@cwi.nl

Abstract

We study the following natural variant of the budgeted maximum coverage problem: We are given a budget B and a hypergraph $G = (V, E)$, where each vertex has a non-negative cost and a non-negative profit. The goal is to select a set of hyperedges $T \subseteq E$ such that the total cost of the vertices covered by T is at most B and the total profit of all covered vertices is maximized. Besides being a natural generalization of the well-studied maximum coverage problem, our motivation for investigating this problem originates from its application in the context of bid optimization in sponsored search auctions, such as Google AdWords.

It is easily seen that this problem is strictly harder than budgeted max coverage, which means that the problem is $(1 - 1/e)$-inapproximable. The difference of our problem to the budgeted maximum coverage problem is that the costs are associated with the covered vertices instead of the selected hyperedges. As it turns out, this difference refutes the applicability of standard greedy approaches which are used to obtain constant factor approximation algorithms for several other variants of the maximum coverage problem. Our main results are as follows:

- We obtain a $(1 - 1/\sqrt{e})/2$-approximation algorithm for graphs.
- We derive a fully polynomial-time approximation scheme (FPTAS) if the incidence graph of the hypergraph is a forest (i.e., the hypergraph is *Berge-acyclic*). We also extend this result to incidence graphs with a fixed-size feedback hyperedge node set.
- We give a $(1 - \varepsilon)/(2d^2)$-approximation algorithm for every $\varepsilon > 0$, where d is the maximum degree of a vertex in the hypergraph.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases maximum coverage problem, approximation algorithms, hypergraphs, submodular optimization, sponsored search

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.50

1 Introduction

In the *budgeted maximum coverage problem* we are given a hypergraph $G = (V, E)$ with a non-negative cost $c(e) \in \mathbb{R}_{\geq 0}$ for every hyperedge $e \in E$ and a non-negative profit $p(i) \in \mathbb{R}_{\geq 0}$

for every vertex $i \in V$, and a non-negative budget $B \in \mathbb{R}_{\geq 0}$. The goal is to select a set of hyperedges $T \subseteq E$ whose total cost is at most B such that the total profit of all vertices covered by the hyperedges in T is maximized.

This is a fundamental combinatorial optimization problem with many applications in resource allocation, job scheduling and facility location (see, e.g., [6] for examples). Feige [4] showed that this problem is not polynomial-time approximable within a factor of $(1 - 1/e)$ unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log \log n)})$, even if all hyperedges have unit cost. Khuller, Moss and Naor [9] derived a $(1 - 1/e)$-approximation algorithm for the budgeted maximum coverage problem (which is the best possible). Their algorithms are based on a natural greedy approach in combination with a standard enumeration technique. Similar approaches were used to derive constant factor approximation algorithms for several other variants and generalizations of the maximum coverage problem.

In this paper, we study the following natural variant of the budgeted maximum coverage problem, which we call the *ground-set-cost budgeted maximum coverage problem (GBMC)*: We are given a hypergraph $G = (V, E)$ with a non-negative cost $c(i) \in \mathbb{R}_{\geq 0}$ and a non-negative profit $p(i) \in \mathbb{R}_{\geq 0}$ for every vertex $i \in V$, and a non-negative budget $B \in \mathbb{R}_{\geq 0}$. For a subset $T \subseteq E$, define $c(T) = \sum_{i \in \cup T} c(i)$ and $p(T) = \sum_{i \in \cup T} p(i)$ as the total cost and profit, respectively, of all vertices covered by the hyperedges in T.[1] Our goal is to select a set of hyperedges $T \subseteq E$ such that the total cost $c(T)$ of all covered vertices is at most B and the total profit $p(T)$ of all covered vertices is maximized. To the best of our knowledge, this problem has not been studied before.

Note that a crucial difference here is that in our problem costs are incurred per covered vertex, while in the budgeted maximum coverage problem costs are incurred per selected hyperedge. Albeit seemingly minor, this change makes the problem much harder to tackle algorithmically. More specifically, most greedy approaches (which give rise to constant factor approximation guarantees for several variants of the maximum coverage problem) turn out to be inapplicable in our setting because of the following reason: The basic idea underlying these greedy approaches is to select in each iteration a hyperedge that is most *cost-efficient*, i.e., maximizes the ratio of the profit of newly covered vertices over the cost of selecting the hyperedge. A property that is crucially exploited in the analysis of these algorithms is that the cost for selecting a hyperedge is constant, i.e., its cost-efficiency can only decrease throughout the course of the algorithm (as more of its vertices get covered). However, this monotonicity property is no longer guaranteed in our setting because the cost for picking a hyperedge depends on the set of already covered vertices. In fact, it is not hard to see that the cost-efficiency of a hyperedge can change arbitrarily from one iteration to the next.

Our motivation for investigating the vertex-cost budgeted maximum coverage problem is two-fold: (i) It is a generalization of the well-studied maximum coverage problem and a natural variant of the budgeted maximum coverage problem. (ii) It is a fundamental combinatorial optimization problem having several applications in practice. Of particular importance is its relation to the problem of computing optimal bids in sponsored search auctions such as Google AdWords (details will be given in the full version of the paper).

Our contributions

The contributions presented in this paper are as follows:

1. We obtain a $(1 - 1/\sqrt{e})/2$-approximation algorithm for graphs (Sections 2 and 3).

[1] Throughout this paper, for a collection of sets F we write $\cup F$ to refer to the set $\cup_{S \in F} S$.

The main idea here is to reduce this problem to the budgeted maximum coverage problem with an exponential number of hyperedges. However, we do not need to generate the exponentially large instance explicitly; but instead we make use of a concise representation of the instance and show that such instances can be approximated in polynomial time, given that we have access to an oracle that can select in polynomial time a hyperedge with approximately highest profit per unit of cost. As a last step in our reduction, we prove that such an oracle exists.

2. We derive in Section 5 a pseudo-polynomial time algorithm for the case when the incidence graph of the hypergraph is a forest (i.e., the hypergraph is *Berge-acyclic*). Further, we adapt this algorithm into a fully polynomial-time approximation scheme (FPTAS).
   At the core of this algorithm lies a bi-level dynamic program. The case of forests is important in its own right and, additionally, this algorithm constitutes an important building block of our $O(1/d^2)$-approximation algorithm (see Contribution 4).
3. In Section 6, we extend the above algorithm to a pseudo-polynomial time algorithm for incidence graphs with a bounded set of nodes that covers all cycles (i.e., the general case, but parametrized).
   More specifically, we show that for any incidence graph with a fixed-size *feedback hyperedge node set*, i.e., a hyperedge node set such that removing it from the incidence graph leaves no cycles, there exists a pseudo-polynomial time algorithm for the GBMC problem.
4. We give a $(1-\varepsilon)/(2d^2)$-approximation algorithm for every $\varepsilon > 0$ for the general case, where d is the maximum degree of a vertex in the hypergraph (Section 4).
   In this algorithm, we first decompose the incidence graph of the hypergraph into a collection of at most d trees for which we compute an approximate solution by using our FPTAS for forests above. From this we then extract a solution that is feasible for the original instance and guarantees an approximation ratio of at least $(1-\varepsilon)/(2d^2)$.

Related work

Much literature is available on the maximum coverage problem and its variants (see, e.g., [1, 3, 9] and the references therein). Most related to our problem is the budgeted maximum coverage problem [9]. As outlined above, the greedy approach of [9] cannot take into account that the costs are incurred per vertex instead of per set. Moreover, in [3], a generalized version of the budgeted maximum coverage problem is studied, but this generalization does not include GBMC as a special case.

Note that our GBMC problem on graphs reduces to the knapsack problem if the incidence graph is a matching. This problem is known to be weakly NP-hard and admits an FPTAS (see, e.g., [8]).

Our GBMC problem is related to the *budgeted bid optimization problem*. This problem was first proposed in the paper by Feldman et al. [5]. The authors derive a $(1-1/e)$-approximation algorithm if the budget constraint is *soft*, i.e., has to be met in expectation only. In contrast, in the budgeted bid optimization problem considered here, this budget constraint is hard.

The GBMC problem can be seen as a special case of a more general set of problems where we have to maximize a submodular profit function subject to the constraint that a submodular cost function does not exceed a given budget. This can be seen by considering the set of hyperedges to be the ground set of the submodular functions. However, when we have oracle access to both submodular functions, it has been shown that this more general problem is not approximable within a factor of $\log(m)/\sqrt{m}$, where m is the number of elements in the ground set. This holds even for the special case that the objective function is the modular function that returns the cardinality of the set. This follows from Theorem 4.2 in [10]; see also [7].

Preliminaries

For an integer $a \in \mathbb{N}$, we write $[a]$ and $[a]_0$ to denote the sets $\{1, \dots, a\}$ and $\{0, 1, \dots, a\}$ respectively. When F is a family of sets, we write $\bigcup F$ to denote the set $\bigcup_{S \in F} S$.

Let $G = (V, E)$ be a hypergraph. The *incidence graph* $I(G)$ of G is defined as the bipartite graph $I(G) = (E \cup V, H)$ with $H = \{\{e, v\} \mid v \in e\}$. We say that G is *acyclic* if its incidence graph $I(G)$ does not contain a cycle. Given a subset $E' \subseteq E$, we use $G[E']$ to refer to the *subgraph* of G induced by the hyperedges in E', i.e., $G[E'] = (V', E')$ with $V' = \cup E'$. A hypergraph T is called a *subtree* of G if T is a subgraph of G that is acyclic.

Throughout this paper we will use the convention that when discussing a hypergraph, n denotes the number of vertices of the hypergraph and m denotes the number of hyperedges of the hypergraph. Moreover, in the remainder of this paper, we assume without loss of generality that all costs (on the nodes or edges) are strictly positive.

It is not hard to prove that GBMC cannot be approximated to within a factor of $(1-1/e)$ in polynomial time, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\log\log n)})$ (details will be provided in the full version of the paper).

Due to space limitations, some figures and technical content is omitted from this paper and will be provided in the full version.

2 Budgeted Maximum Coverage with Oracles

In this section we first consider the classical budgeted maximum coverage problem. The result presented in this section will serve as a building block in the approximation algorithm presented in the next section, for solving GBMC on graphs.

A polynomial-time $(1-1/e)$-approximation algorithm for the budgeted maximum coverage problem was previously given in [9]. In the same paper, various simpler algorithms with worse approximation factors are presented. In this section, we present a variation of one of these algorithms that achieves a $(1 - 1/e)/2$-approximation guarantee, which can run even if the algorithm is not granted direct access to the input instance. We make this precise in the following definition.

▶ **Definition 1** (cost-efficiency oracle)**.** Let $I = (G = (V, E), c, p, B)$ be an instance of the budgeted maximum coverage problem, i.e., $G = (V, E)$ is a hypergraph, $c : E \to \mathbb{Q}_{\geq 0}$ is a function that specifies a cost $c(e)$ for each hyperedge $e \in E$, $p : V \to \mathbb{Q}$ is a function that specifies a profit $p(i)$ for each vertex $i \in V$, with $B \in \mathbb{Q}$ the budget. For $\alpha \in [0, 1]$, an *α-approximate cost-efficiency oracle* for I is a function $f_I : 2^V \to E$ that maps a set of vertices $S \subseteq V$ to a hyperedge $e \in E$ such that $c(e) \leq B$ and

$$\sum_{i \in e \setminus S} \frac{p(i)}{c(e)} \geq \alpha \cdot \sum_{i \in e' \setminus S} \frac{p(i)}{c(e')}.$$

for all $e' \in E$ with $c(e') \leq B$. Thus, a cost-efficiency oracle takes as input vertex set S and selects the hyperedge with the approximately highest cost-efficiency (up to a factor α), excluding the profit that would be contributed by vertices in S. Only hyperedges of which the cost does not exceed the budget are considered.

Let $I = (G = (V, E), c, p, B)$ be an instance of the budgeted maximum coverage problem, and let f_I be an α-approximate cost-efficiency oracle for this instance for some $\alpha \in (0, 1]$.

Consider now the following greedy algorithm $\mathcal{A}$ that takes as input only the cost-efficiency oracle f_I.

1. Set $S := \varnothing$ and $X := \varnothing$. Throughout the execution of the algorithm, X represents a feasible solution and S represents the set of vertices covered by X.
2. Let $e := f_I(S)$. If $S = V$ (i.e., there is no profitable hyperedge left) or if $c(e) + \sum_{e' \in X} c(e') > B$ (i.e., adding the hyperedge to X would exceed the budget), go to Step 3. Otherwise, set $X := X \cup \{e\}$, set $S = \bigcup X$, and repeat this step.
3. Output the solution with the highest total profit among the two solutions X and $\{e\}$.

▶ **Theorem 2.** *Algorithm $\mathcal{A}$ outputs an $(1 - 1/e^{\alpha})/2$-approximate solution to I in time $O(n \cdot t)$, where t is the amount of time it takes to evaluate f_I.*

The approximation factor is obtained by following rather closely the analysis given in [9] for a similar algorithm (that works without oracle access).

3 GBMC on Graphs

In this section, we present a $(1 - 1/\sqrt{e})/2$-approximation algorithm for the GBMC problem when the hypergraph is a graph. We do this by reducing the problem to the budgeted maximum coverage problem. An instance I of GBMC is reduced to an instance $r(I)$ of budgeted maximum coverage on the same set of vertices, such that the optimal solution of $r(I)$ has the same profit as the optimal solution of I. The instance $r(I)$ may have a superpolynomial number of hyperedges. However, instead of generating the budgeted maximum coverage instance explicitly, we construct only a 1/2-approximate cost-efficiency oracle $f_{r(I)}$ for $r(I)$. We then use Algorithm $\mathcal{A}$ on $f_{r(I)}$ in order to obtain a $(1 - 1/\sqrt{e})/2$-approximately optimal solution to $r(I)$ in polynomial time. Last, we show how to transform in polynomial time a feasible solution for $r(I)$ into a feasible solution for I with equal profit.

We begin by defining our reduction r.

▶ **Definition 3.** Let $I = (G = (V, E), c, p, B)$ be an instance of GBMC where G is a graph. Define the budgeted maximum coverage instance $r(I)$ as $r(I) = (G' = (V, E'), c', p, B)$, where

$$E' = \bigcup_{i \in V} E'_i \qquad \text{and} \qquad E'_i = \{S \cup \{i\} \mid \forall i' \in S : \{i', i\} \in E\},$$

that is, E'_i consists of the hyperedges X such that i is in X and all other vertices in X are connected to i by an edge. In other words, E' are all hyperedges corresponding to the stars of G. The cost function c' assigns a cost to each *hyperedge*: for a hyperedge $e \in E'$ we set $c'(e) = \sum_{i \in e} c(i)$. Note that c is a function that assigns a cost to each *vertex*, while c' is a function that assigns a cost to each *hyperedge in E'*. Note that the vertex sets, profit functions, and budgets of I and $r(I)$ are equal.

We first show that every feasible solution X' for $r(I)$ can be transformed into a feasible solution X for I in polynomial time such that the profit is preserved. Consider the following function g_I that maps solutions of $r(I)$ to I:

▶ **Definition 4.** Let $I = (G = (V, E), c, p, B)$ be an instance of GBMC and let X' be a feasible solution for $r(I) = (G' = (V, E'), c', p, B)$. The function g_I maps X' to the following solution for I.

$$g_I(X') = \left\{ \{i', i\} \in E \middle| \{i', i\} \in \bigcup X \right\}.$$

In words, $g_I(X')$ is the set of edges of G that are contained in a hyperedge of X'.

▶ **Lemma 5.** *Let X' be a feasible solution for $r(I)$. The edge set $g_I(X')$ is computable in time $O(mn|X'|)$. Moreover, the solution $g_I(X')$ is feasible (i.e., the total cost of all vertices covered by $g_I(X')$ does not exceed B). Also, $p(X') = p(g_I(X'))$.*

Proof. For the first claim, observe that for each hyperedge in X' and edge in E we need to check if that edge is contained in the hyperedge. This can be done in $O(n)$ time.

The second claim follows from the fact that the edge set $g_I(X')$ covers the same vertex set as X', and by definition

$$B \geq \sum_{e \in X'} c'(e) = \sum_{i \in \bigcup X'} c(i) \cdot |\{e \in X' : i \in e\}| \geq \sum_{i \in \bigcup X'} c(i) = \sum_{i \in \bigcup X'} c(i).$$

The third claim follows from the fact that the edge set $g_I(X')$ covers the same vertex set as X'. ◀

Next we show that the optimal solution for I is at most the profit of the optimal solution for $r(I)$. (Combined with the previous lemma, this entails that the optimal profits of I and $r(I)$ are equal.)

▶ **Lemma 6.** *Let p_{opt} be the maximum profit achievable in instance I. There exists a solution for $r(I)$ with profit p_{opt}.*

Proof. Let X be a profit-maximizing feasible solution for I. Assume without loss of generality that all paths in X are of size at most 2. In other words: no edge in X covers two vertices that are both covered by another edge (such an edge can be removed from X without decreasing the profit). Under this assumption, X is a set of stars. We construct from X a feasible solution X' for $r(I)$ that has the same profit, as follows. We define X' to be the collection of hyperedges that correspond to the maximal stars of X, i.e., for each maximal star of X, we add to X' the hyperedge consisting of the vertices covered by the star.

Since no pair of hyperedges in X' intersects, by definition of c' the total cost $\sum_{e \in X'} c'(e)$ equals $\sum_{i \in \bigcup X} c(i) < B$, and therefore X' is a feasible solution for $r(I)$. Moreover, X' and X cover the same set of vertices, and therefore profits of X in I equals the profit of X' in $r(I)$. ◀

A final ingredient that we need is a 1/2-approximate cost-efficiency oracle f for $r(I)$.

▶ **Definition 7.** We define the function f algorithmically as follows. Let S be the input argument to f. (As a reminder, S represents the set of vertices already covered during the execution of algorithm $\mathcal{A}$.) The high level idea is that we compute for each vertex i a set of vertices e_i in the star centered at i. Our goal for each of these stars is to select for each such i the substar with the (approximately) highest possible cost-efficiency, such that the cost of the vertices in the substar does not exceed the budget. We output the set in $\{e_i : i \in V\}$ that has the highest cost-efficiency.

1. Let V' be subset of vertices of V that have at least one neighbor not in S. For each $i \in V'$ (note that i itself may be in S):
   - **a.** Initialize $e_i := \{i\}$, and $d_i = c(i)$. If $i \in S$, set $n_i := 0$, and otherwise set $n_i := p(i)$.
   - **b.** Order non-increasingly the vertices i' that are not in S and are attached to i in graph G, according to ratio $p(i')/c(i')$. Denote this ordering by σ_i.
   - **c.** Let i' be the next vertex of σ_i (starting with the first vertex). If $(n_i+p(i'))/(d_i+c(i')) \geq n_i/d_i$, then add i' to e_i, set $n_i := n_i+p(i')$, and set $d_i := d_i+c(i')$, and repeat this step in case the total cost of e_i does not exceed B. Otherwise, if $(n_i+p(i'))/(d_i+c(i')) < n_i/d_i$ or if e_i exceeds the budget, stop iterating this step.

d. If the total cost of e_i lies within the budget, skip this step. Otherwise, let i' be the vertex last added in the previous step (i.e., the vertex in e_i with the least $p(i')/c(i')$). We consider two substars of e_i that are within the budget: The one consisting only of vertices i and i', and the one consisting of vertices $e_i \setminus \{i'\}$. We set e_i to be the substar with the highest cost-efficiency. Formally:

i. If $i \notin S$: if $(p(i)+p(i'))/(c(i)+c(i')) \geq (n_i - p(i'))/(d_i - p(i'))$ then set $e_i = \{i, i'\}$, $n_i := p(i)+p(i')$, and $d_i := c(i)+c(i')$. Otherwise set $e_i := e_i \setminus \{i'\}$, $n_i := n_i - p(i')$, and $d_i := d_i - c(i')$.

ii. If $i \in S$: if $p(i')/(c(i)+c(i')) \geq (n_i - p(i'))/(d_i - p(i'))$ then set $e_i := \{i, i'\}$, $n_i := p(i')$, and $d_i := c(i)+c(i')$ otherwise set $e_i := e_i \setminus \{i'\}$, $n_i := n_i - p(i')$, and $d_i := d_i - c(i')$.

2. Output the set in $\{e_i : |e_i| \geq 2 \wedge i \in V'\}$ with the highest cost-efficiency (i.e., the ratio n_i/d_i).

▶ **Lemma 8.** *The function f is a $1/2$-approximate cost-efficiency oracle for $r(I)$ and can be computed in time $O(n^2)$.*

Proof. It is easy to see that the set output by f is always a hyperedge in E', as it only outputs sets of hyperedges that correspond to stars of G. Moreover, in the last step, it is easy to verify that the set $\{e_i : |e_i| \geq 2 \wedge i \in V'\}$ is never empty when $S \neq V$. This implies that f is a valid cost-efficiency oracle. From the description of the algorithm above, it is also straightforward to see that f runs in time n^2: For each vertex, all neighbors are considered, where processing each neighbor takes a constant amount of time. (Not taking into account the bit-complexity of the arithmetic operations in this analysis, although the runtime would remain polynomial if we would take this aspect into account.)

What still needs to be proved is the approximation factor. Let $e_1, e_2, \ldots$ be the sets used in Step 2 of the algorithm. It suffices to show that for each $i \in V'$ for which it holds that $|e_i| \geq 2$, the ratio n_i/d_i is at least $(1/2) \cdot \sum_{i \in e' \setminus S} p(i)/c(e')$ for all $e' \in E'_i$. In words, the cost-efficiency n_i/d_i of the set e_i is at least half the maximum cost-efficiency among all hyperedges in E'_i (with respect to the input set S). (Note that we need not consider those $i \in V'$ for which $|e_i| = 1$: It can be easily verified that in this case, the optimal star centered at i is a single edge $\{i, i'\}$. This edge is also in $E'_{i'}$, and it is necessarily true that $|e'_i| \geq 2$.)

Let $i \in V'$ such that $|e_i| \geq 2$. Denote by $\Gamma(i)$ the vertices attached to i that are not in S. We will compare $|e_i|$ to an optimal *fractional* solution x: In this fractional solution each of the vertices i' attached to i (and not in S) is picked with a certain fraction $x_{i'} \in [0, 1]$, and vertex i is selected with fraction $x_i = 1$. The cost-efficiency is defined as $\frac{p(i)+\sum_{i' \in \Gamma(S)} x_{i'} p(i')}{\sum_{i' \in \Gamma(S)} x'_i c(i')}$ if $i \notin S$, and otherwise as $\frac{\sum_{i' \in \Gamma(S)} x_{i'} p(i')}{\sum_{i' \in \Gamma(S)} x'_i c(i')}$. Then it holds that the cost-efficiency of e_i^{frac} exceeds the cost-efficiency of the hyperedge $e_i^* \in E'_i$ that maximizes $\sum_{i \in e^* \setminus S} p(i)/c(e^*)$, which would be the optimal integral solution.

We claim that x is obtained by greedily selecting vertices in $\Gamma(i)$ according to non-increasing cost-efficiency (i.e., according to the order σ_i as given in Definition 7). A considered vertex is selected with the highest possible fraction as long as the budget is not exceeded, and as long as adding the vertex increases the cost-efficiency of the solution. Hence, in x all vertices of $\Gamma(i)$ are selected with either fraction 0 or 1, except at most one vertex, which is selected with a fraction in $(0, 1)$.

To see why this is true, suppose for contradiction that x has a different structure. In that case, if there is a vertex $i' \in \Gamma(i)$ with $x_{i'} > 0$ such that the cost-efficiency of i' is less than the cost-efficiency of x, then setting $x_{i'}$ to 0 will increase the cost-efficiency of the solution. Therefore, we may assume that the only vertices that are selected with a positive fraction, are vertices that have a cost-efficiency of at least the cost-efficiency of x. We can then consider the following operation: There must be two vertices $i', i'' \in \Gamma(i)$ for which it holds that $x_i' < 1$, $x_i'' > 0$, and the cost-efficiency of i' exceeds that of i''. In that case, decreasing $x_{i''}$ by an amount ϵ and increasing $x_{i'}$ by a maximal amount would increase the cost-efficiency (for a suitably small choice of ϵ), which is a contradiction to x being optimal. This shows that x is obtained by the aforementioned greedy procedure.

Next, we observe that if x happens to be integral, then the set of integrally selected vertices is precisely e_i, which means that e_i is the vertex set that maximizes the cost-efficiency. In this case the claim is proved. We now consider the case that x is not integral. From now on, let i' be the vertex that is fractionally selected in x and let S' be the integral vertices of x excluding i, i.e., $S' = \{i'' : i'' \neq i \wedge x_{i''} = 1\}$. It follows from Definition 7 that e_i is either the set $\{i\} \cup S$ or the set $\{i, i'\}$.

We distinguish four (very similar) subcases.

- We first consider the case that $i \notin S$ and $p(i') \geq \sum_{i'' \in S'} p(i'')$. Because x is the optimal fractional solution, the cost-efficiency of $S' \cup i, i'$ exceeds the optimal fractional solution and thus also the cost-efficiency of the optimal hyperedge e_i^*. Therefore, we conclude that the cost-efficiency of e_i is at least

$$\frac{p(i) + p(i')}{c(i) + c(i')} \geq \frac{p(i) + p(i')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')} \geq \frac{1}{2} \cdot \frac{p(i) + p(i') + \sum_{i'' \in S'} c(i'')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')}$$
$$\geq \frac{1}{2} \cdot \frac{\sum_{i'' \in e_i^*} p(i)}{\sum_{i'' \in e_i^*} c(i)},$$

as needed.

- In case $i \notin S$ and $p(i') < \sum_{i'' \in S'} p(i'')$ we similarly obtain that the cost-efficiency of e_i is at least

$$\frac{p(i) + \sum_{i' \in S'} p(i'')}{c(i) + \sum_{i'' \in S'} c(i'')} \geq \frac{p(i) + \sum_{i'' \in S'} p(i'')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')} \geq \frac{1}{2} \cdot \frac{p(i) + p(i') + \sum_{i'' \in S'} c(i'')}{c(i) + c(i') + \sum_{i'' \in S'} c(i'')}$$
$$\geq \frac{1}{2} \cdot \frac{\sum_{i'' \in e_i^*} p(i)}{\sum_{i'' \in e_i^*} c(i)}.$$

- The remaining two cases are analogous to the above two, where we replace $p(i)$ with 0.

◀

We are now ready to present the algorithm for GBMC on graphs, which we refer to as Algorithm $\mathcal{B}$. The algorithm is defined as follows. Let $I = (G = (V, E), c, p, B)$ be an input instance of GBMC where G is a graph.

- Run algorithm $\mathcal{A}$ on the 1/2-approximate cost-efficiency oracle f of Definition 7. This results in a solution X' for instance $r(I)$ (where $r(I)$ is given in Definition 3).
- Compute and output $g_I(X')$ (see Definition 4).

The correctness, polynomial runtime, and approximation factor of $(1 - 1/\sqrt{e})/2$ of algorithm $\mathcal{B}$ follow directly from the lemmas and definitions above. Note that the bound on the runtime can most likely be improved by a more careful analysis, but that is beyond the scope and goal of this work.

4 GBMC with bounded degree vertices

In this section we derive an approximation algorithm for GBMC for arbitrary hypergraphs $G = (V, E)$ with approximation ratio of $\mathcal{O}(1/d^2)$, where d refers to the maximum *frequency* of a node in a decomposition of G into trees. We first define formally the notions of *trees of hypergraphs*, and *frequency*.

A *tree of a hypergraph* G is defined as a partial hypergraph (i.e., a hypergraph that can be obtained by removing from each hyperedge a set of vertices) of which the incidence graph is a tree (i.e., a partial hypergraph that is *Berge-acyclic*). A *decomposition of a hypergraph* G *into trees* is a collection of trees of G such that (i) the incidence graphs of any two trees in the collection have disjoint edge sets, and (ii) the union of the incidence graphs of these trees equals the incidence graph of G. For a node $i \in V$, define the *frequency* d_i of i with respect to the tree collection $\mathcal{T}$ as the number of times i occurs in a tree of the collection, i.e., $d_i = |\{T_t \in \mathcal{T} \mid i \in V_t\}|$. Let the *maximum frequency* of a tree collection $\mathcal{T}$ be defined as $d = \max_{i \in V} d_i$. Therefore, in the worst case, d is the maximum degree of a node, as we can always decompose a hypergraph into subtrees that each consist of a single hyperedge of G. Our algorithm, which we name Algorithm $\mathcal{C}$, proceeds in three steps:

Step 1: Decomposition into trees. Let $I = (G = (V, E), p, c, B)$ be an instance of GBMC. We first decompose G into a collection of subtrees of G as follows: Initialize $t = 0$ and let $G_0 = G$ be the initial hypergraph. For $t \geq 0$ extract a subtree $T_{t+1} = (V_{t+1}, E_{t+1})$ from G_t and let $G_{t+1} = G_t \setminus T_{t+1}$ be the hypergraph that remains if we remove all hyperedges in E_{t+1} (but not the nodes) from G_t. Repeat the above procedure until eventually we obtain a graph G_z whose set of hyperedges is empty. Let $\mathcal{T} = \{T_1, \dots, T_z\}$ be the collection of subtrees extracted throughout this procedure. Note that by construction, for every two distinct trees $T_t, T_{t'} \in \mathcal{T}$ the set of hyperedges E_t and $E_{t'}$ are disjoint.

Using the above decomposition, we now define a new instance $I' = (G', p', c', B')$ of GBMC. The hypergraph G' consists of all trees $T_1, \dots, T_z$, where each tree $T_t = (V_t, E_t)$, $t \in [z]$, has its own "representative" for each node in V_t (with costs and profits being identical to the original ones). Thus, each node $i \in V$ has at most d_i representatives in G' and all trees in G' are node-disjoint. Finally, the budget B' of I' is set to $B' = dB$.

Step 2: Bin-packing the optimal solution of the decomposed instance. We compute a $(1 - \varepsilon)$-approximate solution X' for I' which respects the overall budget $B' = dB$ and also ensures that the total cost of every $T_t \in \mathcal{T}$ is at most B. The latter condition can easily be incorporated in our dynamic program for forests (thus also in our FPTAS) in Section 5.

The next step is to partition the trees in $\mathcal{T} = \{T_1, \dots, T_z\}$ into at most $2d$ sets $\mathcal{T}_1, \dots, \mathcal{T}_{2d}$ such that the total cost (according to X') in each set does not exceed B. This is in essence a bin-packing problem (i.e., packing a set of items of varying weights in a set of bins of limited capacity). Because every tree induces cost at most B and the total cost is at most dB, standard bin-packing arguments show that such a partition exists and can be computed in polynomial time [11].[2]

Step 3: Obtaining a solution for the original instance. Let $\mathcal{T}$ be a set of maximum profit (according to X') among the sets $\mathcal{T}_1, \dots, \mathcal{T}_{2d}$. We obtain the solution X from X' by picking

[2] To clarify: We can view each tree as an item of weight equal to the cost induced by X'. The goal then is to pack these items into bins of capacity B.

all hyperedges chosen in $\mathcal{T}$. Note that X is a feasible solution for I' but also for I as argued in Step 2. The algorithm outputs X.

▶ **Theorem 9.** *Algorithm $\mathcal{C}$ is a $(1-\varepsilon)/(2d^2)$-approximation algorithm for GBMC that runs in polynomial time, where d is the maximum frequency of a node.*

Proof. It is clear that the algorithm runs in polynomial time. Moreover, the algorithm outputs a feasible solution because the total cost induced by the nodes covered by X' in $\mathcal{T}$ is at most B (by construction). Therefore, the total cost of X in I is at most B.

It remains to analyze the approximation ratio. Let OPT_I be the optimal profit of the original instance I and let $OPT_{I'}$ be the optimal profit of the decomposed instance. Note that any feasible solution for I is also feasible for I'. This follows because the total cost of a solution for I is at most d times larger in I' and $B' = dB$. Therefore, the total profit $p_{I'}(X')$ of X' in I' is at least $(1-\varepsilon)OPT_{I'} \geq (1-\varepsilon)OPT_I$.

Because we choose the maximum set $\mathcal{T}$ among the $2d$ many sets, the total profit of X in I' is at least $(1-\varepsilon)OPT_I/(2d)$. Also, the total profit of X in I is at most a factor d less than the total profit it induces in I'. Therefore, the total profit $p_I(X)$ of X in I satisfies $p_I(X) \geq (1-\varepsilon)OPT_I/(2d^2)$, which completes the proof. ◀

5 GBMC when the incidence graph is a forest

In this section, we derive a bi-level dynamic program for the case when the incidence graph is a forest. We refer to this special case as GBMC-Forest. We also show that our dynamic program can be turned into an FPTAS, for which we introduce P as the maximum profit of a vertex, i.e., $P = \max_{v \in V} p(v)$.

▶ **Theorem 10.** GBMC-Forest *can be solved optimally in time $\mathcal{O}(mn^3P^2)$.*

Suppose the given incidence graph is a forest and consists of z trees $T_1, \dots, T_z$. In order to facilitate the exposition of our dynamic program we combine these trees simply into a single tree T as follows. Introduce an artificial hyperedge e_0, representing the root of the tree. Furthermore, we introduce for each tree T_t with $t \in [z]$ a dummy vertex node d_t with zero profit and cost, i.e., $p(d_t) = c(d_t) = 0$, and connect it to e_0. Finally, we connect d_t to its respective tree T_t by adding an edge in the incidence graph from d_t to an arbitrary hyperedge from T_t.

This yields a bipartite graph that is a single tree. Note that the nodes along a path from the root to any other node are alternately hyperedge/vertex nodes. Assume we “unfold” this tree in order to draw the bipartite graph in a layered manner, as illustrated in Figure 1.

Our dynamic program processes the unfolded tree T in a bottom-up manner. It consists of two separate dynamic programs, one for the hyperedges, and one for the vertices. We describe these programs informally in this section (technical details of the dynamic program will be given in the full version).

Consider an arbitrary subtree in T rooted at either a node represented by a hyperedge or vertex. Both dynamic programs rely on the fact that a subset of this subtree can be solved to optimality, which immediately can be used to solve a greater subset to optimality. In case the subtree is rooted at a node represented by a hyperedge, we consider the subtree up until the first s children in the subtree of the hyperedge. Once the optimal solutions (minimum required cost to obtain a specific profit, if possible) are known for every possible profit (upper bounded by nP), it is possible to find optimal solutions for the subtree until the first $s+1$ children by linear enumeration. A similar, but slightly adapted method works in case the subtree is rooted by a vertex rather than a hyperedge.

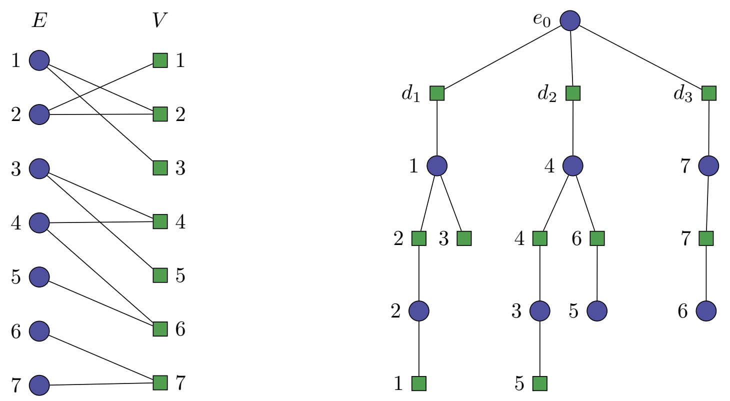


Figure 1 Example of a incidence graph (left) and its "unfolded' tree (right).

Furthermore, we can employ standard techniques (profit truncation) to turn the above pseudo-polynomial time algorithm into an FPTAS, i.e., an algorithm that takes an error parameter $\varepsilon > 0$ and computes in time polynomial in the input size and $1/\varepsilon$ a $(1 - \varepsilon)$-approximation to the optimal solution.

▶ **Theorem 11.** *There exists an FPTAS for* GBMC-FOREST *that runs in time* $\mathcal{O}(mn^5/\epsilon^2)$.

6 GBMC with a bounded size feedback vertex set

We have shown in the previous section that GBMC-FOREST can be solved in pseudo-polynomial time and that there is an FPTAS. This implies that the inapproximability of the general problem is caused by the cycles in the incidence graph. In this section, we provide a fixed parameter tractability result that allows us to handle incidence graphs with cycles. The fixed parameter is the minimum number α of hyperedge nodes that we need to remove in order to make the incidence graph acyclic.

To provide an initial intuition, construct a graph $G' = (E, E')$ where every hyperedge $e \in E$ is represented by a node, and two nodes are connected if and only if the corresponding hyperedges share at least one element. This defines the edge set E' in the new graph. Consider the case in which G contains solely one cycle. Select a hyperedge node of the cycle and fix whether this hyperedge is chosen or not in a solution (i.e., set $x = 0$ or 1). We then consider the reduced problem in which hyperedge e is removed. The incidence graph of the reduced problem is a forest and can thus be solved optimally by using the pseudo-polynomial time algorithm (or approximately by using the FPTAS) of the previous section. Solving this GBMC-FOREST problem for every possible choice of hyperedge to remove, and taking the best solution, yields a solution to the general GBMC problem for the instance with one cycle. Thus, if the graph contains one cycle, the running time is multiplied by 2. We can extend this idea to more general graphs.

Define α as the minimum number of hyperedges whose removal turn the graph into a forest, i.e., α is the cardinality of the *minimum feedback vertex subset* of the hyperedge nodes of the incidence graph. We refer to the latter as the *minimum feedback hyperedge node set*. Then the running time of the algorithms mentioned in the previous sections is multiplied by a factor of 2^α, because it is necessary to solve the problem on a forest for every combination on those α hyperedges.

The problem of finding a minimum feedback vertex set is NP-hard in general, but it is fixed parameter tractable. Cao et al. [2] give an $\mathcal{O}(3.83^\alpha \alpha n^2)$ time algorithm to solve the

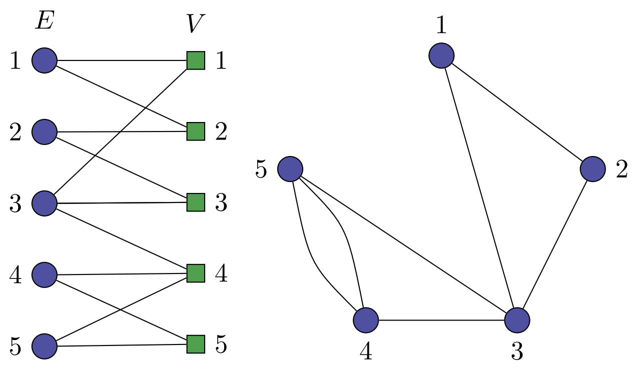


Figure 2 Example of a transformation to the feedback vertex set problem.

problem, where n here refers to the number of nodes in the graph. We use this to prove the following theorem.

▶ **Theorem 12.** *GBMC is solvable in $\mathcal{O}(mn^3P^22^\alpha + 3.83^\alpha \alpha m^2)$ time, where α is the size of the minimum feedback hyperedge node set.*

All that needs to be shown is how to use the $\mathcal{O}(3.83^\alpha \alpha n^2)$ algorithm of [2] in order to find a minimum feedback vertex set restricted to only the hyperedge nodes of the incidence graph. This is straightforward: We reduce the incidence graph of G to the aforementioned multigraph $G' = (E, E')$ with only E as its vertex set. The edge set E' is constructed as follows: there exists an edge between two hyperedges if they share at least one vertex in the original graph. It is now easy to see that there is a one-to-one correspondence between the cycles in the incidence graph of G and the cycles in G', and each cycle in the incidence graph of G corresponds to a cycle in G' on the same set of vertices. Therefore, a minimum feedback vertex set of G' corresponds to a minimum feedback hyperedge node set of G, and the algorithm of Cao et al. will find such a set in $\mathcal{O}(3.83^\alpha \alpha n^2)$ time.

The transformation is illustrated in Figure 2. There, hyperedge 3 and 5 are connected by vertex 4 in (the incidence graph representation of) G, thus an edge $\{3, 5\}$ is added in G'. The edge labels are omitted. Note that hyperedge 4 and 5 are connected by two edges, because they are both connected to vertex 4 and vertex 5.

7 Conclusions

In this paper we have presented various approximation algorithms for important special cases of the GBMC problem. Clearly, the most interesting open problem that remains to be solved is whether there exists a constant factor approximation algorithm for the general GBMC problem that runs in polynomial-time. Such a result would form a very interesting contrast with the inapproximability result for the problem of submodular function maximization with a submodular budget constraint under the oracle access model, which we mentioned in the introduction.

An interesting and challenging intermediate goal would be to find a constant factor approximation algorithm for the case that the hyperedges have a fixed size k. Algorithm $\mathcal{A}$ and Theorem 2 might serve as a useful tool for achieving this goal.

References

1 G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

2 Y. Cao, J. Chen, and Y. Liu. On feedback vertex set new measure and new structures. In *Proceedings of the 12th Scandinavian conference on Algorithm Theory*, pages 93–104. Springer-Verlag, 2010.

3 R. Cohen and L. Katzir. The generalized maximum coverage problem. *Information Processing Letters*, 108(1):15–22, 2008.

4 U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

5 J. Feldman, S. Muthukrishnan, M. Pál, and C. Stein. Budget optimization in search-based advertising auctions. In *Proceedings of the 8th ACM conference on electronic commerce*, pages 40–49, New York, NY, USA, 2007. ACM.

6 D.S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

7 R.K. Iyer and J.A. Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *Advances in Neural Information Processing Systems*, pages 2436–2444. MIT Press, 2013.

8 H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer-Verlag, 2004.

9 S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

10 Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM Journal of Computing*, 40(6):1715–1737, 2011.

11 V.V. Vazirani. *Approximation algorithms*. Springer-Verlag, 2003.

Computational and Proof Complexity of Partial String Avoidability

Dmitry Itsykson[1], Alexander Okhotin[2], and Vsevolod Oparin[3]

1 **St. Petersburg Department of V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia**
`dmitrits@pdmi.ras.ru`

2 **Department of Mathematics and Statistics, University of Turku, Finland**
`alexander.okhotin@utu.fi`

3 **St. Petersburg Academic University, St. Petersburg, Russia**
`oparin.vsevolod@gmail.com`

Abstract

The partial string avoidability problem, also known as partial word avoidability, is stated as follows: given a finite set of strings with possible "holes" (undefined symbols), determine whether there exists any two-sided infinite string containing no substrings from this set, assuming that a hole matches every symbol. The problem is known to be NP-hard and in PSPACE, and this paper establishes its PSPACE-completeness. Next, string avoidability over the binary alphabet is interpreted as a version of conjunctive normal form (CNF) satisfiability problem (SAT), with each clause having infinitely many shifted variants. Non-satisfiability of these formulas can be proved using variants of classical propositional proof systems, augmented with derivation rules for shifting constraints (such as clauses, inequalities, polynomials, etc). Two results on their proof complexity are established. First, there is a particular formula that has a short refutation in Resolution with shift, but requires classical proofs of exponential size (Resolution, Cutting Plane, Polynomial Calculus, etc.). At the same time, exponential lower bounds for shifted versions of classical proof systems are established.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases partial strings, partial words, avoidability, proof complexity, PSPACE-completeness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.51

1 Introduction

The field of proof complexity is concerned with the size of proofs for different kinds of logical formulas, under various measures of size. The most common subject, motivated by SAT-solvers, are Boolean formulas in conjunctive normal form (CNF), and there is a substantial body of literature on lower bounds on the size of a proof that a given CNF formula is unsatisfiable. For instance, there are exponential lower bounds on the size of Resolution [9, 16], Cutting Plane [14] and Polynomial Calculus proofs [15, 10], whereas for Frege and Lovász–Schrijver proof systems, no superlinear lower bounds are known [7]. This line of research is aimed, in particular, at separating the NP and co-NP complexity classes [8].

This paper investigates the complexity issues for a variant of CNF formulae, in which every clause exists in countably many variants, with variable numbers shifted by any constant.

For example, if a formula contains a clause $x_1 \vee \neg x_4$, then it contains all clauses of the form $x_{1+i} \vee \neg x_{4+i}$, where i is any integer. The resulting *Shift-CNF* depends on countably many variables, and represents uniformly defined constraints applied to all blocks of variables. It can be alternatively written as a finite formula, with each clause using a universal quantifier on position numbers, such as in $(\forall i \in \mathbb{Z})(x_{i+1} \vee \neg x_{i+4})$. These formulas have a compactness property (that is, their satisfiability can be tested on a sufficiently large finite block of variables), and several proof systems, such as Resolution, can be applied to clauses of this form. Those systems are a natural subject for proof complexity studies.

Another motivation to study the satisfiability problem for Shift-CNF is that it is equivalent to the *partial substring avoidability problem*, which received some attention in formal language theory [6, 5]. To see this relationship, first consider the following (fairly obvious) representation of the satisfiability problem for standard CNF formulae (SAT) in terms of strings. Let $x_1, \ldots, x_n$ be the set of variables of a CNF formula. Then, each clause in the formula may be written down as a string of length n over the 3-symbol alphabet $\Sigma = \{0, 1, \Box\}$, which lists the values of variables that make this clause false: to be precise, each i-th position in the string contains 0 if the clause contains a literal x_i, or 1 if there is a literal $\neg x_i$, or a "hole" ($\Box$) if this variable does not occur in the clause. Thus, a CNF formula is presented as a set of *forbidden strings*, and its satisfying assignments are exactly all binary strings of length n that *do not match* the string representation of each clause.

In this setting, all strings are of the same length n, and cannot be moved in relation to each other, because that would mean shifting variable numbers. If this restriction is lifted, then each forbidden partial string represents a pattern that may not occur in a desired binary string *beginning from any position.* This is the partial substring avoidability problem, which precisely corresponds to the Shift-CNF satisfiability problem (Shift-SAT).

In the special case when forbidden strings are complete strings (without holes), their avoidability can be decided in linear time using the algorithm by Aho and Corasick [1]. Another solution to this problem, given in Lothaire [13], uses a special case of Resolution proofs, applied to strings instead of clauses: two strings $xy0$ and $y1$ can be resolved to xy. Lothaire [13] proves that a set of (complete) strings is unavoidable if and only if the empty string can be derived; furthermore, the length of this derivation is linear.

The computational complexity of the full case of the partial string avoidability problem was studied by Blanchet-Sadri et al. [6], who proved that it is NP-hard. Soon thereafter, Blakeley et al. [5] showed that the problem is in PSPACE. Its exact complexity remained open. The first result of this paper is that partial string avoidability is actually PSPACE-complete: this is established in Section 3 by a direct reduction from the polynomial-space Turing machine membership problem.

This result puts the Shift-SAT problem in the context of proof systems for PSPACE-complete languages. The proof complexity of such systems is important, in particular, as an approach to separating NP from PSPACE. A generalized Resolution proof system for the Quantified Boolean Formula (QBF) problems—the *Q-Resolution*—was introduced by Kleine Büning et al. [12], and other Resolution-based proof systems for QBF and their proof complexity have recently been studied by Beyersdorff et al. [3, 4], by Balabanov et al. [2] and by Janota and Marques-Silva [11].

The Shift-SAT problem is attractive for being similar to the classical SAT problem, to the point that all proof systems for UNSAT, such as Resolution, Cutting Plane, Polynomial Calculus, etc., can be directly applied to Shift-SAT formulas. For every such proof system Π, there is its shifted version, Shift-Π, with an additional derivation rule for adding an arbitrary integer to the numbers of all variables in a constraint.

Two results on the proof complexity of shifted systems are presented in this paper. Lower bounds on the size of shifted proofs, given in Section 5, are obtained by encoding any of the known superpolynomial lower bounds on classical proofs. Efficient proofs using shifts are presented in Section 6, as an example of an unsatisfiable shifted CNF formula which has a polynomial-sized Shift-Resolution proof, whereas its proofs in every classical proof system are of at least exponential size.

The proof system for complete strings defined by Lothaire [13], is a special case of Shift-Resolution. Lothaire's [13] clauses contain *contiguous blocks* of variables $x_i, x_{i+1}, \dots, x_j$, whereas general CNF and Shift-CNF clauses may have gaps between variable numbers. The results on the proof complexity of Shift-Resolution obtained in this paper, in particular, imply an *unconditional separation* between these two problems, in terms of the length of resolution derivations—as compared to the separation in terms of computational complexity, which is conditional to P $\neq$ PSPACE.

2 The partial string avoidability problem

Any finite set of symbols Σ is called an *alphabet.* A *(two-sided) infinite string* over Σ is a mapping $\alpha\colon \mathbb{Z} \to \Sigma$. If two infinite strings, α and β, are the same up to shifting them by a constant number of positions, that is, if for some offset $d \in \mathbb{Z}$, $\alpha(n) = \beta(n+d)$ for all $n \in \mathbb{Z}$, then α and β are said to be *equal*, and are considered to be the same infinite string.

The set of all infinite strings over an alphabet Σ is denoted by Σ^∞, whereas Σ^* is the set of all finite strings $a_1 \dots a_n$, with $n \geqslant 0$ and $a_1, \dots, a_n \in \Sigma$. Each i-th symbol of a finite string $w = a_1 \dots a_n$ shall be denoted by $w[i] = a_i$, and a *substring* $a_i \dots a_j$ is denoted by $w[i..j]$. The same notation is used to extract a finite substring $\alpha[i..j]$ from an infinite string α.

For a set of finite strings $L \subseteq \Sigma^*$, the set of infinite strings formed by concatenating any elements of L is denoted by $L^\infty = \{\, \dots w_{-1} w_0 w_1 w_2 \dots \mid w_i \in L \text{ for all } i \in \mathbb{Z} \,\}$. In particular, the infinite string formed by repeating a finite string $w \in \Sigma^*$ is $w^\infty = \dots www \dots$

A *partial string* over an alphabet Σ is a finite string over the alphabet $\Sigma \cup \{\square\}$, where a square ($\square$) denotes an unknown symbol (a hole). For the purposes of string matching, a hole may stand for any symbol from Σ: to be precise, two partial strings of the same length, u and v, are said to be *compatible*, if, whenever they differ in some j-th position ($u[j] \neq v[j]$), either $u[j] = \square$ or $v[j] = \square$.

An infinite string α over an alphabet Σ is said to *avoid a partial string* w, if every substring of α of the same length as w is incompatible with w: that is, $\alpha[i+1..i+|w|]$ is incompatible with w for every $i \in \mathbb{Z}$. A finite string avoiding w is defined similarly. A finite or infinite string is said to *avoid a set of partial strings* L, if it avoids every element of L.

The *partial string avoidability problem* is then stated as follows: given an alphabet Σ and a finite set of partial strings S over Σ, determine whether there exists an infinite string that avoids this set.

The first thing to observe is that if a finite set of partial strings is avoided by a sufficiently long finite string then it is avoided by an infinite string. Therefore, the avoidability problem may be regarded as a problem on finite strings, and is guaranteed to have an effective solution.

▶ **Lemma 1.** *Let Σ be an alphabet containing m symbols, let $L \subset (\Sigma \cup \{\square\})^*$ be a finite set of partial strings, and let ℓ be the length of the longest string in L. Then, L is avoided by an infinite string if and only if L is avoided by a finite string of length $m^\ell + \ell$.*

Lemma 1 provides an obvious NEXPTIME algorithm for testing partial string avoidability. However, the problem is known to be easier.

▶ **Lemma 2** (Blakeley et al. [5, Cor. 2]). *The partial string avoidability problem is in* PSPACE.

Sketch of a proof. Let $k = \max_{w \in S} |w|$, and consider the graph with the set of vertices $\{0,1\}^k$, which contains an arc from $u \in \{0,1\}^k$ to $v \in \{0,1\}^k$, if the string uv contains no forbidden substrings from S. An infinite string avoiding all substrings from S exists if and only if there is a cycle in the graph. A polynomial-space nondeterministic Turing machine can guess this cycle by walking over the graph. ◀

In addition, Blakeley et al. [6] proved that the partial string avoidability problem is NP-hard, but its exact complexity remained open. The first contribution of this paper, presented in Section 3, is that this problem is actually PSPACE-complete.

Another crucial property of the partial string avoidability problem is that it can be reduced to the same problem over the binary alphabet.

▶ **Lemma 3** (Blakeley et al. [5, Thm. 7]). *The partial string avoidability problem is reducible in polynomial time to the partial string avoidability problem over the alphabet* $\Sigma = \{0,1\}$.

This allows an interpretation of this problem as a logical formula. An investigation of the proof complexity aspects of testing avoidability is carried out later in Sections 4–6.

3 The PSPACE-hardness proof

The first contribution of this paper is the exact computational complexity of the partial substring avoidability problem (Avoidability).

▶ **Theorem 4.** Avoidability *is* PSPACE-*complete.*

The problem is in PSPACE by Lemma 2, and it remains to prove that it is PSPACE-hard. Let L be any language in PSPACE, that is, there exists a one-tape Turing machine M and a polynomial $s(\ell)$, such that on any input string of length ℓ, the machine M uses $s(\ell)$ space and eventually halts in an accepting state, if the input is in L, or in a rejecting state otherwise. Assume that M is modified, so that, instead of halting in a rejecting state, it loops without using any additional space.

Theorem 4 follows from Lemma 5 applied to M.

▶ **Lemma 5.** *For every Turing machine that uses at most $s(\ell)$ space on inputs of length ℓ and for every input string $w \in \Sigma^\ell$, there exists an alphabet Ω and a finite set of partial strings $P \subset (\Omega \cup \{\Box\})^*$, such that the Turing machine loops on w if and only if there exists a two-sided infinite string $\alpha \in \Omega^\infty$ that avoids all partial strings in P. Given the machine, P can be constructed in time polynomial in $s(\ell)$.*

Consider computation histories of a Turing machine, where its configurations are written one after another. The general plan is to use forbidden strings to ensure that each listed configuration is the successor of the previous one. If the Turing machine loops, then there is an infinite string containing its infinite computation. A final configuration has no successor, so if it is ever reached, then the list of configurations cannot be continued to an infinite string.

However, there is a problem with this idea. If a Turing machine loops on the input, this means that it loops *starting from its initial configuration.* But there could also exist some looping computations beginning from unreachable configurations. These computations give rise to undesired infinite strings.

This problem can be circumvented in the following way. Let the Turing machine be augmented with an *alarm clock* containing a counter that is incremented at every step. The time until the alarm is triggered must be long enough for any accepting computation to terminate. Then, once the counter overflows, this means that the machine has looped, and the alarm clock resets the machine to its initial configuration. This shall ensure that if the machine does not loop starting from the initial configuration, then there is no infinite string.

Denote the Turing machine by $T = (\Sigma, \Gamma, Q, q_0, \delta, q_{acc})$, where Σ is the input alphabet; Γ is the tape alphabet, with $\Sigma \subseteq \Gamma$; Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta \colon (\Gamma \cup \{\vdash, \dashv\}) \times Q \to (\Gamma \cup \{\vdash, \dashv\}) \times Q \times \{-1, +1\}$ is the transition function, and $q_{acc} \in Q$ is the accepting state.

The machine operates on a tape containing $n = s(\ell)$ symbols from Γ enclosed between left and right end-markers ($\vdash$, $\dashv$). It has $(n+2) \cdot |Q| \cdot |\Gamma|^n$ possible configurations. Attached to the tape, there is a separate k-bit counter, with $k = \lceil \log_2(n+2) + \log_2 |Q| + n \log_2 |\Gamma| + 1 \rceil$, that is, with 2^k greater than the number of possible configurations. This information is encoded in a $(n+k+2)$-symbol *block* that consists of a Turing machine configuration ($n+2$ symbols) and of the alarm clock's counter (k digits).

Each symbol in the block is of the form (X, i), where $i \in \{0, \ldots, n+k+1\}$ is a number of the position in the block, and X is a payload, to be defined later. The Turing machine tape is encoded in positions $0, \ldots, n+1$, and the counter is in positions $n+2, \ldots, n+k+1$.

For each symbol used for encoding the tape, the payload is a triple (a, q, f), where $a \in \Gamma \cup \{\vdash, \dashv\}$ is a tape symbol, $q \in Q \cup \{-\}$ is either a state of the Turing machine (if the head is in this square) or a minus sign (if the head is elsewhere), while the third component $f \in \{\text{N}, \text{R}\}$ is a flag used for restarting the machine.

In each cell of the counter, the payload is any of the two digits: *zero* (0) and *one* (1).

Altogether, the following alphabet is used.

$$\Omega = \big((\Gamma \cup \{\vdash, \dashv\}) \times (Q \cup \{-\}) \times \{\text{N}, \text{R}\} \times \{0, \ldots, n+1\}\big) \cup \\ \cup \big(\{0, 1\} \times \{n+2, \ldots, n+k+1\}\big)$$

Some symbols of this alphabet are actually unnecessary. These symbols shall be identified in the proof, and then they can either be removed from the alphabet, or, equivalently, they can be listed as 1-symbol forbidden strings.

In this proof, the set of forbidden substrings P is constructed gradually, with each instalment of substrings ensuring further properties of any infinite string avoiding those substrings.

The first step of the construction is to ensure that the infinite string consists of blocks of length $n+k+2$, each correctly split into tape symbols and counter digits, and with all positions in the block correctly numbered. The payload is not checked yet, with the exception for the correct position of end-markers, namely, that they occur in the beginning and in the end of the tape, and nowhere else.

▶ **Claim 6.** *If a two-sided infinite string contains no forbidden substrings from P, then it is of the form $\ldots \alpha_{-1} \alpha_0 \alpha_1 \alpha_2 \ldots \alpha_\ell \ldots$, where each α_i is a string of the following form, for some tape symbols $a_1, \ldots, a_n \in \Gamma$, states $p_0, \ldots, p_{n+1} \in Q \cup \{-\}$, flags $f_0, \ldots, f_{n+1} \in \{\text{N}, \text{R}\}$ and counter digits $d_0, \ldots, d_{k-1} \in \{0, 1\}$.*

$$\alpha_i = (\vdash, p_0, f_0, 0)(a_1, p_1, f_1, 1) \ldots (a_n, p_n, f_n, n)(\dashv, p_{n+1}, f_{n+1}, n+1) \\ (d_{k-1}, n+2) \ldots (d_1, n+k)(d_0, n+k+1)$$

This is achieved by using two-symbol forbidden strings of the form $(X, i)(Y, j)$, where $i + 1 \neq j \pmod{n+2+k}$, while X and Y represent any payload. The correct use of end-markers on the tape is enforced by removing a few invalid symbols, such as $(\dashv, 0)$, from the alphabet (alternatively, they can be listed as one-symbol forbidden strings).

With the enumeration of positions in place, consider the implementation of the alarm clock. The alarm clock uses a k-bit binary counter, with the least significant digit in position $n + k + 1$ and with the most significant digit in position $n + 2$. The counter is incremented at every step. Upon overflow, it is reset to zero, and at the same time a restart signal is sent to the left into the Turing machine tape.

▶ **Claim 7.** *In every block, the payload in the counter digits forms a binary string, $d_{k-1} \dots d_1 d_0$. The number represented by this string is greater by* 1 *(modulo 2^k) than the number represented in the previous block.*

Then, in particular, the enumeration of the blocks $(\dots, \alpha_{-1}, \alpha_0, \alpha_1, \alpha_2, \dots)$ assumed in the proof can be shifted so that the value of the counter in each α_i is exactly i modulo 2^k.

The forbidden strings implementing Claim 7 ensure that the corresponding digits of two subsequent counters, which are $n + 2 + k$ positions apart, correctly implement addition of 1. For example, two forbidden partial strings, $(0, n + k + 1)\square^{n+k+1}(0, n + k + 1)$ and $(1, n + k + 1)\square^{n+k+1}(1, n + k + 1)$, set the least significant digit to alternate between 0 and 1. Incrementation in further digits is controlled by similarly defined partial strings.

If the most significant digit of the counter changes from 1 to 0, this indicates counter overflow, and therefore the alarm clock sends the restart signal (R) to the left of the current symbol, from where it propagates to all tape squares in this block. The restart flag is handled in the next group of forbidden strings.

▶ **Claim 8.** *In every block α_i, if $i \not\equiv 0 \pmod{2^k}$, then each tape square is marked as normal* (N)*. If $i \equiv 0 \pmod{2^k}$, then each tape square has a restart flag* (R)*.*

The next group of restrictions ensures that if the restart signal sweeps over the tape in some block, then at the same time the Turing machine configuration to overwritten with its initial configuration.

▶ **Claim 9.** *If a restart occurs in a block, then this block contains the initial configuration of the Turing machine on the input string w.*

The last group of forbidden strings ensures the simulation of the Turing machine's transitions in normal situations, when no reset takes place. Every tape square in a configuration depends on three squares in the previous configuration: namely, on the same square, its left neighbour and its right neighbour. This dependence is checked by prohibiting all mismatches.

▶ **Claim 10.** *If a block contains a syntactically correct Turing machine configuration, and no reset occurs at the subsequent block, then the subsequent block contains a syntactically correct configuration at the next step.*

Once a set of forbidden partial strings satisfying Claims 6–10 is constructed, the proof of Lemma 5 follows from these Claims.

4 Proof systems

As outlined in the introduction, the avoidability problem over a binary alphabet $\Sigma = \{0, 1\}$ can be treated as a logical question. Let $\Gamma = \{x_i\}_{i \in \mathbb{Z}}$ be the set of numbered Boolean variables. Any variable x_i or its negation $\neg x_i$ is called a *literal*; a literal of an unknown sign can be denoted by x_i^σ, with $\sigma \in \{0, 1\}$, so that $x_i^0 = x_i$ and $x_i^1 = \neg x_i$. A *clause* is a disjunction of finitely many literals, such as $x_1 \vee \neg x_4$. A clause is *shifted* by adding the same integer to all variable numbers. The conjunction of all shifts of a clause C is denoted by $\mathrm{Shifts}(C)$ and called a *moveable clause.* For instance, in the above example, $\mathrm{Shifts}(x_1 \vee \neg x_4) = \bigwedge_{i \in \mathbb{Z}} (x_{i+1} \vee \neg x_{i+4})$.

A conjunctive normal form (CNF) formula φ is a conjunction of finitely many clauses, and it accordingly depends on finitely many variables. From the perspective of proof systems, it may be regarded as a finite *set of clauses.* If these clauses are replaced with the corresponding moveable clauses, the resulting formula, denoted by $\mathrm{Shifts}(\varphi)$, is called a *Shift-CNF.* A CNF, or a Shift-CNF, it is said to be *satisfiable*, if, for some assignment of Boolean values to variables, all its clauses hold true.

In terms of strings, a clause is a partial string that lists all values that make its literals false, with holes instead of the unused variables. A clause is matched at a specific position in the strings, whereas a moveable clause means that a string is matched at all positions. For example, the above moveable clause $\mathrm{Shifts}(x_1 \vee \neg x_4)$ may be regarded as a forbidden partial string $0\square\square1$. If all the listed values hold at once, then the clause is false. Accordingly, avoidance of all partial strings representing the moveable clauses in a Shift-CNF is equivalent to its satisfiability.

In view of this equivalence, Lemma 1 states that satisfiability of a Shift-CNF can be tested by considering finitely many shifts of each moveable clause—namely, those involving the variables $x_1, x_2, \ldots, x_{2^\ell+\ell}$.

Unsatisfiability of sets of clauses can be proved using *proof systems.* A refutation of a set of clauses in the Resolution proof system is a sequence of clauses $C_1, C_2, \ldots, C_s$, where C_s is the *empty clause* (false), and each clause C_i, with $i \in \{0, \ldots, s-1\}$, is either a clause from the given set, or is derived from some earlier clauses using the weakening rule or the resolution rule. By the *weakening rule*, a clause $C \vee D$ is derived from a clause C by adding any extra literals D. The *resolution rule* is applied to a pair of clauses $x \vee C$ and $\neg x \vee D$, where x is a variable, deriving the clause $C \vee D$. The *length* of a Resolution proof is the number of clauses therein. For an unsatisfiable formula φ, the length of its shortest Resolution refutation is denoted by $S_{Res}(\varphi)$.

The following estimation of the length of Resolution proofs is well-known.

▶ **Lemma 11.** *Let F be an unsatisfiable CNF formula, and let x be one of its variables. Then, $S_{Res}(F) \leqslant S_{Res}(F[x := 0]) + S_{Res}(F[x := 1]) + 1$.*

The definition of Resolution proofs equally applies to infinite sets of clauses. It is known that a set of clauses, finite or infinite, has a Resolution refutation if and only if that set is unsatisfiable. For infinite sets of clauses, this result generally holds by the compactness theorem, although it gives no estimations of the size of a refutation. For infinite formulas of the form $\mathrm{Shifts}(\varphi)$ studied in this paper, there is the following upper bound on the length of their Resolution refutations.

▶ **Lemma 12.** *Assume that an unsatisfiable CNF formula φ depends on variables $x_1, x_2, \ldots, x_n$, and assume that in each clause, the least and the greatest variable numbers differ by at most $k \geqslant 2$. Then φ has a Resolution refutation of size at most $2n^k$.*

Sketch of a proof. Using Lemma 11, a Resolution proof for φ can be constructed by selecting a few (t) variables and substituting all possible values into them. Each substituted formula has a derivation; let T be the length of the longest of them. Then, φ has a derivation of length $2^t T + 2^t - 1$.

Let the middle block of k variables be selected: that is, all variables with numbers between $\frac{n-k}{2}$ and $\frac{n+k}{2}$. Substituting all their values splits the formula into two independent subformulas, and it is sufficient to refute only one of them. Thus, the problem has been reduced to the same problem for $\frac{n-k}{2}$ variables, and the result follows by an inductive argument. ◀

Returning to the avoidability problem for partial strings $w_1, w_2, \ldots, w_m$, Lemma 1 implies that the avoidability test is given by a CNF with $2^k + k$ consecutive variables, where $k = \max_i |w_i|$. Then, by Lemma 12, this formula has a Resolution refutation of size $2^{O(k^2)}$.

For the Resolution method for Shift-CNF formulas, there is a natural derivation rule to be added: the *shifting rule*, by which, from any clause $x_{i_1}^{\sigma_1} \vee x_{i_2}^{\sigma_2} \vee \cdots \vee x_{i_k}^{\sigma_k}$, one can derive any clause $x_{i_1+n}^{\sigma_1} \vee x_{i_2+n}^{\sigma_2} \vee \cdots \vee x_{i_k+n}^{\sigma_k}$, for any $n \in \mathbb{Z}$. In the resulting system, called Shift-Resolution, one can prove only the statements provable in the classical Resolution. Indeed, every application of the shifting rule can be eliminated by deriving each shifted clause from scratch: this is possible, because the formula contains all shifts of the original clauses. However, as will be shown in Section 6, there is a formula, for which a Shift-Resolution proof is exponentially shorter than any classical proofs.

The shifting rule can be similarly added to other proof systems, such as Cutting Plane, Polynomial Calculus, etc. For a proof system Π, its extension with the shifting rule is denoted by Shift-Π.

5 Lower bounds on the size of shifted proofs

Lower bounds on the size of proofs with shifting can be inferred from the known lower bounds on classical proofs, as follows. Let φ_n be an unsatisfiable CNF formula in variables $x_1, \ldots, x_n$. This formula shall be encoded in a Shift-CNF formula Φ_n, in a way that for every proof system Π, from any Shift-Π proof of Φ_n, one could extract a (typically, smaller) classical Π-proof of φ_n. Then, every known lower bound on the size a Π-proof of φ_n translates to a lower bound on the size of Shift-Π proof of Φ_n.

The general idea of encoding a CNF formula φ in a Shift-CNF Φ is that every satisfying assignment $x_1, \ldots, x_n$ to φ should be repeated as something like an infinite binary string $(x_1 \ldots x_n)^\infty$ representing a satisfying assignment to Φ. The main challenge is that φ is not designed to be shifted, and therefore Φ should somehow apply φ only *to every n-th substring* of length n, that is, $x_1 \ldots x_n$, and not to any improperly shifted substrings $x_i \ldots x_n x_1 \ldots x_{i-1}$, with $i \in \{2, \ldots, n\}$. Since, by definition, shifted formulas apply to all shifts, this selective evaluation is not possible, and it is necessary to use some kind of encoding that would disable all unintended shifts.

The proposed encoding of φ represents each of its variables x_i as four consecutive Boolean variables: y_{4i+1}, y_{4i+2}, y_{4i+3} and y_{4i+4}. The first three of them shall always have values 011, whereas the last variable, y_{4i+4}, holds the actual value of x_i. In order to distinguish the encoded variable x_1, a special separator code 0100 is inserted between every two complete blocks of n encoded variables.

The formula Φ_n is a conjunction of two parts: the first part W_n ensures that the infinite string representing the variable values is a valid encoding of the form described above, while H_n simulates φ over that encoding.

The formula W_n has to make sure that the infinite string is of the form $(\$\{\widetilde{0}, \widetilde{1}\}^n)^\infty$, where $\$ = 0100$, $\widetilde{0} = 0110$ and $\widetilde{1} = 0111$. The first task towards this goal is to define all sequences of the form $\{\$, \widetilde{0}, \widetilde{1}\}^\infty$. This is done by nine constraints (Shift-CNF clauses). First, all substrings of length 5 that do not contain the control pair 01 are forbidden: namely, 11111, 11110, 11100, 11000, 10000 and 00000. Two more forbidden partial substrings 01□□1 and 01□□00 ensure that for each control pair 01, after two symbols, there cannot be anything except another control pair 01. The last forbidden substring 0101 makes sure that the data digits between two control pairs cannot be 01.

It remains to ensure that separators ($) never occur close to each other, and that there is a separator after every n encoded digits. The former is done by adding n forbidden partial strings $0100\square^{4k}0100$, for all $k \in \{0, ..., n-1\}$, and the existence of separators is asserted by prohibiting $n+1$ subsequent encoded digits using a partial string $(011\square)^{n+1}$. This completes the formula W_n.

The second part of the formula, denoted by H_n, contains as many clauses as φ_n. Whenever φ_n contains a clause $x_{i_1}^{\sigma_1} \vee \ldots \vee x_{i_k}^{\sigma_k}$, it is represented in H_n by the following corresponding clause.

$$\underbrace{y_1 \vee \neg y_2 \vee y_3 \vee y_4}_{D_\$\text{: false only on 0100 (\$)}} \quad \vee \quad \underbrace{y_{4i_1+4}^{\sigma_1} \vee \ldots \vee y_{4i_k+4}^{\sigma_k}}_{p(x_{i_1}^{\sigma_1} \vee \ldots \vee x_{i_k}^{\sigma_k})}$$

The disjunction of the first four literals is true, unless there is a substring 0100 there, that is, the separator ($). For that reason, any unintended shifts of this clause hold true, and are therefore irrelevant. On a correct shift, the first four literals are false, and the rest, denoted by $p(C)$, correctly apply the original clause C to the encoded variables of φ_n.

Each satisfying assignment to the Shift-CNF formula $\text{Shifts}(W_n \wedge H_n)$ encodes at least one satisfying assignment to the original CNF formula φ, and since the latter is unsatisfiable by assumption, so is $\text{Shifts}(W_n \wedge H_n)$.

The proposed lower bound method applies to a class of proof systems, so that a lower bound on the size of a Π-proof of φ_n, where Π is a proof system, implies a fairly close lower bound on the size of Shift-Π proofs for $\text{Shifts}(W_n \wedge H_n)$. To keep things simple, in this extended abstract, the theorem is formulated for three well-known proof methods, and actually established only for the case of Resolution (other cases are similar).

▶ **Theorem 13.** *Let Π be one of the three proof systems: Resolution, Cutting Plane or Polynomial Calculus. Then the length of any Shift-Π refutation of $\text{Shifts}(W_n \wedge H_n)$ is at least $\Omega\left(\frac{S_\Pi(\varphi_n)}{n}\right)$, where $S_\Pi(\varphi_n)$ is the length of the shortest Π-refutation of φ_n. The same holds true for the total size of refutations.*

The proof consists of two parts. First, a Shift-Π refutation of $\text{Shifts}(W_n \wedge H_n)$ is transformed to a Π-refutation of the same formula, by mapping each variable y_i, with $i \in \mathbb{Z}$, to $y_{(i \bmod 4n+4)}$ and then eliminating the shift rules. Then the latter Π-refutation of $\text{Shifts}(W_n \wedge H_n)$ is transformed by substituting the sequence $\$(011\square)^n$ into all auxiliary variables, resulting in a Π-refutation of φ_n of the stated size.

Proof for the case of Resolution. Consider any refutation π of $\text{Shifts}(W_n \wedge H_n)$ in the Shift-Resolution proof system, and let λ_n be its length. Let σ be a substitution that maps each variable y_i, with $i \in \mathbb{Z}$, to the variable $y_{(i \bmod m)}$, where $m = 4(n+1)$. Then, $\pi[\sigma]$ denotes the sequence of clauses in π under the substitution σ.

As stated in the following lemma, this substituted refutation remains a valid resolution refutation.

▶ **Lemma 14.** *Let $C_1, C_2, \ldots, C_s$ be a Resolution refutation of a set of clauses F, and let τ be a substitution of a variable with another variable ($x := y$) or with a constant ($x := 0$ or $x := 1$). Then, the list of substituted clauses $C_1[\tau], C_2[\tau], \ldots, C_s[\tau]$, with all constant true clauses omitted, is a valid Resolution refutation of $F[\tau]$.*

For every clause C with variables from $\Gamma = \{y_i\}_{i \in \mathbb{Z}}$, under the substitution σ, there are at most m distinct shifts of C. Hence, the size of the formula $\text{Shifts}(W_n \wedge H_n)[\sigma]$ is at most m times the size of $W_n \wedge H_n$. In order to transform the proof $\pi[\sigma]$ to a Resolution refutation of the formula $\text{Shifts}(W_n \wedge H_n)[\sigma]$, one should eliminate the shift rules. For that purpose, along with every clause, all its shifts need to be deduced as well. Let π' be the resulting refutation, which is of size at most $m\lambda_n$.

Consider a substitution into π', defined by $y_0 \dots y_{m-1} := \$(011\Box)^n$, where each square ($\Box$) indicates a variable unaffected by the substitution. Under such a substitution, all clauses of $\text{Shifts}(W_n)[\sigma]$ are satisfied. The clauses of $\text{Shifts}(H_n)[\sigma]$ are either satisfied or are reduced to clauses of the form $p(C)$, where C is a clause from φ_n. In the end, all such clauses are obtained. Let $p(\varphi_n)$ denote their conjunction. By Lemma 14, there is a derivation that contains no true clauses, that is, a refutation of the formula $p(\varphi_n)$. Also, the lemma asserts that the length of the proof is not increased.

Thus, a Resolution refutation of φ_n of size at most $m\lambda_n$ has been obtained. Therefore, $\lambda_n \geqslant \Omega\left(\frac{S_\Pi(\varphi_n)}{n}\right)$. ◀

▶ **Corollary 15.** *For each number $n \geqslant 1$, there exists a 3-CNF formula φ_n of n variables and with $O(n)$ clauses, such that every Shift-Resolution proof of the corresponding Shift-CNF Φ_n is of size at least $2^{\Omega(n)}$.*

Proof. It is sufficient to take any family of formulas with Resolution proof complexity $2^{\Omega(n)}$. Such a family is constructed, for instance, by Urquhart [16]. ◀

▶ **Corollary 16.** *There exists such a CNF formula φ_n of size n, that every Shift-Cutting Plane proof of the corresponding Shift-CNF Φ_n is of size at least $2^{n^{\Omega(1)}}$.*

Proof. The proof uses a family of formulas with the Cutting Plane proof complexity $2^{n^{\Omega(1)}}$. Such formulas were constructed by Pudlák [14]. ◀

▶ **Corollary 17.** *For some CNF formula φ_n of size $O(n^2)$, the size of every Shift-Polynomial Calculus proof of the corresponding Shift-CNF Φ_n is at least $2^{n^{\Omega(1)}}$.*

Proof. The proof uses the formulas that encode the pigeonhole principle $\mathrm{PHP}_\mathrm{n}^{\mathrm{n}+1}$. By the results of Razborov [15] and of Impagliazzo et al. [10], every Polynomial Calculus derivation of $\mathrm{PHP}_\mathrm{n}^{\mathrm{n}+1}$ is of size at least $2^{n^{\Omega(1)}}$. ◀

6 Separation of Resolution with and without shift

In this section, it is shown that, in some cases, Shift-Resolution can be exponentially more succinct than classical proof systems without shifts. This is proved by presenting a certain false formula, which has a small refutation in Shift-Resolution, whereas in classical proof systems, it requires exponential-size refutations.

For a constant $n \geqslant 1$, the formula Ψ_n asserts the existence of an infinite string of the form $\dots\$w_{-1}\$w_0\$w_1\dots$, where each w_i is an n-digit binary notation of a certain natural number, and every subsequent number in the list is greater by 1 than the previous number. For every number $i \in \{0, \dots, 2^n - 1\}$, let $\text{bin}(i) \in \{0,1\}^n$ be its n-bit binary representation. The longest finite string, on which this formula is true, is $\$\text{bin}(0)\$\text{bin}(1)\$\dots\$\text{bin}(2^n-1)\$$, but for any longer string, in particular for any infinite string, the counter eventually overflows and the formula becomes false. In view of Lemma 1, this formula contains a finite set of contradictory clauses, and hence is subject to classical proof methods.

The construction of the formula is based on the encoding of digits and separators given in Section 5. In particular, the formula $\text{Shifts}(W_n)$ ensuring that the infinite string is of the form $(\$\{\widetilde{0},\widetilde{1}\}^n)^\infty$, where $\$ = 0100$, $\widetilde{0} = 0110$ and $\widetilde{1} = 0111$, is used again, and so is the clause $D_\$ = y_1 \vee \neg y_2 \vee y_3 \vee y_4$ that identifies a separator (\$) beginning at y_1.

With the syntactic structure defined by the formula W_n, the desired counter is implemented by a CNF formula $Step^n_k(x_1, x_2, \dots, x_n; y_1, y_2, \dots, y_n)$, with $n \geqslant 1$ and $k \in \{0, 1, 2, \dots, n-1\}$, which is true if and only if the binary number $(x_1x_2 \dots x_n)_2$ is greater than $(y_1y_2 \dots y_n)_2$ exactly by 2^k. There is a formula with this property that contains $\Theta(n)$ clauses of constant size.

Given two propositions $Step^n_k$ about adding 2^k, one asserting that $x + 2^k = y$ and the other that $y + 2^k = z$, one can infer from them that $x + 2^{k+1} = z$, that is, a proposition using the formula $Step^n_{k+1}$. The next lemma formalizes this intuition, and shows that this inference can be carried out using resolutions.

▶ **Lemma 18.** *For any $n \geqslant 1$ and $k \in \{0, 1, 2, \dots, n-2\}$, given all clauses of the CNF formula $Step^n_k(x_1, x_2, \dots, x_n; y_1, y_2, \dots, y_n) \wedge Step^n_k(y_1, y_2, \dots, y_n; z_1, z_2, \dots, z_n)$, all clauses of $Step^n_{k+1}(x_1, x_2, \dots, x_n; z_1, z_2, \dots, z_n)$ can be derived using $O(n)$ resolutions.*

For every CNF formula $\varphi = C_1 \wedge \dots \wedge C_k$ and for every clause D, the CNF formula obtained from φ by adding all literals from D into every clause is denoted by $D \vee \varphi = (D \vee C_1) \wedge \dots \wedge (D \vee C_k)$.

Furthermore, denote by V_n a CNF formula containing the following clauses which assert that after the current separator (\$), there is another one $4n$ symbols later: $D_\$ \vee \neg x_{4n+5}$, $D_\$ \vee x_{4n+6}$, $D_\$ \vee \neg x_{4n+7}$ and $D_\$ \vee \neg x_{4n+8}$. These conditions actually follow from W_n, but it is more convenient to add them than to derive them using Resolution.

In this notation, the formula separating classical proof systems from Shift-Resolution is constructed as follows.

▶ **Theorem 19.** *For every classical proof system Π, every Π-refutation of the following formula is of size $\Omega(2^n)$.*

$$\Psi_n = \text{Shifts}\big(W_n \wedge \big(D_\$ \vee Step^n_0(x_8, x_{12}, \dots, x_{4n+4}; x_{4n+12}, x_{4n+16}, \dots, x_{8n+8})\big) \wedge V_n \wedge (D_\$ \vee \neg x_8)\big)$$

At the same time, there exists a Shift-Resolution refutation of Ψ_n of size $\text{poly}(n)$.

The first part of Ψ_n is W_n, which enforces the syntactic structure of the string. The second part $(D_\$ \vee Step^n_0(x_8, x_{12}, \dots, x_{4n+4}; x_{4n+12}, x_{4n+16}, \dots, x_{8n+8})$ states that any two subsequent values of the counter differ by 1. The last part $(D_\$ \vee \neg x_8)$, requires the highest digit of the counter to be 0. The formula Ψ_n is unsatisfiable, because, after a series of incrementations, the highest digit shall eventually become 1.

Sketch of a proof. The lower bound on the size of Π-refutations of Ψ_n is based on the fact that every such refutation must use more than $\frac{1}{3}2^{n-2}$ clauses of this formula. This is proved by showing that the conjunction of any $\frac{1}{3}2^{n-2}$ clauses of Ψ_n is satisfiable.

The key element of a small Shift-Resolution refutation of Ψ_n is the use of Lemma 18. The formula contains a clause about incrementing the counter by 1; by Lemma 18, it can be shifted, and two such clauses resolved, to obtain a clause about incrementing by 2. The latter clause can be again shifted and resolved, resulting in a clause about adding 4, and so on. This gives a proof of an appropriate $Step^n_{n-1}$ formula, in $\Theta(n)$ steps. A contradiction is obtained by resolving that formula with other clauses of Ψ_n. ◀

7 Conclusion

An interesting direction for further research would be to prove a lower bound for any proof system with shift, for which no non-trivial lower bounds are known in the classical case, such as for the Lovász–Schrijver proof system. This task may potentially be easier than proving a lower bound in the classical case, because some instances of shift-CNF encode harder problems, such as PSPACE-complete problems, and therefore proving lower bounds on their proof complexity could actually be less difficult.

Acknowledgements. The authors are grateful to Alexander Shen for bringing the resolution method for string avoidability to their attention, and to Juhani Karhumäki for inspiring discussions.

The research presented in Sections 3–5 was supported by Russian Science Foundation (project 16-11-10123).

References

1 Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

2 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 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 154–169, 2014.

3 Olaf Beyersdorff, Leroy Chew, and Mikolas Janota. On unification of QBF resolution-based calculi. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 81–93, 2014.

4 Olaf Beyersdorff, Leroy Chew, and Mikolás Janota. Proof complexity of resolution-based QBF calculi. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 76–89, 2015.

5 Brandon Blakeley, Francine Blanchet-Sadri, Josh Gunter, and Narad Rampersad. On the complexity of deciding avoidability of sets of partial words. *Theor. Comput. Sci.*, 411(49):4263–4271, 2010.

6 Francine Blanchet-Sadri, Raphaël M. Jungers, and Justin Palumbo. Testing avoidability on sets of partial words is hard. *Theor. Comput. Sci.*, 410(8-10):968–972, 2009.

7 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.

8 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, March 1979.

9 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

10 Russell Impagliazzo, Pavel Pudlák, and Jirí Sgall. Lower bounds for the polynomial calculus and the gröbner basis algorithm. *Computational Complexity*, 8(2):127–144, 1999.

11 Mikolás Janota and Joao Marques-Silva. Expansion-based QBF solving versus q-resolution. *Theor. Comput. Sci.*, 577:25–42, 2015.

12 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.

13 M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, 2002. Cambridge Books Online.

14 Pavel Pudlak. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symbolic Logic*, 62(3):981–998, 1997.

15 Alexander A. Razborov. Lower bounds for the polynomial calculus. *Computational Complexity*, 7(4):291–324, 1998.

16 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, 1987.

Deciding Semantic Finiteness of Pushdown Processes and First-Order Grammars w.r.t. Bisimulation Equivalence

Petr Jančar

Dept. of Computer Science, FEI, Technical University Ostrava, Czech Republic
petr.jancar@vsb.cz

Abstract

The problem if a given configuration of a pushdown automaton (PDA) is bisimilar with some (unspecified) finite-state process is shown to be decidable. The decidability is proven in the framework of first-order grammars, which are given by finite sets of labelled rules that rewrite roots of first-order terms. The framework is equivalent to PDA where also deterministic popping epsilon-steps are allowed, i.e. to the model for which Sénizergues showed an involved procedure deciding bisimilarity (FOCS 1998). Such a procedure is here used as a black-box part of the algorithm. For deterministic PDA the regularity problem was shown decidable by Valiant (JACM 1975) but the decidability question for nondeterministic PDA, answered positively here, had been open (as indicated, e.g., by Broadbent and Goeller, FSTTCS 2012).

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases pushdown processes, first-order grammars, bisimulation, regularity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.52

1 Introduction

The question of deciding semantic equivalences of systems, like language equivalence, has been a frequent topic in computer science. A closely related question asks if a given system in a class $\mathcal{C}_1$ has an equivalent in a simpler class $\mathcal{C}_2$. Pushdown automata (PDA) constitute a well-known example. Language equivalence and regularity are undecidable for PDA. In the case of deterministic PDA (DPDA), the decidability and complexity results for regularity (see [13] and the references therein) preceded the famous decidability result for equivalence by Sénizergues [9].

In concurrency theory, logic, verification, and other areas, a finer equivalence, called *bisimulation equivalence* or *bisimilarity*, has emerged as another fundamental behavioural equivalence; on deterministic systems it essentially coincides with language equivalence. An on-line survey of the results which study this equivalence in a specific area of process rewrite systems is maintained by Srba [11].

Among the most involved results in this area is the decidability of bisimilarity for pushdown processes, generated by (nondeterministic) PDA with only deterministic and popping ε-steps; this was shown by Sénizergues [10] who thus generalized his above mentioned result for DPDA. There is no known upper bound on the complexity of this decidable problem. The nonelementary lower bound established in [1] is, in fact, TOWER-hardness in the terminology of [8], and it holds even for real-time PDA, i.e. PDA with no ε-steps. For the above mentioned PDA with deterministic and popping ε-steps the bisimilarity problem is even not primitive recursive, its Ackermann-hardness is shown in [5]. In the deterministic case, the equivalence problem is known to be PTIME-hard, and has a primitive recursive upper bound shown by Stirling [12] (where a finer analysis places the problem in TOWER [5]).

Extrapolating the deterministic case, we might expect that for PDA the "regularity" problem w.r.t. bisimilarity (asking if a given PDA-configuration is bisimilar with a state in a finite-state system) is decidable as well, and that this problem might be easier than the equivalence problem solved in [10]; "only" EXPTIME-hardness is known here (see [7], and [11] for detailed references). Nevertheless, this decidability question has been open so far, as also indicated in [2] (besides [11]).

Contribution of this paper. We show that semantic finiteness of pushdown configurations w.r.t. bisimilarity is decidable. The decidability is proven in the framework of *first-order grammars*, i.e. of finite sets of labelled rules that rewrite roots of first-order terms. The framework is equivalent to PDA where also deterministic and popping ε-steps are allowed, i.e. to the model to which Sénizergues's general decidability proof [10] applies. (A simplified proof directly in the first-order grammar framework is given in [4].) The presented algorithm, answering if a given configuration, i.e. a first-order term in the labelled transition system generated by a first-order grammar, has a bisimilar finite-state system, uses the result of [10] (or of [4]) as a black-box procedure. By [5] we cannot get a primitive recursive upper bound via a black-box use of the decision procedure for bisimilarity.

Semidecidability of the semantic finiteness problem has been long clear, hence it is the existence of finite effectively verifiable witnesses of the negative case that is the crucial point here. Such witnesses are shown by considering "limits" of repeated substitutions, resulting in regular terms (i.e.infinite terms with only finitely many subterms). Some finite paths with "pumpable" segments are shown to be increasing the "equivalence-level" with the respective limit above any bound while never reaching the equivalence class of the limit. The (black-box) procedure deciding equivalence is used to show a verifiable bound on the number of segment-pumpings that allows to confirm the witness property of a path.

A full version of this paper is planned to appear as the second version of the paper at `http://arxiv.org/abs/1305.0516`; it will contain detailed proofs.

2 Basic Notions and Result

In this section we define the basic notions and state the result in the form of a theorem. Some standard definitions are restricted when we do not need the full generality. We finish the section by a note about a transformation of pushdown automata to first-order grammars.

By $\mathbb{N}$ and $\mathbb{N}_+$ we denote the sets of nonnegative integers and of positive integers, respectively. By $[i,j]$ we denote the set $\{i, i{+}1, \dots, j\}$. For a set $\mathcal{A}$, by $\mathcal{A}^*$ we denote the set of finite sequences of elements of $\mathcal{A}$, which are also called *words* (over $\mathcal{A}$). By $|w|$ we denote the *length* of $w \in \mathcal{A}^*$, and by ε the *empty sequence* (hence $|\varepsilon| = 0$). We put $\mathcal{A}^+ = \mathcal{A}^* \smallsetminus \{\varepsilon\}$.

Labelled transition systems. A *labelled transition system*, an *LTS* for short, is a tuple $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a\in\Sigma})$ where $\mathcal{S}$ is a finite or countable set of *states*, Σ is a finite set of *actions* (or *letters*), and $\xrightarrow{a} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of *a-transitions* (for each $a \in \Sigma$). We say that $\mathcal{L}$ is a *deterministic LTS* if for each pair $s \in \mathcal{S}$, $a \in \Sigma$ there is at most one s' such that $s \xrightarrow{a} s'$ (which stands for $(s, s') \in \xrightarrow{a}$). By $s \xrightarrow{w} s'$, where $w = a_1 a_2 \dots a_n \in \Sigma^*$, we denote that there is a *path* $s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_n} s_n = s'$; if $s \xrightarrow{w} s'$, then s' is *reachable from* s. By $s \xrightarrow{w}$ we denote that w is *enabled in* s, i.e., $s \xrightarrow{w} s'$ for some s'. If $\mathcal{L}$ is deterministic, then $s \xrightarrow{w} s'$ or $s \xrightarrow{w}$ denotes a unique path.

Bisimilarity. Given $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a\in\Sigma})$, we say that a *set* $\mathcal{B} \subseteq \mathcal{S} \times \mathcal{S}$ *covers* $(s,t) \in \mathcal{S} \times \mathcal{S}$ if for any $s \xrightarrow{a} s'$ there is $t \xrightarrow{a} t'$ such that $(s',t') \in \mathcal{B}$, and for any $t \xrightarrow{a} t'$ there is $s \xrightarrow{a} s'$ such that $(s',t') \in \mathcal{B}$. For $\mathcal{B}, \mathcal{B}' \subseteq \mathcal{S} \times \mathcal{S}$ we say that $\mathcal{B}'$ *covers* $\mathcal{B}$ if $\mathcal{B}'$ covers each $(s,t) \in \mathcal{B}$. A set $\mathcal{B} \subseteq \mathcal{S} \times \mathcal{S}$ is a *bisimulation* if $\mathcal{B}$ covers $\mathcal{B}$. States $s,t \in \mathcal{S}$ are *bisimilar*, written $s \sim t$, if there is a bisimulation $\mathcal{B}$ containing (s,t). A standard fact is that $\sim \subseteq \mathcal{S} \times \mathcal{S}$ is an equivalence relation, and it is the largest bisimulation, the union of all bisimulations.

Semantic finiteness. Given $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a\in\Sigma})$, we say that $s_0 \in \mathcal{S}$ is *finite up to bisimilarity*, or *bisim-finite* for short, if there is some state f in some finite LTS such that $s_0 \sim f$; otherwise s_0 is *infinite up to bisimilarity*, or *bisim-infinite*. (When comparing states from different LTSs, we implicitly refer to the disjoint union of these LTSs.)

First-order terms, regular terms, finite graph presentations. We will consider LTSs in which states are first-order regular terms. They are built from *variables* from a fixed countable set $\textsc{Var} = \{x_1, x_2, x_3, \dots\}$ and from *function symbols*, also called *(ranked) nonterminals*, from some specified finite set $\mathcal{N}$; each $A \in \mathcal{N}$ has $arity(A) \in \mathbb{N}$. (An example of a finite term is $C(D(x_3, B), x_2)$, where the arities of B, C, D are $0, 2, 2$, respectively.)

Transitions will be determined by a finite set of (schematic) *root-rewriting* rules (that can be exemplified by $A(x_1, x_2, x_3) \xrightarrow{b} C(D(x_3, B), x_2)$, where x_1, x_2, x_3 serve as the "place-holders" for the depth-1 subterms of a term with the root A that might be rewritten by performing action b). We will now formalize this, making also some conventions on the use of (finite and infinite) terms and substitutions.

We identify terms with their syntactic trees. Thus a *term over* $\mathcal{N}$ is (viewed as) a rooted, ordered, finite or infinite tree where each node has a label from $\mathcal{N} \cup \textsc{Var}$; if the label of a node is $x_i \in \textsc{Var}$, then the node has no successors, and if the label is $A \in \mathcal{N}$, then it has m (immediate) successor-nodes where $m = arity(A)$. A subtree of a term E is also called a *subterm* of E. We make no difference between isomorphic (sub)trees, and thus a subterm can have more (maybe infinitely many) *occurrences* in E. Each *subterm-occurrence* has its (nesting) *depth in* E, which is its (naturally defined) distance from the root of E. We also use the standard notation for terms: we write $E = x_i$ or $E = A(G_1, \dots, G_m)$ with the obvious meaning; in the latter case we have $\textsc{root}(E) = A \in \mathcal{N}$, $m = arity(A)$, and $G_1, \dots, G_m$ are the ordered occurrences of depth-1 subterms of E.

A *term* is *finite* if the respective tree is finite. A (possibly infinite) *term* is *regular* if it has only finitely many subterms (though the subterms may be infinite and can have infinitely many occurrences). We note that any regular term has at least one *finite-graph presentation*, i.e. a finite directed graph, with a designated root, where each node has a label from $\mathcal{N} \cup \textsc{Var}$; if the label of a node is $x_i \in \textsc{Var}$, then the node has no outgoing arcs, if the label is $A \in \mathcal{N}$, then it has m ordered outgoing arcs where $m = arity(A)$. The standard tree-unfolding of the graph is the respective term, which is infinite if there are cycles in the graph. The nodes in the least presentation of E are bijectively mapped onto (the roots of) the subterms of E.

In what follows, by a "term" we mean a "regular term" unless the context makes clear that the term is finite. (We do not consider non-regular terms.) We reserve symbols A, B, C, D to range over nonterminals, and E, F, G, H to range over (regular) terms.

Substitutions, associative composition, limits of infinite compositions. By $\textsc{Terms}_{\mathcal{N}}$ we denote the set of all (regular) terms over a set $\mathcal{N}$ of (ranked) nonterminals (and over the set $\textsc{Var}$ of variables). A *substitution* σ is a mapping $\sigma : \textsc{Var} \to \textsc{Terms}_{\mathcal{N}}$ whose *support* $\textsc{supp}(\sigma) = \{x_i \mid \sigma(x_i) \neq x_i\}$ is *finite*; we reserve the symbol σ for substitutions. By *applying* *a substitution* σ to a term E we get the term $E\sigma$ that arises from E by replacing each occurrence of x_i with $\sigma(x_i)$; given graph presentations, in the graph of E we just redirect each arc leading to x_i towards the root of $\sigma(x_i)$ (which includes the special "root-designating arc" when $E = x_i$). Hence $E = x_i$ implies $E\sigma = x_i\,\sigma = \sigma(x_i)$.

The natural *composition of substitutions*, where $\sigma = \sigma_1\sigma_2$ is defined by $x_i\sigma = (x_i\sigma_1)\sigma_2$, can be easily verified to be associative. We thus write simply $E\sigma_1\sigma_2$ when meaning $(E\sigma_1)\sigma_2$ or $E(\sigma_1\sigma_2)$. We let σ^0 be the empty-support substitution, and we put $\sigma^{i+1} = \sigma\sigma^i$. If σ is *guarded*, which means that $x_i\sigma = x_j$ implies $i = j$ (in other words, for each $x_i \in \textsc{supp}(\sigma)$ the root of $E_i = x_i\sigma$ is a nonterminal "guarding" the occurrences of variables in E_i), then even the limit σ^ω is well-defined: "operationally", to get graph presentations of terms $x_i\sigma^\omega$ from graph presentations of $x_i\sigma$, for all $x_i \in \textsc{supp}(\sigma)$, we redirect any arc leading to x_j, where $x_j \in \textsc{supp}(\sigma)$, towards the root of (the presentation of) $x_j\sigma$. We note that no variable $x_i \in \textsc{supp}(\sigma)$ occurs in any term $E\sigma^\omega$, for any guarded substitution σ; such variables "disappear" by applying σ^ω. (Hence $E\sigma^\omega$ can only contain variables x_i for which $x_i\sigma = x_i$.)

First-order grammars. A *first-order grammar*, or just a *grammar* for short, is a tuple $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$ where $\mathcal{N} = \{A_1, A_2, \dots\}$ is a finite set of ranked *nonterminals*, viewed as function symbols with arities, $\Sigma = \{a_1, a_2, \dots\}$ is a finite set of *actions* (or letters), and $\mathcal{R} = \{r_1, r_2, \dots\}$ is a finite set of *rules* of the form

$$A(x_1, x_2, \dots, x_m) \xrightarrow{a} E \tag{1}$$

where $A \in \mathcal{N}$, $arity(A) = m$, $a \in \Sigma$, and E is a *finite* term over $\mathcal{N}$ in which each occurring variable is from the set $\{x_1, x_2, \dots, x_m\}$. We can exemplify the rules by $A(x_1, x_2, x_3) \xrightarrow{b} C(D(x_3, B), x_2)$, $A(x_1, x_2, x_3) \xrightarrow{b} x_2$, $D(x_1, x_2) \xrightarrow{a} A(D(x_2, x_2), x_1, B)$; here the arities of A, B, C, D are $3, 0, 2, 2$, respectively.

A rule $A(x_1, x_2, \dots, x_m) \xrightarrow{a} E$ will generate a-transitions $A(x_1, x_2, \dots, x_m)\sigma \xrightarrow{a} E\sigma$ for all substitutions σ. The concrete rule $A(x_1, x_2, x_3) \xrightarrow{b} C(D(x_3, B), x_2)$ generates the transitions like $A(x_1, x_2, x_3) \xrightarrow{b} C(D(x_3, B), x_2)$ and $A(x_5, x_5, x_2) \xrightarrow{b} C(D(x_2, B), x_5)$, and more generally $A(G_1, G_2, G_3) \xrightarrow{b} C(D(G_3, B), G_2)$ for any (regular) terms G_1, G_2, G_3. The rule $A(x_1, x_2, x_3) \xrightarrow{b} x_2$ generates $A(G_1, G_2, G_3) \xrightarrow{b} G_2$. We now give a more formal definition.

LTSs generated by grammars. Given $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$, by $\mathcal{L}^{\text{R}}_{\mathcal{G}}$ we denote the (*rule-based*) LTS $\mathcal{L}^{\text{R}}_{\mathcal{G}} = (\textsc{Terms}_{\mathcal{N}}, \mathcal{R}, (\xrightarrow{r})_{r\in\mathcal{R}})$ where each rule r of the form $A(x_1, x_2, \dots, x_m) \xrightarrow{a} E$ induces transitions $A(x_1, \dots, x_m)\sigma \xrightarrow{r} E\sigma$ for any substitution σ (also unguarded; we can have $x_i\sigma = x_j$ for $i \neq j$). Thus the rule $A(x_1, \dots, x_m) \xrightarrow{r} E$ is itself a transition, using σ with $\textsc{supp}(\sigma) = \emptyset$.
The LTS $\mathcal{L}^{\text{R}}_{\mathcal{G}}$ is deterministic, since for each F and r there is at most one H such that $F \xrightarrow{r} H$. We note that *variables* are *dead* (have no outgoing transitions), and *transitions cannot add variables*, i.e., $F \xrightarrow{w} H$ implies that each variable occurring in H also occurs in F (but not necessarily vice versa).

Since the rhs (right-hand sides) E in the rules (1) are finite, all terms reachable from a finite term are finite. (It is convenient to have the rhs finite while including regular terms into our LTSs; the other options are in principle equivalent.)

The deterministic rule-based LTS $\mathcal{L}^{\text{R}}_{\mathcal{G}}$ is helpful technically, but we are primarily interested in the (generally nondeterministic) *action-based* LTS $\mathcal{L}^{\text{A}}_{\mathcal{G}} = (\textsc{Terms}_{\mathcal{N}}, \Sigma, (\xrightarrow{a})_{a\in\Sigma})$ where each rule $A(x_1,\ldots,x_m) \xrightarrow{a} E$ induces the transitions $A(x_1,\ldots,x_m)\sigma \xrightarrow{a} E\sigma$ for all substitutions σ.

Given a grammar $\mathcal{G} = (\mathcal{N},\Sigma,\mathcal{R})$, two *terms* from $\textsc{Terms}_{\mathcal{N}}$ are *bisimilar* if they are bisimilar as states in the action-based LTS $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$. By our definitions all variables are bisimilar, since they are dead terms. The variables serve us primarily as "place-holders for subterm-occurrences" in terms (that might themselves be variable-free); such a use of variables has been already exemplified in the rules (1).

Main result, and its relation to pushdown automata. We now state the theorem, to be proven in the next section, and we mention why the result also applies to pushdown automata (PDA) with deterministic popping ε-steps.

▶ **Theorem 1.** *There is an algorithm that, given a grammar $\mathcal{G} = (\mathcal{N},\Sigma,\mathcal{R})$ and (a finite presentation of) $E_0 \in \textsc{Terms}(\mathcal{N})$, decides if E_0 is bisim-finite (i.e., if $E_0 \sim f$ for a state f in some finite LTS).*

A transformation of (nondeterministic) PDA in which deterministic popping ε-steps are allowed to first-order grammars (with no ε-steps) is recalled in the full arxiv-version. This makes clear that the semantic finiteness of PDA with deterministic popping ε-steps (w.r.t. bisimilarity) is also decidable. In fact, the problems are interreducible; the close relationship between (D)PDA and first-order schemes has been long known (see, e.g., [3]). The proof of Theorem 1 presented here uses the fact that bisimilarity of first-order grammars is decidable; this was shown for the above mentioned PDA model by Sénizergues [10], and a direct proof in the first-order-term framework was presented in [4]. We note that for PDA where popping ε-steps can be in conflict with "visible" steps bisimilarity is already undecidable [6]; hence the proof presented here does not yield the decidability of semantic finiteness in this more general model.

3 Proof of Theorem 1

3.1 Computability of eq-levels, and semidecidability of bisim-finiteness

We will note that the semidecidability of bisim-finiteness is clear, but we first recall the computability of eq-levels, which is one crucial ingredient in our proof of semidecidability of bisim-infiniteness.

Stratified equivalence, and eq-levels. Assuming an LTS $\mathcal{L} = (\mathcal{S},\Sigma,(\xrightarrow{a})_{a\in\Sigma})$, we put $\sim_0 = \mathcal{S}\times\mathcal{S}$, and define $\sim_{k+1} \subseteq \mathcal{S}\times\mathcal{S}$ (for $k \in \mathbb{N}$) as the set of pairs covered by $\sim_k$. (Hence $s \sim_{k+1} t$ iff for any $s \xrightarrow{a} s'$ there is $t \xrightarrow{a} t'$ such that $s' \sim_k t'$ and for any $t \xrightarrow{a} t'$ there is $s \xrightarrow{a} s'$ such that $s' \sim_k t'$.)

We easily verify that $\sim_k$ are equivalence relations, and that $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots\cdots \supseteq \sim$. For the (first infinite) ordinal ω we put $s \sim_\omega t$ if $s \sim_k t$ for all $k \in \mathbb{N}$; hence $\sim_\omega = \bigcap_{k\in\mathbb{N}} \sim_k$. It is standard (and can be easily checked) that $\bigcap_{k\in\mathbb{N}} \sim_k$ is a bisimulation in image-finite LTSs, and thus $\sim = \bigcap_{k\in\mathbb{N}} \sim_k = \sim_\omega$. We recall that $\mathcal{L}$ is *image-finite* if the set $\{s' \mid s \xrightarrow{a} s'\}$ is finite for each pair $s \in \mathcal{S}$, $a \in \Sigma$. Our grammar-generated LTSs $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$ are obviously image-finite (while $\mathcal{L}^{\mathrm{R}}_{\mathcal{G}}$ are even deterministic); we thus further assume image-finiteness.

We attach the *equivalence level* (eq-level) $\textsc{EqLv}(s,t) = \max\{k \in \mathbb{N}\cup\{\omega\} \mid s \sim_k t\}$ to each pair of states.

Eq-levels are computable for first-order grammars. We now state an important lemma that follows easily from the involved decidability proof in [10] (and a transformation to first-order grammars); as already mentioned, a proof given directly for the first-order grammars was presented in [4]. (This is surely a *fundamental theorem* in general, the name *lemma* has been chosen here to reflect that it is a prerequisite for the only theorem proven in this paper.)

▶ **Lemma 2.** *There is an algorithm that, given $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$ and $E_0, F_0 \in \textsc{Terms}(\mathcal{N})$, computes $\textsc{EqLv}(E_0, F_0)$ in $\mathcal{L}^{\mathsf{A}}_{\mathcal{G}}$ (and thus also decides if $E_0 \sim F_0$).*

Proof. For each fixed $k \in \mathbb{N}$ it is decidable if $E_0 \sim_k F_0$, as can be shown by a straightforward induction on k. The question $E_0 \overset{?}{\sim} F_0$, i.e. $E_0 \overset{?}{\sim}_\omega F_0$, can be decided by [10] (and [4]). ◀

Semidecidability of bisim-finiteness. Given $\mathcal{G}$ and E_0, we can systematically generate all finite LTSs, presenting them by first-order grammars with nullary nonterminals (which then coincide with states); for each state f of each generated system we can check if $E_0 \sim f$ by Lemma 2. In fact, Lemma 2 is not crucial here, since decidability of $E_0 \sim f$ can be shown in a much simpler way (see, e.g., [7]).

3.2 Semidecidability of bisim-infiniteness.

In Section 3.2.1 we note a few simple general facts on bisim-infiniteness, and also note the obvious compositionality (congruence properties) of bisimulation equivalence in our framework of first-order terms. In Section 3.2.2 we describe some finite structures that are candidates for witnessing bisim-infiniteness of a given term, and show an algorithm checking if a candidate is indeed a witness. In Section 3.2.3 we then show that each bisim-infinite term has a witness. Together this yields a proof of Theorem 1.

3.2.1 Some facts on bisim-infiniteness, and compositionality

Bisimilarity quotient. Given an LTS $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a\in\Sigma})$, the *quotient-LTS* $\mathcal{L}_\sim$ is the tuple $(\{\,[s]; s \in \mathcal{S}\,\}, \Sigma, (\xrightarrow{a})_{a\in\Sigma})$ where $[s] = \{s' \mid s' \sim s\}$, and $[s] \xrightarrow{a} [t]$ if $s' \xrightarrow{a} t'$ for some $s' \in [s]$ and $t' \in [t]$; in fact, $[s] \xrightarrow{a} [t]$ implies that for each $s' \in [s]$ there is $t' \in [t]$ such that $s' \xrightarrow{a} t'$. We have $s \sim [s]$, since $\{(s, [s]) \mid s \in \mathcal{S}\}$ is a bisimulation (in the union of $\mathcal{L}$ and $\mathcal{L}_\sim$). We refer to the states of $\mathcal{L}_\sim$ as to the *bisim-classes* (of $\mathcal{L}$).

A sufficient condition for bisim-infiniteness. We recall that $s_0 \in \mathcal{S}$ is *bisim-finite* if there is some state f in a finite LTS such that $s_0 \sim f$; otherwise s_0 is *bisim-infinite*. We observe that s_0 is bisim-infinite in $\mathcal{L}$ iff the reachability set of $[s_0]$ in $\mathcal{L}_\sim$, i.e. the set of states reachable from $[s_0]$ in $\mathcal{L}_\sim$, is infinite. The LTSs generated by first-order grammars are *finitely branching* (i.e., the set $\{s' \mid s \xrightarrow{a} s'$ for some $a\}$ is finite for each $s \in \mathcal{S}$), and we also use (one implication in) the following simple fact:

▶ **Proposition 3.** *A state s_0 of a finitely branching LTS is bisim-infinite iff there is an infinite path $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \cdots$ where $s_i \not\sim s_j$ for all $i \neq j$.*

To demonstrate that s_0 is bisim-infinite, it suffices to show that its reachability set contains states with arbitrarily large *finite* eq-levels w.r.t. a "test state" t. The sufficiency of this condition is based on the simple fact that $s \sim s'$ implies $\textsc{EqLv}(s, t) = \textsc{EqLv}(s', t)$. More formally:

▶ **Proposition 4.** *Given $\mathcal{L} = (\mathcal{S}, \Sigma, (\xrightarrow{a})_{a\in\Sigma})$ and states s_0, t, if for every $e \in \mathbb{N}$ there is s' that is reachable from s_0 and satisfies $e < \textsc{EqLv}(s', t) < \omega$, then s_0 is bisim-infinite.*

Eq-levels w.r.t. a test set in a bounded region. Our final general observation (tailored to a later use) is also straightforward: if two states are bisimilar, then the states in their equally bounded reachability regions must yield the same eq-levels when compared with states from a fixed (test) set. We formalize this observation as follows.

For any $s \in \mathcal{S}$ and $d \in \mathbb{N}$ (a distance, or a "radius") we put

$$\textsc{Region}(s,d) = \{s' \mid s \xrightarrow{w} s' \text{ for some } w \in \Sigma^* \text{ where } |w| \leq d\}.$$

For any $s \in \mathcal{S}$, $d \in \mathbb{N}$, and $\mathcal{T} \subseteq \mathcal{S}$ (a test set), we define the following subset of $\mathbb{N}$ (*finite* TestEqLevels):

$$\mathrm{TEL}(s,d,\mathcal{T}) = \{e \in \mathbb{N} \mid e = \textsc{EqLv}(s',t) \text{ for some } s' \in \textsc{Region}(s,d) \text{ and some } t \in \mathcal{T}\}.$$

▶ **Proposition 5.** *If* $\mathrm{TEL}(s,d,\mathcal{T}) \neq \mathrm{TEL}(s',d,\mathcal{T})$ *then* $s \not\sim s'$.

Compositionality of the states of the grammar-generated LTSs. Regarding the congruence properties, in principle it suffices for us to observe that if in a term E we replace a subterm F with F' such that $F' \sim F$ then the resulting term E' satisfies $E' \sim E$ (replacing a subterm with an equivalent one does not change the bisim-class). Hence $A(G_1,\dots,G_m) \not\sim A(G'_1,\dots,G'_m)$ implies that $G_i \not\sim G'_i$ for some $i \in [1,m]$. (This is surely not specific to bisimilarity.) Formally, we put $\sigma \sim \sigma'$ if $x_i\sigma \sim x_i\sigma'$ for each x_i, and we note:

▶ **Proposition 6.** *If* $\sigma \sim \sigma'$*, then* $E\sigma \sim E\sigma'$*.*
(Hence $E\sigma \not\sim E\sigma'$ *implies that* $x_i\sigma \not\sim x_i\sigma'$ *for some* x_i *occurring in* E*.)*

Conventions. We further consider only the *normalized* grammars $\mathcal{G} = (\mathcal{N},\Sigma,\mathcal{R})$, i.e. those satisfying the following condition: for any $A(x_1,\dots,x_m)$ and any $i \in [1,m]$ there is a word w such that $A(x_1,\dots,x_m) \xrightarrow{w} x_i$; hence for any E it is possible to "sink" to any of its subterm-occurrences by applying the grammar-rules. Such a normalization can be efficiently achieved by harmless modifications of the nonterminal arities and of the rules in $\mathcal{R}$, while the LTS $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$ remains the same up to isomorphism.

For convenience, in our notation we use m as the arity of all nonterminals in the considered grammar, though formally the *maximum* arity is meant. We will thus harmlessly write $A(G_1,\dots,G_m)$ instead of $A(G_1,\dots,G_{m_A})$ where $m_A = arity(A)$. (In fact, such uniformity can be achieved while keeping the above normalization condition, when a slight problem with arity 0 is handled; but this is not necessary for us to discuss.)

From now on, we view the expressions like $G \xrightarrow{w} H$ as referring to the deterministic LTS $\mathcal{L}^{\mathrm{R}}_{\mathcal{G}}$ (hence $w \in \mathcal{R}^*$), though $\sim_k$, $\sim$, and the eq-levels refer solely to (the action-based LTS) $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$.

3.2.2 Simple witnesses of bisim-infiniteness

We fix a grammar $\mathcal{G} = (\mathcal{N},\Sigma,\mathcal{R})$. Before defining the candidates for witnesses of bisim-infiniteness, we discuss some building segments of ("non-sinking") paths in the LTS $\mathcal{L}^{\mathrm{R}}_{\mathcal{G}}$.

Stairs, direct stairs, simple stairs, stairs eligible for "pumping". A nonempty sequence of rules $w = r_1r_2\dots r_\ell \in \mathcal{R}^+$ is a *stair* if we have $A(x_1,\dots,x_m) \xrightarrow{w} F$ where the rule r_1 is of the form $A(x_1,\dots,x_m) \xrightarrow{a} E$, and F has a nonterminal-root (hence F is not a variable x_i, i.e., the path $A(x_1,\dots,x_m) \xrightarrow{w} F$ does not "sink"); such w is a *direct stair* if there is no v such that $|v| < |w|$ and $A(x_1,\dots,x_m) \xrightarrow{v} F$. If (the above) w is a direct stair and F is a subterm of E (the right-hand side of r_1), then w is a *simple stair.* A *stair* w has the *type* (A, B) ("from A to B") if $A(x_1, \dots, x_m) \xrightarrow{w} F$ where $\textsc{root}(F) = B$.

(E.g., the sequence r_1r_2 of rules used in the path $A(G_1, G_2, G_3) \xrightarrow{r_1} C(D(G_3, B), G_2) \xrightarrow{r_2} D(G_3, B)$ is a stair, of type (A, D), that might be a simple stair; on the other hand r_2 is no stair. Some simple stairs can be of the form $A(x_1, \dots, x_m) \xrightarrow{w} A'(x_{i_1}, \dots, x_{i_m})$, where $\{i_1, \dots, i_m\} \subseteq \{1, \dots, m\}$; we might even have $A = A'$ but in this case $(x_1, \dots, x_m) \neq (x_{i_1}, \dots, x_{i_m})$ since $A(x_1, \dots, x_m) \xrightarrow{w} A(x_1, \dots, x_m)$ is no direct stair.)

It is easy to observe that any direct stair w is a sequence of compatible simple stairs, i.e., $w = w_1w_2 \dots w_n$ where w_i is a simple stair of type (A_{i-1}, A_i), for each $i \in [1, n]$; we thus have $A_0(x_1, \dots, x_m) \xrightarrow{w_1} A_1(x_1, \dots, x_m)\sigma_1 \xrightarrow{w_2} A_2(x_1, \dots, x_m)\sigma_2\sigma_1 \xrightarrow{w_3} \cdots$ for the respective (not necessarily guarded) substitutions σ_i.

A *stair* w where $A(x_1, \dots, x_m) \xrightarrow{w} A(E_1, \dots, E_m)$ is *eligible* (for "pumping") if the set of "root-sticks" $R = \{x_i \mid E_j = x_i \text{ for some } j \in [1, m]\}$ is equal to $\{x_i \mid i \in [1, m], E_i = x_i\}$.

(E.g., the stair $A(x_1, x_2, x_3, x_4) \xrightarrow{w} A(x_1, B(x_2, x_2, x_4, x_1), x_3, x_3)$ is eligible, with $R = \{x_1, x_3\}$. The stair $A(x_1, x_2, x_3, x_4) \xrightarrow{v} A(x_2, B(x_2, x_2, x_4, x_1), x_3, x_3)$ is not eligible but the respective "double" stair $A(x_1, x_2, x_3, x_4) \xrightarrow{vv} A(B(x_2, x_2, x_4, x_1), B(\dots), x_3, x_3)$ is eligible. In particular, if in $A(x_1, \dots, x_m) \xrightarrow{w} A(E_1, \dots, E_m)$ all E_j have nonterminal-roots, then w is eligible, with $R = \emptyset$.)

An important fact is that for any eligible stair w, where $A(x_1, \dots, x_m) \xrightarrow{w} A(x_1, \dots, x_m)\sigma$, we can define the terms $G_{(w,z)}$ for all $z \in \mathbb{N} \cup \{\omega\}$ by putting

$$A(x_1, \dots, x_m) \xrightarrow{w^z} A(x_1, \dots, x_m)\sigma^z = G_{(w,z)}$$

which is well defined also for $z = \omega$. (Though σ might be not guarded, we have that $x_j\sigma = x_i$ for $i \neq j$ implies $x_i\sigma = x_i$ due to the eligibility and thus $x_j\sigma^\omega = x_i$.)

Candidates for simple witnesses of bisim-infiniteness. Given a grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$ and a term E_0, by a *candidate for a simple witness* (of bisim-infiniteness of E_0), or by a *candidate* for short, we mean a pair (u, w) where $u \in \mathcal{R}^*$, $w \in \mathcal{R}^+$, $E_0 \xrightarrow{uw}$, and w is an eligible stair, of the form $A(x_1, \dots, x_m) \xrightarrow{w} A(x_1, \dots, x_m)\sigma$; we thus have

$$E_0 \xrightarrow{u} A(x_1, \dots, x_m)\sigma_0 \xrightarrow{w} G_{(w,1)}\sigma_0 \xrightarrow{w} G_{(w,2)}\sigma_0 \xrightarrow{w} G_{(w,3)}\sigma_0 \xrightarrow{w} \cdots$$

for the respective substitution σ_0. We have $G_{(w,j)} = A(x_1, \dots, x_m)\sigma^j$, and we denote the term $G_{(w,\omega)} = A(x_1, \dots, x_m)\sigma^\omega$ also by $\textsc{Lim}$.

We now formalize the simple observation that the terms $G_{(w,k)}\sigma_0$ with increasing $k \in \mathbb{N}$ "approach" the term $\textsc{Lim}\,\sigma_0$ syntactically, and thus also semantically.

Top-tails presentations. Given a term G and (depth) $d \in \mathbb{N}$, let $\textsc{nod}_1, \textsc{nod}_2, \dots, \textsc{nod}_n$ ($n \geq 0$) be the ordered nodes of (the syntactic tree of) G in depth d (if there are some); let $F_1, F_2, \dots, F_n$ be the (occurrences of) subterms of G rooted in $\textsc{nod}_1, \textsc{nod}_2, \dots, \textsc{nod}_n$.

- By $\textsc{Top}_d(G)$ we denote the term that coincides with G up to depth $d-1$ while its ordered nodes in depth d are (leaves) labelled with $x_1, x_2, \dots, x_n$, respectively; here we assume that no x_i, $i \in [1, n]$, occurs in G in the depths less than d.
- By $\textsc{tails}_d(G)$ we mean the substitution defined by $x_i\, \textsc{tails}_d(G) = F_i$ for $i \in [1, n]$.

We have $G = (\textsc{Top}_d(G))\sigma$ where $\sigma = \textsc{tails}_d(G)$. In particular, if $G = A(F_1, \dots, F_m)$, then $G = (\textsc{Top}_1(G))\textsc{tails}_1(G) = A(x_1, \dots, x_m)\sigma$ where $x_i\sigma = F_i$.

If some x_i occur in G in the depths less than d, then we define $\textsc{Top}_d(G)$, $\textsc{tails}_d(G)$ by introducing the variables (in the role of place-holders) other than such x_i. In the following example we highlight this by using "another set of variables" y_i.

For $G = A(B(x_3,x_2,x_2),x_2,C(x_1,B(x_2,x_3,x_1),x_1))$ we have

- $\textsc{Top}_2(G) = A(B(y_1,y_2,y_3),x_2,C(y_4,y_5,y_6))$, and
- $\textsc{tails}_2(G) = \{(y_1,x_3),(y_2,x_2),(y_3,x_2),(y_4,x_1),(y_5,B(x_2,x_3,x_1)),(y_6,x_1)\}$.

The next proposition refers to a candidate $E_0 \xrightarrow{u} A(x_1,\dots,x_m)\sigma_0 \xrightarrow{w} A(x_1,\dots,x_m)\sigma\sigma_0$ where we denote $x_i\sigma$ by E_i for $i \in [1,m]$.

▶ **Proposition 7.** *The following conditions hold for all $k \in \mathbb{N}$ and $i \in [1,m]$.*

1. $\textsc{Top}_k(E_i\sigma^k\sigma_0) = \textsc{Top}_k(E_i\sigma^\omega\sigma_0)$, *hence* $\textsc{Top}_k(G_{(w,k)}\sigma_0) = \textsc{Top}_k(\textsc{Lim}\,\sigma_0)$.
2. $E_i\sigma^k\sigma_0 \sim_k E_i\sigma^\omega\sigma_0$ *and thus* $\textsc{EqLv}(G_{(w,k)}\sigma_0, \textsc{Lim}\,\sigma_0) \geq k$.

Checking if a candidate is a simple witness. A candidate $E_0 \xrightarrow{u} A(x_1,\dots,x_m)\sigma_0 \xrightarrow{w}$ is a *simple witness* (of bisim-infiniteness of E_0) if $G_{(w,k)}\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$ for infinitely many $k \in \mathbb{N}$. Since $\textsc{EqLv}(G_{(w,k)}\sigma_0, \textsc{Lim}\,\sigma_0) \geq k$ (Prop. 7(2)), we then have

$$e < \textsc{EqLv}(G_{(w,e+1)}\,\sigma_0, \textsc{Lim}\,\sigma_0) < \omega \text{ for infinitely many } e \in \mathbb{N},$$

and by Prop. 4 we derive that E_0 is bisim-infinite if it has a simple witness.

The existence of an algorithm checking if a candidate is a simple witness follows from the next lemma, if we recall the fundamental fact captured by Lemma 2.

▶ **Lemma 8.** *Given a candidate $E_0 \xrightarrow{u} A(x_1,\dots,x_m)\sigma_0 \xrightarrow{w} A(E_1,\dots,E_m)\sigma_0$, there is a computable number e such that one of the following conditions holds:*

1. $G_{(w,e)}\sigma_0 \sim \textsc{Lim}\,\sigma_0$, *in which case* $G_{(w,k)}\sigma_0 \sim \textsc{Lim}\,\sigma_0$ *for all* $k \geq e$, *or*
2. $G_{(w,e)}\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$, *in which case* $G_{(w,k)}\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$ *for all* $k \geq e$.

(The candidate is a simple witness of bisim-infiniteness of E_0 in the case 2.)

Proof. We restrict our attention to those E_i that "almost always matter" when we apply σ, where $x_i\sigma = E_i$, to the term $A(E_1,\dots,E_m)\sigma^\ell$, for growing $\ell \in \mathbb{N}$. The sets

$$\mathcal{V}_\ell = \{x_i \mid x_i \text{ occurs in } A(E_1,\dots,E_m)\sigma^\ell\}$$

satisfy $\mathcal{V}_{\ell+1} = \{x_j \mid x_j \text{ occurs in } E_i \text{ for some } i \text{ such that } x_i \in \mathcal{V}_\ell\}$, and thus $\{x_1,\dots,x_m\} \supseteq \mathcal{V}_0 \supseteq \mathcal{V}_1 \supseteq \mathcal{V}_2 \supseteq \cdots$. Let ℓ_0 be the smallest such that $\mathcal{V}_{\ell_0} = \mathcal{V}_{\ell_0+1}$ $(= \mathcal{V}_{\ell_0+2} = \cdots)$; hence $\ell_0 \leq m$.

We can compute a number d (the "radius" for the below defined region that will be used in the application of Prop. 5) so that within d transition-steps we can reach any variable $x_i \in \mathcal{V}_{\ell_0}$ from both $A(E_1,\dots,E_m)\sigma^{\ell_0}$ and $A(E_1,\dots,E_m)\sigma^{\ell_0+1}$.

Suppose now that $A(E_1,\dots,E_m)\sigma^k\sigma_0 \not\sim A(E_1,\dots,E_m)\sigma^\omega\sigma_0$ (i.e., $G_{(w,k+1)}\,\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$) and $k > \ell_0$; then by compositionality (Prop. 6) we deduce that $x_i\sigma^{k-\ell_0}\sigma_0 \not\sim x_i\sigma^\omega\sigma_0$ for some $x_i \in \mathcal{V}_{\ell_0}$, and also $x_{i'}\sigma^{k-\ell_0-1}\sigma_0 \not\sim x_{i'}\sigma^\omega\sigma_0$ for some $x_{i'} \in \mathcal{V}_{\ell_0}$. In other words,

$$E_i\sigma^{k-\ell_0-1}\sigma_0 \not\sim E_i\sigma^\omega\sigma_0, \text{ and } E_{i'}\sigma^{k-\ell_0-2}\sigma_0 \not\sim E_{i'}\sigma^\omega\sigma_0 \text{ for some } i,i' \text{ such that } x_i, x_{i'} \in \mathcal{V}_{\ell_0}.$$

Since x_i occurs in $A(E_1,\dots,E_m)\sigma^{\ell_0}$ and $x_{i'}$ occurs in $A(E_1,\dots,E_m)\sigma^{\ell_0+1}$, we have that both $E_i\sigma^{k-\ell_0-1}\sigma_0$ and $E_{i'}\sigma^{k-\ell_0-2}\sigma_0$ are in $\textsc{Region}(G_{(w,k+1)}\,\sigma_0, d)$, for the above defined "radius" d.

We would like to deduce that also $G_{(w,k+2)}\,\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$, using the fact that $E_i\sigma^{k-\ell_0-1}\sigma_0$ is in $\textsc{Region}(G_{(w,k+2)}\sigma_0, d)$ (though maybe farther than previously but still within the bound d). But this deduction is unsubstantiated in general.

Hence we recall Prop. 5, and compute the maximum $\textsc{Max}_{\text{TEL}}$ in the set $\text{TEL}(\textsc{Lim}\,\sigma_0, d, \mathcal{T})$ where the test set is $\mathcal{T} = \{E_i\sigma^\omega\sigma_0 \mid x_i \in \mathcal{V}_{\ell_0}\}$. (We can compute this set by Lemma 2.) Let

$$e = \textsc{Max}_{\text{TEL}} + m + 3. \text{ Hence } e \geq \textsc{Max}_{\text{TEL}} + \ell_0 + 3.$$

If we return to the above analysis of the case $A(E_1,\dots,E_m)\sigma^k\sigma_0 \not\sim A(E_1,\dots,E_m)\sigma^\omega\sigma_0$, now assuming $k \geq e$, then the fact that $E_i\sigma^{k-\ell_0-1}\sigma_0$ (for which $E_i\sigma^{k-\ell_0-1}\sigma_0 \not\sim E_i\sigma^\omega\sigma_0$) is also present in $\textsc{Region}(G_{(w,k+2)}\,\sigma_0, d)$ indeed testifies that $G_{(w,k+2)}\,\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$.

(Since $E_i\sigma^{k-\ell_0-1}\sigma_0 \sim_{k-\ell_0-1} E_i\sigma^\omega\sigma_0$, and $k-\ell_0-1 \geq e-\ell_0-1 > \textsc{Max}_{\mathrm{TEL}}$, the value $\textsc{EqLv}(E_i\sigma^{k-\ell_0-1}\sigma_0, E_i\sigma^\omega\sigma_0)$ is finite but bigger than $\textsc{Max}_{\mathrm{TEL}}$; we thus must have $G_{(w,k+2)}\,\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$ by Prop. 5.)

Then $G_{(w,k+2)}\,\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$ similarly entails that $G_{(w,k+3)}\,\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$, etc.

To finish a demonstration that the above e proves the claim, we note that $G_{(w,e)}\,\sigma_0 \sim \textsc{Lim}\,\sigma_0$ entails that we have $E_i\sigma^{e-\ell_0-1}\sigma_0 \sim E_i\sigma^\omega\sigma_0$ for all i (where $x_i \in \mathcal{V}_{\ell_0}$), since otherwise the sets $\mathrm{TEL}(G_{(w,e)}\sigma_0, d, \mathcal{T})$ and $\mathrm{TEL}(\textsc{Lim}\,\sigma_0, d, \mathcal{T})$ would differ. Then $G_{(w,k)}\sigma_0 \sim \textsc{Lim}\,\sigma_0$ for all $k \geq e$ by compositionality. ◀

3.2.3 Each bisim-infinite term has a simple witness

Once we show the next lemma, the proof of Theorem 1 will be finished.

▶ **Lemma 9.** *For any grammar $\mathcal{G}$ and any bisim-infinite E_0 there is a simple witness satisfying the condition 2 in Lemma 8 ($G_{(w,e)}\sigma_0 \not\sim \textsc{Lim}\,\sigma_0$ for the respective computable e).*

We prove the lemma in the rest of this section. We assume a given grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$ and a term E_0 that is bisim-infinite.

An infinite simple-stair sequence witnessing bisim-infiniteness. Let us fix an infinite path $E_0 \xrightarrow{r_1} E_1 \xrightarrow{r_2} E_2 \xrightarrow{r_3} \cdots$ in $\mathcal{L}^{\mathrm{R}}_{\mathcal{G}}$ such that $E_i \not\sim E_j$ (in $\mathcal{L}^{\mathrm{A}}_{\mathcal{G}}$) for all $i \neq j$ (recall Prop. 3); this entails that there is *no repeat*, i.e., we have $E_i \neq E_j$ for all $i \neq j$. Hence there must be the least $i_0 \in \mathbb{N}$ such that $r_{i_0+1}r_{i_0+2}\dots r_{i_0+\ell}$ is a stair for each $\ell \in \mathbb{N}$. (This obviously holds even if E_0 is an infinite term, since it has only finitely many subterms due to its regularity.) Moreover, given i_j, there must be the least i_{j+1} such that $i_j < i_{j+1}$ and $r_{i_{j+1}+1}r_{i_{j+1}+2}\dots r_{i_{j+1}+\ell}$ is a stair for each $\ell \in \mathbb{N}$.

For each $j \in \mathbb{N}$ we put $H_j = E_{i_j}$, and we present the suffix of the above path starting with E_{i_0} as

$$H_0 \xrightarrow{w_1} H_1 \xrightarrow{w_2} H_2 \xrightarrow{w_3} \cdots, \text{ denoting } H_j = A_j(x_1,\dots,x_m)\sigma_j\sigma_{j-1}\cdots\sigma_0 \tag{2}$$

where $A_j(x_1,\dots,x_m) \xrightarrow{w_{j+1}} A_{j+1}(x_1,\dots,x_m)\sigma_{j+1}$. We also write $H_j = A_j(x_1,\dots,x_m)\sigma'_j$.

The words w_i are stairs; we can even assume that w_i are direct stairs (i.e., we replace them with the respective direct stairs if they are not direct stairs) while keeping the property that $H_i \not\sim H_j$ for all $i \neq j$. We thus assume that w_i are *direct stairs*, which in our case obviously implies that they are *simple stairs*.

A specific "keep-and-drown task" extracted from the bisim-infinite stair path. Prop. 10 captures a first step in demonstrating the existence of a simple witness induced by the path $H_0 \xrightarrow{w_1} H_1 \xrightarrow{w_2} H_2 \xrightarrow{w_3} \cdots$ (2), related to a fixed grammar $\mathcal{G} = (\mathcal{N}, \Sigma, \mathcal{R})$.

A *task* T is a tuple $(A, \textsc{Drown}, \textsc{keep})$ where $A \in \mathcal{N}$, $\textsc{Drown} \subseteq \{x_1,\dots,x_m\}$, and $\textsc{keep} \in [1,m]$, $x_{\textsc{keep}} \in \textsc{Drown}$. We put $\textsc{NotCare} = \{x_1,\dots,x_m\} \smallsetminus \textsc{Drown}$.

For a term G, by $G \models (\textsc{Drown}, z)$, for $z \in \mathbb{N} \cup \{\omega\}$, we denote that the depth of each occurrence of $x_i \in \textsc{Drown}$ in G (if there is any) is at least z, which means that x_i does not occur in G when $z = \omega$. By $G \models \textsc{keep}$ we denote that $x_{\textsc{keep}}$ occurs in G, and $G \models (\textsc{Drown}, z) \wedge \textsc{keep}$ denotes that we have $G \models (\textsc{Drown}, z)$ and $G \models \textsc{keep}$.

A *task* $\mathrm{T} = (A, \textsc{Drown}, \textsc{keep})$ is *satisfied for* $k \in \mathbb{N}$, which is denoted by $\models (\mathrm{T}, k)$, if there is $w \in \mathcal{R}^*$ and G such that $A(x_1, \dots, x_m) \xrightarrow{w} G$ and $G \models (\textsc{Drown}, k) \wedge \textsc{keep}$. By $\models (\mathrm{T}, \omega)$ we denote that we have $\models (\mathrm{T}, k)$ for all $k \in \mathbb{N}$.

A technical proof of the next proposition is given in the full arxiv-version.

▶ **Proposition 10.** *There is a task* $\mathrm{T} = (A, \textsc{Drown}, \textsc{keep})$ *and an infinite subsequence* $\textsc{Seq}$ *of the sequence* $0, 1, 2, \dots$ *such that* $\models (\mathrm{T}, \omega)$ *and the following conditions hold in the path (2), where we use the notation* $H_j = A_j(x_1, \dots, x_m)\sigma'_j$*:*
1. $A_j = A$ *for all* $j \in \textsc{Seq}$*;*
2. *for each* $x_i \in \textsc{NotCare}$ *we have* $x_i\sigma'_{j_1} \sim x_i\sigma'_{j_2}$ *for all* $j_1, j_2 \in \textsc{Seq}$*;*
3. *for each* $x_i \in \textsc{Drown}$ *we have* $x_i\sigma'_{j_1} \not\sim x_i\sigma'_{j_2}$ *for any* $j_1 \neq j_2$ *where* $j_1, j_2 \in \textsc{Seq}$.

Witness schemes. For our fixed path $H_0 \xrightarrow{w_1} H_1 \xrightarrow{w_2} H_2 \xrightarrow{w_3} \cdots$ (2) we aim to show that there are a stair u and an eligible stair w such that $H_0 \xrightarrow{u}\xrightarrow{w}$ is a simple witness (of bisim-infiniteness of H_0 and thus also of E_0). We will also have that both u and w are sequences of simple stairs, hence $uw = w_1 w_2 \cdots w_\ell$ where $A_0(x_1, \dots, x_m) \xrightarrow{w_1}\xrightarrow{w_2} \cdots \xrightarrow{w_\ell}$, each w_i is a simple stair, and the sequence $w = w_j w_{j+1} \dots w_\ell$ is *marked as a pumping stair*. It is useful to make the following generalization (of simple witnesses).

A *stair-scheme*, or just a *scheme* for short, is a sequence $W = w_1, w_2, \dots, w_\ell$ of (compatible) simple stairs, where $A(x_1, \dots, x_m) \xrightarrow{w_1}\xrightarrow{w_2} \cdots \xrightarrow{w_\ell}$ for A determined by the first grammar-rule r_1 in w_1, and where any segment $w_i w_{i+1} \dots w_j$ that is an eligible stair might be marked as a *pumping stair*; the pumping stairs can be "nested", one can be contained in another, but no pumping stair can start or end inside another pumping stair.

We use the notation of regular expressions with concatenation and iteration (star) to denote such schemes; an example is $u_1((v_1)^* u_2 (v_2)^*)^* u_3 (v_3)^* u_4 u_5 ((v_5)^* u_6)^*$ (where we have six pumping stairs, namely v_1, v_2, $v_1 u_2 v_2$, v_3, v_5, and $v_5 u_6$).

For a scheme W (like above), by $\textsc{Pump}(W, z)$, where $z \in \mathbb{N} \cup \{\omega\}$, we denote the sequence arising from $w_1 w_2 \dots w_\ell$ by repeating each pumping stair z times. (In our example, $\textsc{Pump}(W, z)$ is $u_1((v_1)^z u_2 (v_2)^z)^z u_3 (v_3)^z u_4 u_5 ((v_5)^z u_6)^z$.) In the case $z = \omega$ we get infinite "words" whose ordinal lengths can be bigger than ω, but since all pumping stairs are eligible, we can soundly define the terms $G_{(W,z)}$ by

$$A(x_1, \dots, x_m) \xrightarrow{\textsc{Pump}(W,z)} G_{(W,z)}; \text{ we also put } \textsc{Lim}_W = G_{(W,\omega)}.$$

We say that a scheme W, where A is the left-hand side nonterminal of the first rule in W, is a ("non-simple") *witness* (of bisim-infiniteness) *for* $A(x_1, \dots, x_m)\sigma$ if $G_{(W,k)}\sigma \not\sim \textsc{Lim}_W\sigma$ for infinitely many $k \in \mathbb{N}$.

It is not difficult to generalize Lemma 8 for schemes (viewed as candidates for witnesses), and to derive the next proposition (as is shown in the full version).

▶ **Proposition 11.** *A term* E_0 *has a simple witness (of bisim-infiniteness) iff there is a term* $H = A(x_1, \dots, x_m)\sigma$ *reachable from* E_0 *for which there is (a scheme* W *that is) a witness.*

It suffices that $\models (\mathrm{T}, \omega)$ can be demonstrated by a scheme. We first show that Lemma 12 suffices for finishing the proof of Theorem 1, and then we sketch a proof idea for the lemma.

▶ **Lemma 12.** *For any task* $\mathrm{T} = (A, \textsc{Drown}, \textsc{keep})$ *where* $\models (\mathrm{T}, \omega)$ *there is a scheme* W*, starting from* $A(x_1, \dots, x_m)$*, such that* $G_{(W,k)} \models \textsc{keep}$ *for all* $k \in \mathbb{N}$ *and* $G_{(W,\omega)} \models (\textsc{Drown}, \omega)$*; for such* W *we have* $G_{(W,k)} \models (\textsc{Drown}, k) \wedge \textsc{keep}$ *for all* $k \in \mathbb{N}$.

We consider a task $\mathrm{T} = (A, \textsc{Drown}, \textsc{keep})$, where $\models (\mathrm{T}, \omega)$, that can be extracted from the path $H_0 \xrightarrow{w_1} H_1 \xrightarrow{w_2} H_2 \xrightarrow{w_3} \cdots$ (2) by Prop. 10; we also recall the respective sequence $\textsc{Seq}$ and the notation $H_j = A(x_1, \dots, x_m)\sigma'_j$. Let W be a scheme guaranteed by Lemma 12 for T; we recall the notation $\textsc{Lim}_W = G_{(W,\omega)}$. For all $j \in \textsc{Seq}$ the terms $\textsc{Lim}_W\, \sigma'_j$ are from the same bisim-class (by 2 in Prop. 10); let $\textsc{Lim}$ be a term representing this class.

For the sake of contradiction we now suppose that W is not a witness for any $H_j = A(x_1, \dots, x_m)\sigma'_j$. Then there is some $e \in \mathbb{N}$ (determined by $\textsc{Lim}$) such that $G_{(W,e)}\sigma'_j \sim \textsc{Lim}$ for all $j \in \textsc{Seq}$ (due to the mentioned generalization of Lemma 8). Since there is $d \in \mathbb{N}$ such that $x_{\textsc{keep}}\sigma'_j \in \textsc{Region}(G_{(W,e)}\sigma'_j, d)$ for all $j \in \textsc{Seq}$, all bisim-classes $[x_{\textsc{keep}}\sigma'_j]_\sim$ for $j \in \textsc{Seq}$ must be in $\textsc{Region}([\textsc{Lim}]_\sim, d)$ in the quotient-LTS $(\mathcal{L}^{\mathsf{A}}_{\mathcal{G}})_\sim$ (which follows from the fact $G_{(W,e)}\sigma'_j \sim \textsc{Lim}$). There are thus only finitely many bisim-classes $[x_{\textsc{keep}}\sigma'_j]_\sim$ where $j \in \textsc{Seq}$, which contradicts with the condition 3 of Prop. 10 that $x_{\textsc{keep}}\sigma'_{j_1} \not\sim x_{\textsc{keep}}\sigma'_{j_2}$ for any $j_1 \neq j_2$ in $\textsc{Seq}$ (recall that $x_{\textsc{keep}} \in \textsc{Drown}$).

Hence W is a witness for some $H_j = A(x_1, \dots, x_m)\sigma'_j$; by Prop. 11 this proves Lemma 9 (and thus Theorem 1).

The fact $\models (\mathrm{T}, \omega)$ can be demonstrated by a scheme. We now sketch a proof idea for Lemma 12. If $\models (\mathrm{T}, \omega)$, where $\mathrm{T} = (A, \textsc{Drown}, \textsc{keep})$, then there is a collection of paths $A(x_1, \dots, x_m) \xrightarrow{w_k} G_k$ where $G_k \models (\textsc{Drown}, k) \wedge \textsc{keep}$, for all $k \in \mathbb{N}$. We can choose shortest possible words w_k; in fact, they are sequences of simple stairs.

Each path $A(x_1, \dots, x_m) \xrightarrow{w_k} G_k$ must be progressing to its goal, stepwise "drowning" the (occurrences of) variables $x_i \in \textsc{Drown}$, while keeping at least one occurrence of $x_{\textsc{keep}}$. For a term F we can define its *drown-quality* as the function $\mathrm{DQ}(F) : \textsc{Drown} \to \mathbb{N} \cup \{\omega\}$ where $\mathrm{DQ}(F)(x_i)$ is the smallest (shallowest) depth of an occurrence of x_i in F (where $\mathrm{DQ}(F)(x_i) = \omega$ means that x_i does not occur in F). The *keep-quality* $\mathrm{KQ}(F)$ is one bit (1 ir 0) that captures the fact if $x_{\textsc{keep}}$ occurs in F. For each term $H = B(F_1, \dots, F_m)$ on a path $A(x_1, \dots, x_m) \xrightarrow{w_k} G_k$ we define its *level-quality* as $\mathrm{LQ}(H) = (B, \mathrm{DQ}(F_1), \dots, \mathrm{DQ}(F_m), \mathrm{KQ}(F_1), \dots, \mathrm{KQ}(F_m))$.

By standard facts, in particular Dickson's Lemma and König's Lemma, in any sufficiently long w_k there is an eligible stair that keeps or increases the level-quality in each component (where we put $B \leq B'$ if $B = B$). This does not solve the problem completely, due to the possible long segments with root-sticking depth-1 subterms. This subtle point is handled in the full arxiv-version.

4 Additional Remarks

The mentioned deterministic case studied by Valiant [13] could be roughly explained as follows: for a deterministic grammar, if an eligible stair is reachable from E_0 where the start and the end of the stair are non-equivalent, then E_0 is bisim-infinite. Hence by compositionality a bound on the size of the potential equivalent finite system can be derived, and thus decidability of the full equivalence is not needed here.

In the case equivalent to *normed* pushdown processes, the regularity problem essentially coincides with the boundedness problem, and is thus much simpler. (See, e.g., [11] for a further discussion.)

Acknowledgements. This work has been supported by the Grant Agency of the Czech Rep., project GAČR:15-13784S. I also thank Stefan Göller for drawing my attention to the decidability question for regularity of pushdown processes, and for discussions about some related works (like [13]).

References

1 Michael Benedikt, Stefan Göller, Stefan Kiefer, and Andrzej S. Murawski. Bisimilarity of pushdown automata is nonelementary. In *Proc. LICS 2013*, pages 488–498. IEEE Computer Society, 2013.

2 Christopher H. Broadbent and Stefan Göller. On bisimilarity of higher-order pushdown automata: Undecidability at order two. In *FSTTCS 2012*, volume 18 of *LIPIcs*, pages 160–172. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

3 Bruno Courcelle. Recursive applicative program schemes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol. B*, pages 459–492. Elsevier, MIT Press, 1990.

4 Petr Jančar. Bisimulation equivalence of first-order grammars. In *Proc. ICALP'14 (II)*, volume 8573 of *LNCS*, pages 232–243. Springer, 2014.

5 Petr Jančar. Equivalences of pushdown systems are hard. In *Proc. FOSSACS 2014*, volume 8412 of *LNCS*, pages 1–28. Springer, 2014.

6 Petr Jančar and Jiri Srba. Undecidability of bisimilarity by defender's forcing. *J. ACM*, 55(1), 2008. doi:10.1145/1326554.1326559.

7 Antonín Kučera and Richard Mayr. On the complexity of checking semantic equivalences between pushdown processes and finite-state processes. *Inf. Comput.*, 208(7):772–796, 2010.

8 Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3, 2016.

9 Géraud Sénizergues. L(A)=L(B)? Decidability results from complete formal systems. *Theoretical Computer Science*, 251(1–2):1–166, 2001.

10 Géraud Sénizergues. The bisimulation problem for equational graphs of finite out-degree. *SIAM J.Comput.*, 34(5):1025–1106, 2005.

11 Jiri Srba. Roadmap of infinite results. In *Current Trends In Theoretical Computer Science, The Challenge of the New Century*, volume 2, pages 337–350. World Scientific Publishing Co., 2004. Updated version at http://users-cs.au.dk/srba/roadmap/.

12 Colin Stirling. Deciding DPDA equivalence is primitive recursive. In *Proc. ICALP'02*, volume 2380 of *LNCS*, pages 821–832. Springer, 2002.

13 Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *J. ACM*, 22(1):1–10, 1975. doi:10.1145/321864.321865.

Minimal Phylogenetic Supertrees and Local Consensus Trees

Jesper Jansson*[1] and Wing-Kin Sung[2]

1 **Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Kyoto, Japan**
`jj@kuicr.kyoto-u.ac.jp`

2 **School of Computing, National University of Singapore, Singapore; and Genome Institute of Singapore, Singapore**
`ksung@comp.nus.edu.sg`

Abstract

The problem of constructing a *minimally resolved phylogenetic supertree* (i.e., having the smallest possible number of internal nodes) that contains all of the rooted triplets from a consistent set $\mathcal{R}$ is known to be NP-hard. In this paper, we prove that constructing a phylogenetic tree consistent with $\mathcal{R}$ that contains the minimum number of additional rooted triplets is also NP-hard, and develop exact, exponential-time algorithms for both problems. The new algorithms are applied to construct two variants of the *local consensus tree*; for any set $\mathcal{S}$ of phylogenetic trees over some leaf label set L, this gives a minimal phylogenetic tree over L that contains every rooted triplet present in all trees in $\mathcal{S}$, where "minimal" means either having the smallest possible number of internal nodes or the smallest possible number of rooted triplets. The second variant generalizes the *RV-II tree*, introduced by Kannan, Warnow, and Yooseph in 1998.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, J.3 Life and Medical Sciences

Keywords and phrases phylogenetic tree, rooted triplet, local consensus, minimal supertree, computational complexity, bioinformatics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.53

1 Introduction

Phylogenetic trees are used to describe evolutionary relationships between species [11]. The *supertree approach* is a relatively new divide-and-conquer-based technique for reconstructing phylogenetic trees that may be useful when dealing with very big datasets [4]. The general idea behind it is to first infer a set of highly accurate trees for overlapping subsets of the species (e.g., using a computationally expensive method such as maximum likelihood [9, 11]) and then combine all the trees into one tree according to some well-defined rule. An example of a famous phylogenetic supertree for more than 4500 species can be found in [5]; see also [4, 15] for references to many other supertrees in the biological literature. One class of supertree methods consists of the BUILD algorithm [2] and its various extensions [10, 13, 14, 18, 19, 20, 21, 24] for combining a set of *rooted triplets* (binary phylogenetic trees with three leaves each), e.g., inferred by the method in [9].

A *consensus tree* [1, 7, 17] can be regarded as the special case of a phylogenetic supertree where all the trees that are to be combined have the same leaf label set. Such inputs

* Funded by The Hakubi Project and KAKENHI grant number 26330014.

arise when a collection of alternative datasets, each covering all the species, is available, or when applying bootstrapping or different tree reconstruction algorithms to the same basic dataset [11]. A consensus tree can also measure the similarity between two identically leaf-labeled trees or identify parts of trees that are similar. Many different types of consensus trees, whose formal definitions of how to handle conflicts differ, have been proposed in the last 45 years. See the surveys in [7], Chapter 30 in [11], and Chapter 8.4 in [23] for more details about different consensus trees and their advantages and disadvantages.

In situations where more than one phylogenetic tree can explain some given experimental data equally well, it is natural to select a "minimal" tree that supports the data while making as few extra statements about the evolutionary history as possible. A *minimally resolved phylogenetic supertree* [16] is a supertree that is consistent with all of the input and that has the minimum number of internal nodes. By minimizing the number of internal nodes, the risk of creating false groupings called "spurious novel clades" [4] is reduced. Furthermore, such a tree gives a simpler overview of the data than a tree with many internal nodes and can in general be stored in less memory. Another way to define "minimal" above, giving what we call a *minimally rooted-triplet-inducing phylogenetic supertree*, instead requires that the supertree contains the minimum number of rooted triplets. This interpretation of minimal was previously considered in the definition of the *RV-II local consensus tree* in [17].

The goal of this paper is further develop the mathematical framework of minimal phylogenetic supertrees and to design new supertree algorithms that can also be applied to the construction of consensus trees.

1.1 Problem Definitions

A *rooted phylogenetic tree* is a rooted, unordered, leaf-labeled tree in which all leaf labels are different and every internal node has at least two children. For example, T_1 and T_2 in Figure 1 are two rooted phylogenetic trees. In this paper, rooted phylogenetic trees are referred to as "trees" and every leaf in a tree is identified with its unique label.

Let T be a tree. The set of all nodes in T, the set of internal nodes in T, and the set of leaves in T are denoted by $V(T)$, $I(T)$, and $\Lambda(T)$, respectively. For any $u, v \in V(T)$, if u is a descendant of v and $u \neq v$ then we write $u \prec v$. $lca(u, v)$ means the lowest common ancestor of u and v.

A *rooted triplet* is a binary tree with exactly three leaves. We use the notation $xy|z$ to refer to the rooted triplet with leaf label set $\{x, y, z\}$ such that $lca(x, y) \prec lca(x, z) = lca(y, z)$. Let T be a tree. For any $x, y, z \in \Lambda(T)$, if $lca(x, y) \prec lca(x, z) = lca(y, z)$ holds in T then the rooted triplet $xy|z$ and T are said to be *consistent* with each other. For example, $ab|c$ is consistent with T_1 but not with T_2 in Figure 1. Observe that for any $\{x, y, z\} \subseteq \Lambda(T)$, exactly zero or one of the three rooted triplets $xy|z$, $xz|y$, and $yz|x$ is consistent with T. The set of all rooted triplets that are consistent with T is denoted by $r(T)$. For any set $\mathcal{R}$ of rooted triplets, if $\mathcal{R} \subseteq r(T)$ then $\mathcal{R}$ and T are *consistent* with each other. Finally, a set $\mathcal{R}$ of rooted triplets is *consistent* if there exists a tree that is consistent with $\mathcal{R}$.

Next, we give the definitions of the *minimally resolved phylogenetic supertree consistent with rooted triplets problem* (MinRS) (studied in [16]) and the *minimally rooted-triplet-inducing phylogenetic supertree consistent with rooted triplets problem* (MinIS). In both problems, the input is a consistent set $\mathcal{R}$ of rooted triplets[1], and the output is a tree T

[1] This paper assumes without loss of generality that the input $\mathcal{R}$ to MinRS/MinIS is consistent. The reason is that given an arbitrary $\mathcal{R}$, one can check whether $\mathcal{R}$ is consistent or not in polynomial time using the BUILD algorithm [2] described below.

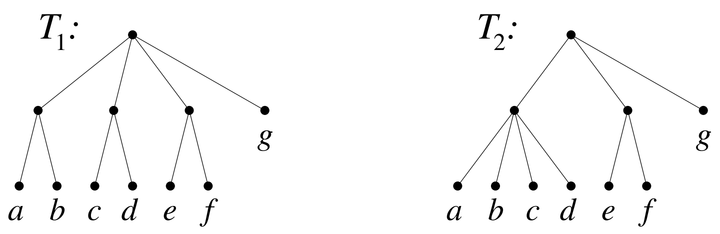


Figure 1 An example. Let $\mathcal{S} = \{T_1, T_2\}$ as above with $\Lambda(T_1) = \Lambda(T_2) = \{a, b, c, d, e, f, g\}$. Then $r(T_1) \cap r(T_2) = \{ab|e,\ ab|f,\ ab|g,\ cd|e,\ cd|f,\ cd|g,\ ef|a,\ ef|b,\ ef|c,\ ef|d,\ ef|g\}$ and T_2 is an optimal solution to MinRLC. On the other hand, $|r(T_1)| = 15$ while $|r(T_2)| = 23$, so T_2 cannot be an optimal solution to MinILC.

satisfying $\Lambda(T) = \bigcup_{t\in\mathcal{R}} \Lambda(t)$ and $\mathcal{R} \subseteq r(T)$. The objectives are to minimize the value of $|I(T)|$ (for MinRS) and to minimize the value of $|r(T)|$ (for MinIS), respectively.

In the *minimally resolved local consensus tree problem* (MinRLC) and the *minimally rooted-triplet-inducing local consensus tree problem* (MinILC) (introduced in [17] for the special case $k = 2$), the input is a set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L$ for some leaf label set L, and the output is a tree T satisfying $\Lambda(T) = L$ and $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$. The objectives in MinRLC and MinILC are, respectively, to minimize the value of $|I(T)|$ and to minimize the value of $|r(T)|$.

Note that MinRLC and MinILC admit trivial polynomial-time reductions to MinRS and MinIS, respectively, by letting $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$.

See Figure 1 for a simple example showing that MinRLC and MinILC are indeed different problems, and consequently, that MinRS is different from MinIS. From here on, we will use *Newick notation*[2] to describe trees compactly. E.g., in Figure 1, we have $T_1 = ((a, b),\ (c, d),\ (e, f),\ g)$; and $T_2 = ((a, b, c, d),\ (e, f),\ g)$;.

Throughout the paper, the size of the input to MinRS/MinIS is expressed in terms of $k = |\mathcal{R}|$ and $n = |L|$, where $L = \bigcup_{t\in\mathcal{R}} \Lambda(t)$. For MinRLC/MinILC, $k = |\mathcal{S}|$ and $n = |L|$, where $L = \Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k)$.

1.2 Previous Work

Here, we give an overview of some relevant results from the literature.

BUILD: Aho *et al.* [2] presented a polynomial-time algorithm called BUILD for determining if an input set $\mathcal{R}$ of rooted triplets is consistent, and if so, constructing a tree T with $\Lambda(T) = \bigcup_{t\in\mathcal{R}} \Lambda(t)$ and $\mathcal{R} \subseteq r(T)$. (When the input $\mathcal{R}$ is *not* consistent, one can for example look for a tree T that maximizes $|r(T) \cap \mathcal{R}|$; cf. Section 2 in [8] for a survey on this problem variant.) BUILD is summarized in Section 2.1 below. Henzinger *et al.* [15] gave a faster implementation of BUILD, and we note that substituting the data structure for dynamic graph connectivity used in the proof of Theorem 1 in [15] by the one in [25] yields a time complexity of $\min\{O(n + k\frac{\log^2 n}{\log\log n}),\ O(k + n^2 \log n)\}$, where $k = |\mathcal{R}|$ and $n = |\bigcup_{t\in\mathcal{R}} \Lambda(t)|$.

Importantly, BUILD does not solve MinRS and MinIS. This was first observed by Bryant [6, Section 2.5.2], who gave the following counterexample: $\mathcal{R} = \{bc|a,\ bd|a,\ ef|a,\ eg|a\}$. Given $\mathcal{R}$ as input, BUILD constructs the tree $T_B = (a,\ (b, c, d),\ (e, f, g))$;, which has three

[2] See, e.g., `http://evolution.genetics.washington.edu/phylip/newicktree.html`.

internal nodes and 24 rooted triplets. In contrast, the optimal solution to both MinRS and MinIS is the tree $T_O = (a,\ (b, c, d, e, f, g));$, which has two internal nodes and 15 rooted triplets. As pointed out in [16], the claim by Henzinger *et al.* in [15] that their Algorithm A' always constructs a minimal tree is therefore false. In another highly cited paper, Ng and Wormald [18] presented an extension of BUILD named OneTree to so-called *fans*. However, Note 2 in Section 4 of [18] incorrectly states that OneTree outputs a tree with the minimum number of nodes.

MinRS: For MinRS, the following strong negative result is known: MinRS cannot be approximated within $n^{1-\epsilon}$ for any constant $\epsilon > 0$ in polynomial time, unless P = NP [16]. An algorithm named AllMinTrees in [19] outputs all minor-minimal trees consistent with $\mathcal{R}$, where a tree T is *minor-minimal* if it is consistent with $\mathcal{R}$ and it is not possible to obtain a tree consistent with $\mathcal{R}$ by contracting any edges of T, and this algorithm can be used to solve MinRS. However, it runs in $\Omega((\frac{n}{2})^{n/2})$ time [16], which is self-exponential in $n/2$. Some special cases of MinRS can be solved in polynomial time; e.g., if the output tree has at most three internal nodes or if it is a caterpillar (a tree in which every node has at most one child that is an internal node) [16]. Also, for any positive integer p, if every node in the output tree has at most p children which are internal nodes then MinRS can be solved in $p^{O(n)}$ time [16].

MinIS: To determine the computational complexity of MinIS was listed as an open problem in Section 6 in [16]. As far as we know, it has not been studied previously.

MinRLC, MinILC, and local consensus trees: The MinILC problem originates from Kannan *et al.* [17], who gave several alternative definitions of a "local consensus tree". They called a tree T an *RV-II* ("relaxed version II") tree of two trees T_1 and T_2 with identical leaf label sets if $r(T_1) \cap r(T_2) \subseteq r(T)$ and $|r(T)|$ is minimized. (Thus, an RV-II tree is a solution to MinILC when $k = 2$.) In Section 5.4 of [17], the authors suggested that applying BUILD to the set $r(T_1) \cap r(T_2)$ always produces an RV-II tree, but this is not correct. A counterexample, analogous to the one for MinRS and MinIS above, is obtained by letting T_1 and T_2 be the two trees T_B and T_O, which gives $r(T_1) \cap r(T_2) = \{bc|a,\ bd|a,\ cd|a,\ ef|a,\ eg|a,\ fg|a\}$, so that the solution to both MinRLC and MinILC is T_O while BUILD's output is T_B. This shows that one cannot solve MinRLC and MinILC by taking $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$ and applying BUILD directly.

Bryant [7] later defined the "local consensus tree" as the output of BUILD when given $\bigcap_{i=1}^{k} r(T_i)$ as input. The algorithm in Section 5.4.1 of [17] constructs such a tree in $O(n^2)$ time for the case $k = 2$, while the $O(kn^3)$-time algorithm in Theorem 7 in [15] by Henzinger *et al.* can be used for unbounded k. Note that finding a tree T satisfying only $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$ is trivial since one can just select $T = T_1$, so some additional conditions are needed to make the tree informative. The advantages of Bryant's local consensus tree are that it is unique and can be computed efficiently; the disadvantages are that it does not minimize the number of nodes or induced rooted triplets and that it is defined in terms of the output of an algorithm and not axiomatically.

1.3 Our New Results and Organization of the Paper

Section 2 reviews the BUILD algorithm from [2] and a useful result by Semple in [19] that characterizes all trees consistent with the input $\mathcal{R}$. Based on Semple's characterization, Section 3 presents an $O^*((1+\sqrt{3})^n) = O(2.733^n)$-time algorithm[3] for MinRS and an $O(kn^3 + 2.733^n)$-time algorithm for MinRLC, and Section 4 presents an $O^*(4^n)$-time algorithm for MinIS and an $O(kn^3 + 4^n \cdot poly(n))$-time algorithm for MinILC.

All four problems are NP-hard; MinRS was shown to be NP-hard in [16], and we complement this result by establishing the NP-hardness of MinRLC in Section 5 and the NP-hardness of MinIS and MinILC in Section 6.

2 Preliminaries

2.1 Aho et al.'s BUILD Algorithm [2]

The BUILD algorithm [2] is a recursive, top-down algorithm that takes as input a set $\mathcal{R}$ of rooted triplets and a leaf label set L such that $\bigcup_{t \in \mathcal{R}} \Lambda(t) \subseteq L$ and outputs a tree T with $\Lambda(T) = L$ that is consistent with all of the rooted triplets in $\mathcal{R}$, if such a tree exists; otherwise, it outputs *fail*. The time complexity of BUILD is polynomial (q.v., Section 1.2).

A summary of how BUILD works is given here. It first partitions the leaf label set L into *blocks* based on the information contained in $\mathcal{R}$. More precisely, BUILD constructs an *auxiliary graph*, defined as the undirected graph $\mathcal{G}(L) = (L, E)$ where for any $x, y \in L$, the edge $\{x, y\}$ belongs to E if and only if $\mathcal{R}$ contains at least one rooted triplet of the form $xy|z$ with $z \in L$. It then computes the connected components in $\mathcal{G}(L)$ and assigns the leaf labels in each connected component to one block. (Henceforth, the set of leaf labels belonging to any connected component C in $\mathcal{G}(L)$ is denoted by $\Lambda(C)$, and for every $L' \subseteq L$, we define $\mathcal{R}|L' = \{t \in \mathcal{R} : \Lambda(t) \subseteq L'\}$.) Next, for each block $\Lambda(C)$, BUILD builds a tree T_C by calling itself recursively using $\mathcal{R}|\Lambda(C)$ together with $\Lambda(C)$ as input. Finally, BUILD returns a tree consisting of a newly created root node whose children are the roots of all the recursively constructed T_C-trees. The recursion's base case is when the leaf label set consists of one element x, in which case the algorithm just returns a tree with a single leaf labeled by x. If any auxiliary graph $\mathcal{G}(L')$ constructed during BUILD's execution has only one connected component and $|L'| > 1$ holds then the algorithm terminates and outputs *fail*. See, e.g., [2] for the correctness proof and further details.

Returning to the example in Section 1.2 where $\mathcal{R} = \{bc|a,\ bd|a,\ ef|a,\ eg|a\}$ and $L = \{a, b, c, d, e, f, g\}$, the blocks in the auxiliary graph $\mathcal{G}(L)$ are $\{a\}$, $\{b, c, d\}$, and $\{e, f, g\}$. The auxiliary graphs on the successive recursive levels contain no edges, so BUILD outputs the tree $(a,\ (b, c, d),\ (e, f, g));$.

2.2 Semple's Characterization

In [19], Semple clarified the relationship between the auxiliary graph $\mathcal{G}(L)$ used in the BUILD algorithm and the trees consistent with $\mathcal{R}$. For any tree T, define $\pi(T)$ as the partition of $\Lambda(T)$ whose parts are the leaves in the different subtrees attached to the root of T; as an example, $\pi(T_1) = \{\{a, b\}, \{c, d\}, \{e, f\}, \{g\}\}$ in Figure 1. With this notation, Semple's characterization can be expressed as:

▶ **Lemma 1.** *(Corollary 3.3 in [19]) Let T be any tree that is consistent with $\mathcal{R}$. For each connected component C in $\mathcal{G}(L)$, $\Lambda(C) \subseteq B$ for some $B \in \pi(T)$.*

Lemma 1 implies that if T is any tree consistent with $\mathcal{R}$ then the partition $\pi(T)$ can be obtained by performing zero or more mergings of $\mathcal{G}(L)$'s connected components. Thus, every

[3] The notation $O^*(f(n))$ means $O(f(n) \cdot poly(n))$.

tree consistent with $\mathcal{R}$ can be recovered by trying all possible mergings of the connected components in $\mathcal{G}(L)$ at each recursion level.[4]

We remark that Lemma 1 is very useful. For example, it can be employed to prove the non-uniqueness of solutions to MinRLC and MinILC (and hence, MinRS and MinIS) by considering the instance $\mathcal{S} = \{T_1, T_2, T_3\}$ with $T_1 = ((1,2,3,4,5,6),\ (7,8),\ (9,10),\ 11);$, $T_2 = ((1,2,3,4,5,6,7,8),\ (9,10),\ 11);$, and $T_3 = ((1,2,3,4,5,6,9,10),\ (7,8),\ 11);$. The connected components in $\mathcal{G}(L)$, where $\mathcal{R} = \bigcap_{i=1}^{3} r(T_i)$, consist of $\{1,2,3,4,5,6\}$, $\{7,8\}$, $\{9,10\}$, and $\{11\}$. By Lemma 1, we only need to check a few possible candidate trees (corresponding to the different ways of merging these connected components), and we find that each of T_1, T_2, and T_3 is an optimal solution to MinILC since $|r(T_1)| = |r(T_2)| = |r(T_3)| = 93$. Furthermore, each of T_2 and T_3 is an optimal solution to MinRLC.

3 Exponential-Time Algorithms for MinRS and MinRLC

This section presents an exact $O(2.733^n)$-time algorithm for MinRS. As a consequence, MinRLC can be solved in $O(kn^3 + 2.733^n)$ time.

The main idea is to use Lemma 1 together with dynamic programming. For every $L' \subseteq L$, let $opt(L')$ be the number of internal nodes in an optimal solution to MinRS for $\mathcal{R}|L'$. Clearly, if $|L'| = 1$ then $opt(L') = 0$. To compute $opt(L')$ when $|L'| \geq 2$, observe that if T' is any optimal solution for $\mathcal{R}|L'$ then T' consists of a root node whose children are the roots of the optimal solutions for $\mathcal{R}|P_1$, $\mathcal{R}|P_2$, …, $\mathcal{R}|P_t$, where $\{P_1, P_2, \ldots, P_t\}$ is equal to $\pi(T')$. The partition $\pi(T')$ can be found by enumerating partitions of L' and using dynamic programming to identify the best one; according to Lemma 1, only partitions corresponding to the different ways of merging connected components in the auxiliary graph $\mathcal{G}(L')$ need to be considered.

The details are explained next. Let $\mathcal{C}_{L'}$ be the set of connected components in $\mathcal{G}(L')$. For every subset $\mathcal{D} \subseteq \mathcal{C}_{L'}$, define $Merge(\mathcal{D})$ as the set of all leaf labels belonging to components in $\mathcal{D}$, i.e., $Merge(\mathcal{D}) = \bigcup_{Q \in \mathcal{D}} \Lambda(Q)$. Also define $DP(\mathcal{D})$ for every $\mathcal{D} \subseteq \mathcal{C}_{L'}$ to be the minimum value of $\sum_{\mathcal{X} \in \mathcal{Q}} opt(Merge(\mathcal{X}))$ taken over all possible true partitions $\mathcal{Q}$ of $\mathcal{D}$, where we say that a partition $\mathcal{Q}$ of a set X is a *true partition of* X if $|X| \geq 2$ and $\mathcal{Q} \neq \{X\}$ (i.e., if $|\mathcal{Q}| > 1$), or if $|X| = |\mathcal{Q}| = 1$. Then:

▶ **Lemma 2.** *For every $L' \subseteq L$ with $|L'| \geq 2$, it holds that $opt(L') = DP(\mathcal{C}_{L'}) + 1$.*

Proof. Let T' be any optimal tree for $\mathcal{R}|L'$. The children of the root of T' are the roots of the optimal solutions for $\mathcal{R}|P_1$, $\mathcal{R}|P_2$, …, $\mathcal{R}|P_t$, where each P_i equals $Merge(\mathcal{D})$ for some $\mathcal{D} \subseteq \mathcal{C}_{L'}$ because of Lemma 1. By definition, $DP(\mathcal{C}_{L'})$ is the minimum value of $\sum_{\mathcal{X} \in \mathcal{P}} opt(\mathcal{X})$ over all true partitions $\mathcal{P}$ of L' such that each $\mathcal{X} \in \mathcal{P}$ equals $Merge(\mathcal{D})$ for some $\mathcal{D} \subseteq \mathcal{C}_{L'}$. Together with the common root node, this gives $opt(L') = DP(\mathcal{C}_{L'}) + 1$. ◀

▶ **Lemma 3.** *For every $\mathcal{D} \subseteq \mathcal{C}_{L'}$ with $|\mathcal{D}| \geq 2$, $DP(\mathcal{D}) = \min_{\emptyset \neq \mathcal{X} \subsetneq \mathcal{D}} \big\{ opt(Merge(\mathcal{X})) + \min\{DP(\mathcal{D} \setminus \mathcal{X}),\ opt(Merge(\mathcal{D} \setminus \mathcal{X}))\} \big\}$.*

Proof. $DP(\mathcal{D}) = \min\big\{\sum_{\mathcal{X} \in \mathcal{Q}} opt(Merge(\mathcal{X})) : \mathcal{Q}$ is a true partition of $\mathcal{D}\big\} = \min\big\{opt(Merge(\mathcal{X})) + \min\{DP(\mathcal{D} \setminus \mathcal{X}),\ opt(Merge(\mathcal{D} \setminus \mathcal{X}))\} : \mathcal{X} \in \mathcal{Q},\ \mathcal{Q}$ is a true partition of $\mathcal{D}\big\} = \min\big\{opt(Merge(\mathcal{X})) + \min\{DP(\mathcal{D} \setminus \mathcal{X}),\ opt(Merge(\mathcal{D} \setminus \mathcal{X}))\} : \emptyset \neq \mathcal{X} \subsetneq \mathcal{D}\big\}$. ◀

[4] This technique was actually used even earlier than [19]; the SUPERB algorithm in [10] outputs all binary trees consistent with $\mathcal{R}$ by considering all ways of merging the connected components of $\mathcal{G}(L)$ into exactly two connected components at each recursion level.

```
Algorithm MinRS_exact
Input: Set $\mathcal{R}$ of rooted triplets over a leaf label set $L$.
Output: The number of internal nodes in a minimally resolved tree consistent with $\mathcal{R}$
and leaf-labeled by $L$.
1:  For every $x \in L$, initialize $opt(\{x\}) := 0$;
2:  for $i := 2$ to $n$ do
3:    for every cardinality-$i$ subset $L'$ of $L$ do
4:      Construct $\mathcal{G}(L')$. Let $\mathcal{C}$ and $\mathcal{U}$ be the set of connected components and the set
        of singleton components, respectively, in $\mathcal{G}(L')$;
5:      Let $DP(\emptyset) := 0$. For every $X \in \mathcal{C} \setminus \mathcal{U}$, let $DP(\{X\}) := opt(\Lambda(X))$;
6:      for $j := 2$ to $|\mathcal{C}| - |\mathcal{U}|$ do
7:        for every cardinality-$j$ subset $\mathcal{D}$ of $\mathcal{C} \setminus \mathcal{U}$ do
8:          $DP(\mathcal{D}) :=$
              $\min_{\emptyset \neq \mathcal{X} \subsetneq \mathcal{D}} \{opt(\bigcup_{Q \in \mathcal{X}} \Lambda(Q)) + \min\{DP(\mathcal{D} \setminus \mathcal{X}), opt(\bigcup_{Q \in \mathcal{D} \setminus \mathcal{X}} \Lambda(Q))\}\}$;
9:        end for
10:     end for
11:     $opt(L') := DP(\mathcal{C} \setminus \mathcal{U}) + 1$;
12:   end for
13: end for
14: return $opt(L)$;
```

Figure 2 Algorithm `MinRS_exact`.

Lemmas 2 and 3 suggest the following strategy: Compute $opt(L')$ for all subsets L' of L in order of increasing cardinality by evaluating the formula in Lemma 2, while using dynamic programming to compute and store the relevant DP-values. To do this, for each L', we first construct $\mathcal{G}(L')$ in polynomial time. We then enumerate all subsets $\mathcal{D}$ of $\mathcal{C}_{L'}$ in a loop having $|\mathcal{C}_{L'}|$ iterations in which iteration j uses Lemma 3 to compute all $DP(\mathcal{D})$-values where $|\mathcal{D}| = j$. Each application of Lemma 3 takes $O^*(2^{|\mathcal{D}|})$ time, so this takes a total of $O^*(\sum_{j=1}^{|\mathcal{C}_{L'}|} \binom{|\mathcal{C}_{L'}|}{j} 2^j) = O^*((2+1)^{|\mathcal{C}_{L'}|}) = O^*(3^{|\mathcal{C}_{L'}|})$ time for each L'. To obtain $opt(L)$, we iterate over all subsets L' of L of cardinality $i = 1, 2, \ldots, n$; iteration i computes $opt(L')$ for each L' with $|L'| = i$ in $O^*(3^{|\mathcal{C}_{L'}|})$ time as just described. The total running time becomes $O^*(\sum_{i=1}^{n} \binom{n}{i} 3^i) = O^*((3+1)^n) = O^*(4^n)$.

To reduce the time complexity, we will reduce the number of applications of Lemma 3 in the main loop that computes $opt(L')$ for any $L' \subseteq L$. We rely on the following simple observation, which essentially tells us that the singleton components of $\mathcal{G}(L')$ can be ignored.

▶ **Lemma 4.** *Let $\mathcal{U}$ be the set of singleton components in $\mathcal{G}(L')$. $DP(\mathcal{C}_{L'}) = DP(\mathcal{C}_{L'} \setminus \mathcal{U})$.*

Proof. Consider any $x \in \mathcal{U}$. By the construction of $\mathcal{G}(L')$ and $\mathcal{U}$, there are no rooted triplets of the form $xy|z$ for any $y, z \in L'$ in the set $\mathcal{R}|L'$. Hence, there exists a minimally resolved tree consistent with $\mathcal{R}|L'$ in which x is attached directly to the root. The lemma follows. ◀

The resulting algorithm, called `MinRS_exact`, is summarized in Figure 2.

▶ **Theorem 5.** *Algorithm `MinRS_exact` solves* MinRS *in $O^*((1+\sqrt{3})^n)$ time.*

Proof. First note that $\mathcal{C}_{L'} \setminus \mathcal{U}$ contains no singleton components. Therefore, the number of connected components in $\mathcal{C}_{L'} \setminus \mathcal{U}$ is at most $\frac{|L'|}{2}$, i.e., $|\mathcal{C}_{L'}| - |\mathcal{U}| \leq \frac{|L'|}{2}$. Now, when computing

$opt(L')$ for any subset L' of L, the number of applications of the formula in Lemma 3 is reduced since there no are subsets $\mathcal{D}$ of cardinality larger than $|\mathcal{C}_{L'}| - |\mathcal{U}|$. More precisely, the time for each L' is reduced to $O^*(\sum_{j=1}^{|\mathcal{C}_{L'}|-|\mathcal{U}|} \binom{|\mathcal{C}_{L'}|-|\mathcal{U}|}{j} 2^j) = O^*((2+1)^{|\mathcal{C}_{L'}|-|\mathcal{U}|}) = O^*(3^{|\mathcal{C}_{L'}|-|\mathcal{U}|}) = O^*(\sqrt{3}^{|L'|})$. Finally, replacing $O^*(3^{|\mathcal{C}_{L'}|})$ by $O^*(\sqrt{3}^{|L'|})$ in the analysis of computing $opt(L)$ above gives a total time complexity of $O^*(\sum_{i=1}^{n} \binom{n}{i} \sqrt{3}^i) = O^*((\sqrt{3}+1)^n)$. ◀

▶ Remark. The algorithm as presented here returns $opt(L)$. An optimal tree with this number of internal nodes can be obtained by standard traceback techniques.

▶ **Corollary 6.** MinRLC *can be solved in* $O(kn^3 + (1+\sqrt{3})^n \cdot poly(n))$ *time.*

Proof. First construct $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$ in $O(kn^3)$ time, e.g., by preprocessing each T_i in $O(n)$ time so that any query of the form $lca(x, y)$ in T_i with $x, y \in L$ can be answered in $O(1)$ time [3] and then, for every $L' \subseteq L$ with $|L'| = 3$, doing $3k$ *lca*-queries to see if L' induces the same rooted triplet in all of the k trees. Next, run `MinRS_exact` on $\mathcal{R}$. ◀

4 Exponential-Time Algorithms for MinIS and MinILC

We now describe an $O^*(4^n)$-time algorithm for MinIS based on the technique from Section 3. Applying it to MinILC yields an $O(kn^3 + 4^n \cdot poly(n))$-time algorithm for the latter.

Lemma 1 guarantees that every valid solution to MinIS can be discovered by trying all ways of merging connected components in the auxiliary graphs $\mathcal{G}(L')$. As in Section 3, we use dynamic programming to compute and store optimal values to subproblems but make the following modifications. First of all, redefine opt so that $opt(L')$ for every $L' \subseteq L$ means the value of $|r(T')|$ for an optimal solution T' to MinIS for $\mathcal{R}|L'$. Secondly, redefine $DP(\mathcal{D})$ for every $\mathcal{D} \subseteq \mathcal{C}_{L'}$ to mean the minimum value of $\sum_{\mathcal{X} \in \mathcal{Q}}(opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})|)$, taken over all possible true partitions $\mathcal{Q}$ of $\mathcal{D}$. With the new definitions of opt and DP, the analogues of Lemmas 2 and 3 become:

▶ **Lemma 7.** *For every* $L' \subseteq L$ *with* $|L'| \geq 2$*, it holds that* $opt(L') = DP(\mathcal{C}_{L'})$*.*

Proof. $DP(\mathcal{C}_{L'})$ counts the minimum number of rooted triplets in a tree consistent with $\mathcal{R}|L'$ among all partitions $\mathcal{Q}$ of $\mathcal{C}_{L'}$. Hence, $opt(L') = DP(\mathcal{C}_{L'})$. ◀

▶ **Lemma 8.** *For every* $\mathcal{D} \subseteq \mathcal{C}_{L'}$ *with* $|\mathcal{D}| \geq 2$, $DP(\mathcal{D}) = \min_{\emptyset \neq \mathcal{X} \subsetneq \mathcal{D}} \{opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})| + \min\{DP(\mathcal{D} \setminus \mathcal{X}),\ opt(Merge(\mathcal{D} \setminus \mathcal{X})) + \binom{|Merge(\mathcal{D} \setminus \mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{D} \setminus \mathcal{X})|\}\}$.

Proof. $DP(\mathcal{D}) = \min\{\sum_{\mathcal{X} \in \mathcal{Q}}(opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})|) : \mathcal{Q}$ is a true partition of $\mathcal{D}\} = \min\{opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})| + \min\{DP(\mathcal{D} \setminus X),\ opt(Merge(\mathcal{D} \setminus \mathcal{X})) + \binom{|Merge(\mathcal{D} \setminus \mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{D} \setminus \mathcal{X})|\} : \mathcal{X} \in \mathcal{Q},\ \mathcal{Q}$ is a true partition of $\mathcal{D}\}$ $= \min\{opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})| + \min\{DP(\mathcal{D} \setminus X),\ opt(Merge(\mathcal{D} \setminus \mathcal{X})) + \binom{|Merge(\mathcal{D} \setminus \mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{D} \setminus \mathcal{X})|\} : \emptyset \neq \mathcal{X} \subsetneq \mathcal{D}\}$. ◀

The new algorithm, called `MinIS_exact`, is obtained by modifying Algorithm `MinRS_exact` as follows:

- Change Step 8 so that it computes $DP(\mathcal{D})$ using Lemma 8 instead Lemma 3.
- Change Step 11 so that it assigns $opt(L') := DP(\mathcal{C}_{L'})$, in accordance with Lemma 7.
- Change Step 4 so that it always sets $\mathcal{U}$ to $\emptyset$.

The reason why we force $\mathcal{U} = \emptyset$ is that we do not have an analogue of Lemma 4 for MinIS that would allow us to ignore the singleton components. The algorithm therefore spends $O^*(\sum_{j=1}^{|\mathcal{C}_{L'}|} \binom{|\mathcal{C}_{L'}|}{j} 2^j) = O^*((2+1)^{|\mathcal{C}_{L'}|}) = O^*(3^{|\mathcal{C}_{L'}|})$ time for each L', just like the slower version of Algorithm `MinRS_exact` in Section 3, and the total running time is $O^*(\sum_{i=1}^{n} \binom{n}{i} 3^i)$ $= O^*((3+1)^n) = O^*(4^n)$.

▶ **Theorem 9.** *Algorithm* `MinIS_exact` *solves* MinIS *in $O^*(4^n)$ time.*

▶ **Corollary 10.** MinILC *can be solved in $O(kn^3 + 4^n \cdot poly(n))$ time.*

5 NP-Hardness of MinRLC

Section 3 in [16] proved that MinRS is NP-hard. It follows from the proof in [16] that MinRS remains NP-hard even if restricted to a particular special case which we now describe.

Suppose that $L_0 = \{v_1, v_2, \ldots, v_q\}$ is a set of elements. Define $L_0' = \{v_{1'}, v_{1''}, v_{2'}, v_{2''}, \ldots,$ $v_{q'}, v_{q''}\}$, and for any integers i, j with $1 \leq i < j \leq q$, define $\mathcal{R}(v_i, v_j)$ as the set of four rooted triplets $\{v_{i'}v_{i''}|v_{j'},\ v_{i'}v_{i''}|v_{j''},\ v_{j'}v_{j''}|v_{i'},\ v_{j'}v_{j''}|v_{i''}\}$ over L_0'. For any set S, let $\binom{S}{2}$ denote the set of all subsets of S of cardinality 2. According to Section 3 in [16], MinRS is NP-hard even if restricted to instances where $\mathcal{R}$ has the form $\mathcal{R} = \bigcup_{\{v_i, v_j\} \in Z} \mathcal{R}(v_i, v_j)$ for some set L_0 and some $Z \subseteq \binom{L_0}{2}$.

▶ **Theorem 11.** MinRLC *is NP-hard.*

Proof. We reduce from the above variant of MinRS. Let $\mathcal{R}$ be any given instance of the problem. Let P be the set of pairs of indices that form rooted triplets in $\mathcal{R}$, i.e., $P = \bigl\{\{i, j\} : v_{i'}v_{i''}|v_{j'},\ v_{i'}v_{i''}|v_{j''},\ v_{j'}v_{j''}|v_{i'},\ v_{j'}v_{j''}|v_{i''} \in \mathcal{R}\bigr\}$, and let $Q = \binom{\{1,2,\ldots,q\}}{2} \setminus P$.

Define a tree $T_0 = ((v_{1'}, v_{1''}), (v_{2'}, v_{2''}), \ldots, (v_{q'}, v_{q''}))$; and for every $f = \{x, y\} \in Q$, define a tree T_f by taking a copy of T_0 and merging the two subtrees $(v_{x'}, v_{x''})$ and $(v_{y'}, v_{y''})$ so that $T_f = ((v_{x'}, v_{x''}, v_{y'}, v_{y''}), (v_{1'}, v_{1''}), (v_{2'}, v_{2''}), \ldots, (v_{n'}, v_{n''}))$;. Let $\mathcal{S} = \{T_0\} \cup \{T_f :$ $f \in Q\}$. Note that $\mathcal{R} = \bigcap_{T_i \in \mathcal{S}} r(T_i)$. This is because for any $\{x, y\} \in P$, the four rooted triplets $v_{x'}v_{x''}|v_{y'},\ v_{x'}v_{x''}|v_{y''},\ v_{y'}v_{y''}|v_{x'},\ v_{y'}v_{y''}|v_{x''}$ appear in $\mathcal{R}$ as well as in $r(T_i)$ for every $T_i \in \mathcal{S}$. On the other hand, for any $\{x, y\} \in Q$, $v_{x'}v_{x''}|v_{y'},\ v_{x'}v_{x''}|v_{y''},\ v_{y'}v_{y''}|v_{x'},\ v_{y'}v_{y''}|v_{x''}$ do not appear in $\mathcal{R}$ or in $r(T_{\{x,y\}})$. Thus, there exists a tree T with $\bigcap_{T_i \in \mathcal{S}} r(T_i) \subseteq r(T)$ having x internal nodes if and only if there exists a tree T' with $\mathcal{R} \subseteq r(T')$ having x internal nodes. ◀

6 NP-Hardness of MinILC and MinIS

To prove the NP-hardness of MinILC, we give a polynomial-time reduction from the Maximum Clique problem, which is NP-hard [12]. Maximum Clique takes as input an undirected graph $G = (V, E)$ and asks for a largest clique in G, where $X \subseteq V$ is a *clique in G* if every two vertices belonging to X are adjacent in G.

The reduction is as follows. Let $n = |V|$ and write $V = \{1, 2, \ldots, n\}$. Create a set L of leaf labels such that $L = \{v_i, v_i' : i \in V\} \cup \{z, w_1, w_2, \ldots, w_{n^2}\}$. Define a tree $T_\emptyset =$ $(z, (w_1, w_2, \ldots, w_{n^2}), (v_1, v_1'), (v_2, v_2'), \ldots, (v_n, v_n'))$; with $\Lambda(T_\emptyset) = L$. For any nonempty subset $X = \{i_1, i_2, \ldots, i_p\} \subseteq V$, let T_X be the tree with $\Lambda(T_X) = L$ obtained by taking a copy of $T_\emptyset$, deleting the subtrees (v_i, v_i') for all $i \in X$, and replacing the subtree $(w_1, w_2, \ldots, w_{n^2})$ by $((w_1, w_2, \ldots, w_{n^2}, v_{i_1}, v_{i_2}, \ldots, v_{i_p}),\ v_{i_1}', v_{i_2}', \ldots, v_{i_p}')$. Finally, let $\mathcal{S} = \{T_\emptyset\} \cup \{T_{\{i\}} : i \in$ $V\} \cup \{T_{\{i,j\}} : \{i, j\} \in E\}$.

Section 6.1 first states some general properties satisfied by the trees defined above. Then, Section 6.2 establishes some specific properties satisfied by any local consensus tree of $\mathcal{S}$. After that, we will prove that for any $X \subseteq V$, X is a maximum clique in G if and only if T_X is a local consensus tree of $\mathcal{S}$ with the smallest possible number of rooted triplets, giving the main result of this section.

6.1 General Properties

The following additional notation is used. For any tree H, *Child*(H) is the set of children of the root of H. For any $u \in V(H)$, the subtree of H induced by u and all proper descendants of u is called *the subtree of H rooted at u* and is denoted by H^u. For any $L' \subseteq \Lambda(H)$, $H|L'$ is the tree obtained from H by deleting all nodes with no descendants in L' and their incident edges, and then contracting every edge between a node having one child and its child. Finally, for every positive integer n, define a function $f_n(k) = n^5 - \frac{k-1}{2}n^4 + 4kn^3 - \frac{6k^2-3k-3}{2}n^2 + (4k^2-4k-1)n - \frac{7k^3-7k^2}{2}$. We immediately have:

▶ **Lemma 12.** *For any tree H, $|r(H)| = \sum_{u \in Child(H)} \Big(|r(H^u)| + \binom{|\Lambda(H^u)|}{2} \cdot (|\Lambda(H)| - |\Lambda(H^u)|)\Big)$.*

▶ **Lemma 13.** *Let X be any subset of the given V. Write $k = |X|$. Then $|r(T_X)| = f_n(k)$.*

Proof. By Lemma 12, the number of rooted triplets consistent with T_X is $\binom{n^2+k}{2} \cdot k + \binom{n^2+2k}{2} \cdot (2n - 2k + 1) + (n-k) \cdot (n^2 + 2n - 1)$. Expanding this expression yields the formula. ◀

▶ **Corollary 14.** *For any fixed $n \geq 8$, $f_n(k)$ is strictly decreasing as k increases.*

Proof. By Lemma 13, $f_n(k+1) - f_n(k) = -\frac{1}{2}n^4 + 4n^3 - \frac{12k+3}{2}n^2 + 8kn - \frac{7}{2}(3k^2 + k)$. Since $n \geq 8$, $-\frac{1}{2}n^4 + 4n^3 \leq 0$ holds. Also, $\frac{12k+3}{2}n > 8k$ for $n \geq 8$ and $-\frac{7}{2}(3k^2+k) \leq 0$. The corollary follows. ◀

▶ **Lemma 15.** *Consider any $u \in Child(H)$ in a tree H. Suppose that $\Lambda(H^u) = \alpha \cup \beta$ for some $\alpha, \beta \neq \emptyset$ with $\alpha \cap \beta = \emptyset$. Let H' be the tree obtained from H by deleting H^u and its parent edge and attaching the roots of $H|\alpha$ and $H|\beta$ as children of the root of H. If $|\alpha| + |\beta| \leq \frac{2|\Lambda(H)|}{3}$ then $|r(H')| < |r(H)|$.*

Proof. Define $m = |\Lambda(H)|$. Lemma 12 gives $|r(H)| - |r(H')| = |r(H^u)| + \binom{|\alpha|+|\beta|}{2} \cdot (m - |\alpha| - |\beta|) - |r(H|\alpha)| - \binom{|\alpha|}{2} \cdot (m - |\alpha|) - |r(H|\beta)| - \binom{|\beta|}{2} \cdot (m - |\beta|)$. Noting that $|r(H^u)| \geq |r(H|\alpha)| + |r(H|\beta)|$, we have $|r(H)| - |r(H')| \geq \binom{|\alpha|+|\beta|}{2} \cdot (m - |\alpha| - |\beta|) - \binom{|\alpha|}{2} \cdot (m - |\alpha|) - \binom{|\beta|}{2} \cdot (m - |\beta|) = |\alpha| \cdot |\beta| \cdot (m + 1 - \frac{3}{2} \cdot (|\alpha| + |\beta|)) \geq |\alpha| \cdot |\beta| \cdot (m + 1 - m) = |\alpha| \cdot |\beta| > 0$. ◀

6.2 Properties of a Local Consensus Tree of $\mathcal{S}$

By the definition of $\mathcal{S}$, we have the next lemma.

▶ **Lemma 16.** *The set $\bigcap_{T_i \in \mathcal{S}} r(T_i)$ consists of the following rooted triplets:*

- *$w_i w_j | z$ for all $1 \leq i < j \leq n^2$ and $v_i v'_i | z$ for all $1 \leq i \leq n$;*
- *$w_i w_j | v'_k$ for all $1 \leq i < j \leq n^2$ and $1 \leq k \leq n$;*
- *$v_i v'_i | v_j$, $v_i v'_i | v'_j$, $v_j v'_j | v_i$, and $v_j v'_j | v'_i$ for all $1 \leq i < j \leq n$ with $\{i, j\} \notin E$.*

Let T be any local consensus tree of $\mathcal{S}$, i.e., any tree T such that $\Lambda(T) = L$ and $\bigcap_{T_i \in \mathcal{S}} r(T_i) \subseteq r(T)$. According to Lemma 16, $r(T)$ contains $v_i v'_i | z$ for all $1 \leq i \leq n$, so the two leaves v_i and v'_i must belong to the same subtree attached to the root of T for all $1 \leq i \leq n$. Similarly, all leaves in $\{w_1, w_2, \ldots, w_{n^2}\}$ must belong to one subtree attached to the root of T. The *core of* T, denoted by γ_T, is the subtree of T rooted at the node $lca(w_1, w_2, \ldots, w_{n^2})$. The path from the root of T to the parent of γ_T is called the *core path of* T. For any node $u \in V(T)$, if u is a child of the core path of T that does not belong to the core path itself and $u \neq lca(w_1, w_2, \ldots, w_{n^2})$, the subtree of T rooted at u is called a *secondary subtree of* T. Note that the secondary subtrees of T are disjoint. Define $C_T = \{i : v_i \in \Lambda(\gamma_T)\}$.

▶ **Lemma 17.** *Let T be a local consensus tree of $\mathcal{S}$. T has the following properties:*

1. *The core γ_T does not contain the leaf v'_i for any $1 \leq i \leq n$.*
2. *C_T forms a clique in G.*
3. *For any $i \in \{1, 2, \ldots n\}$, if $C_T \cup \{i\}$ is not a clique in G then v_i and v'_i belong to the same secondary subtree of T.*

Proof.

1. Suppose $v'_i \in \Lambda(\gamma_T)$. Let w_a, w_b be any two leaves such that $lca(\{w_a, w_b\}) = lca(\{w_1, w_2, \ldots, w_{n^2}\})$. Then, $w_a w_b | v'_i \notin r(T)$, contradicting Lemma 16.
2. Consider any $i, j \in C_T$ with $i \neq j$. By point 1., $v_i v_j | v'_i \in r(T)$, so $v_i v'_i | v_j \notin r(T)$. According to Lemma 16, $\{i, j\} \notin E$ does not hold, which means that $\{i, j\} \in E$.
3. Since $C_T \cup \{i\}$ is not a clique, there exists some $j \in C_T$ where $\{i, j\} \notin E$. By Lemma 16, $v_i v'_i | v_j \in r(T)$. Thus, v_i and v'_i are in the same subtree attached to the core path.

◀

Observe that Lemma 17.1 implies $\Lambda(\gamma_T) = \{w_1, w_2, \ldots, w_{n^2}\} \cup \{v_p \mid p \in C_T\}$. Moreover, by Lemma 17.3, for any $i \in \{1, 2, \ldots n\}$, if v_i and v'_i belong to subtrees attached to different nodes on the core path then $C_T \cup \{i\}$ is a clique in G.

▶ **Lemma 18.** *Let $n \geq 10$ and let T be a local consensus tree of $\mathcal{S}$. T can be transformed into a local consensus tree of $\mathcal{S}$ of the form T_X for some $X \subseteq V$ where X is a clique in G and $|r(T_X)| \leq |r(T)|$.*

Proof. We describe a sequence of transformations that can be applied to T without increasing the number of rooted triplets consistent with it. After each transformation, the resulting tree still contains all of the rooted triplets listed in Lemma 16, so it is still a local consensus tree of $\mathcal{S}$.

First, consider the leaf z in T. Let P be the path from the root of T to z, and let $\rho_1, \rho_2, \ldots, \rho_e$ be the disjoint subtrees of T attached to P whose roots do not belong to P themselves. Let T^1 be the tree formed by removing P and attaching z and the roots of $\rho_1, \rho_2, \ldots, \rho_e$ as children of the root. Then $|r(T^1)| \leq |r(T)|$, and T^1 has the property that z is a child of the root of T^1.

Secondly, transform T^1 to T^2 by contracting the core γ_{T^1}, i.e., by replacing γ_{T^1} by a single node to which all leaves in $\Lambda(\gamma_{T^1})$ are directly attached. Clearly, $|r(T^2)| \leq |r(T^1)|$.

Thirdly, suppose that for some $q \in \{1, 2, \ldots, n\}$, it holds that $q \notin C_{T^2}$ while $C_{T^2} \cup \{q\}$ is a clique in G. Let T^3 be the tree formed by removing the leaf v_q from its location in T^2 and attaching it to the root of γ_{T^2}, and attaching the leaf v'_q as a child of the root of γ_{T^2}. There are two types of rooted triplets involving v_q: (i) $x v_q | y$ and (ii) $xy | v_q$. For (i), there are at most $n^2 + n - 1$ choices of x by Lemma 17.1 and at most $2n$ choices of y, so T^3 has at most $(n^2 + n - 1) \cdot 2n$ more rooted triplets than T^2 of this form. For (ii), there are at least

$\binom{n^2}{2}$ such rooted triplets in T^2 but not in T^3, corresponding to pairs of the form (w_i, w_j), and at most $\binom{2n}{2}$ such rooted triplets in T^3 that are not present in T^2. Similarly, there are two types of rooted triplets involving v'_q: (iii) $xv'_q|y$ and (iv) $xy|v'_q$. As above, T^3 has at most $(n^2+n-1)\cdot 2n$ more rooted triplets than T^2 of the form (iii) and at most $\binom{2n}{2}$ rooted triplets of the form (iv). Hence, by transforming T^2 to T^3, the number of rooted triplets is reduced by at least $\binom{n^2}{2} - 2\cdot\binom{2n}{2} - 2\cdot(n^2+2-1)\cdot(2n)$, which is larger than 0 when $n \geq 10$. We repeat this step until T^3 has no leaf v_q such that $q \notin C_{T^3}$ and $C_{T^3} \cup \{q\}$ is a clique. This gives $|r(T^3)| \leq |r(T^2)|$.

Next, transform T^3 to a tree T^4 in which every secondary subtree contains at most two leaves and, furthermore, the leaves in any secondary subtree with precisely two leaves are of the form $\{v_q, v'_q\}$ where $C_{T^4} \cup \{q\}$ is not a clique in G. To do this, consider each secondary subtree s of T^3. By the definition of T^3 in the previous paragraph, $C_{T^3} \cup \{q\}$ is not a clique in G for any $v_q \in \Lambda(s)$. While $|\Lambda(s)| > 2$, extract any pair of leaves $\{v_q, v'_q\}$ from s (recall from Lemma 17.3 that any two leaves of the form v_i and v'_i must belong to the same secondary subtree), and create a new secondary subtree with the leaves $\{v_q, v'_q\}$ attached to the core path as a sibling of s. Every secondary subtree s satisfies $|\Lambda(s)| \leq 2n \leq \frac{2n^2}{3} \leq \frac{2|\Lambda(H)|}{3}$, where H is the subtree rooted at the parent of the root of s, so we get $|r(T^4)| \leq |r(T^3)|$ by Lemma 15.

Lastly, transform T^4 to a tree T^5 whose core path consists of a single edge (u_0, u_1), where u_0 is the root of T^5, as follows. Attach each secondary subtree of T^4 having two leaves as a child of u_0, and attach each secondary subtree of T^4 having one leaf as a child of u_1. Attach z as a child of u_0 and the core γ_{T^4} as a child of u_1. Note that this will not destroy any of the rooted triplets in Lemma 16, and that $|r(T^5)| \leq |r(T^4)|$.

By the definition of T_X, T^5 is equal to T_X if we select $X = C_{T^5}$. Finally, C_{T^5} is a clique in G by Lemma 17.2. ◀

▶ **Lemma 19.** *Let $n \geq 10$. $X \subseteq V$ is a maximum clique in G if and only if T_X is a local consensus tree of $\mathcal{S}$ that minimizes the number of rooted triplets.*

Proof. ($\rightarrow$) For the purpose of obtaining a contradiction, suppose there exists a local consensus tree T' of $\mathcal{S}$ with $|r(T')| < |r(T_X)|$. Apply Lemma 18 to T' to get a tree T_Q that is also a local consensus tree of $\mathcal{S}$ with $|r(T_Q)| \leq |r(T')|$ and where Q is a clique in G. Then $|Q| > |X|$ by Lemma 13 and Corollary 14, which is impossible.

($\leftarrow$) Suppose X' is a larger clique in G than X. Lemma 13 and Corollary 14 imply $|r(T_{X'})| < |r(T_X)|$, contradicting that T_X is a local consensus tree of $\mathcal{S}$ minimizing the number of rooted triplets. ◀

Now, assuming without loss of generality that $n \geq 10$ in the reduction from Maximum Clique above, Lemma 19 gives:

▶ **Theorem 20.** MinILC *is NP-hard.*

Finally, by the reduction from MinILC to MinIS mentioned in Section 1.1:

▶ **Corollary 21.** MinIS *is NP-hard.*

7 Concluding Remarks

The main open problem is to obtain faster exponential-time algorithms than the ones presented here. In particular, can MinRS be solved in $O^*(2^n)$ time?

Another open problem is to extend the algorithms in this paper to unrooted phylogenetic trees. This would be interesting because many existing methods for inferring phylogenetic trees produce unrooted trees [11]. The unrooted case appears to be much harder than the rooted case, as the basic problem of determining the consistency of an input set of rooted triplets is solvable in polynomial time (see Section 1.2), while the corresponding problem for *unrooted quartets* (unrooted, distinctly leaf-labeled trees with exactly four leaves each and in which every internal node has at least three neighbors) is already NP-hard [22].

References

1 E. N. Adams III. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21(4):390–397, 1972.

2 A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

3 M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics* (LATIN 2000), volume 1776 of *LNCS*, pages 88–94. Springer-Verlag, 2000.

4 O. R. P. Bininda-Emonds. The evolution of supertrees. *TRENDS in Ecology and Evolution*, 19(6):315–322, 2004.

5 O. R. P. Bininda-Emonds, M. Cardillo, K. E. Jones, R. D. E. MacPhee, R. M. D. Beck, R. Grenyer, S. A. Price, R. A. Vos, J. L. Gittleman, and A. Purvis. The delayed rise of present-day mammals. *Nature*, 446(7135):507–512, 2007.

6 D. Bryant. *Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis.* PhD thesis, University of Canterbury, Christchurch, New Zealand, 1997.

7 D. Bryant. A classification of consensus methods for phylogenetics. In M. F. Janowitz, F.-J. Lapointe, F. R. McMorris, B. Mirkin, and F. S. Roberts, editors, *Bioconsensus*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. American Mathematical Society, 2003.

8 J. Byrka, S. Guillemot, and J. Jansson. New results on optimizing rooted triplets consistency. *Discrete Applied Mathematics*, 158(11):1136–1147, 2010.

9 B. Chor, M. Hendy, and D. Penny. Analytic solutions for three taxon ML trees with variable rates across sites. *Discrete Applied Mathematics*, 155(6–7):750–758, 2007.

10 M. Constantinescu and D. Sankoff. An efficient algorithm for supertrees. *Journal of Classification*, 12(1):101–112, 1995.

11 J. Felsenstein. *Inferring Phylogenies.* Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.

12 M. Garey and D. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

13 L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of constructing evolutionary trees. *Journal of Combinatorial Optimization*, 3(2–3):183–197, 1999.

14 Y. J. He, T. N. D. Huynh, J. Jansson, and W.-K. Sung. Inferring phylogenetic relationships avoiding forbidden rooted triplets. *Journal of Bioinformatics and Computational Biology*, 4(1):59–74, 2006.

15 M. R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999.

16 J. Jansson, R. S. Lemence, and A. Lingas. The complexity of inferring a minimally resolved phylogenetic supertree. *SIAM Journal on Computing*, 41(1):272–291, 2012.

17 S. Kannan, T. Warnow, and S. Yooseph. Computing the local consensus of trees. *SIAM Journal on Computing*, 27(6):1695–1724, 1998.

18 M. P. Ng and N. C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1–2):19–31, 1996.

19 C. Semple. Reconstructing minimal rooted trees. *Discrete Applied Mathematics*, 127(3):489–503, 2003.

20 C. Semple, P. Daniel, W. Hordijk, R. D. M. Page, and M. Steel. Supertree algorithms for ancestral divergence dates and nested taxa. *Bioinformatics*, 20(15):2355–2360, 2004.

21 S. Snir and S. Rao. Using Max Cut to enhance rooted trees consistency. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):323–333, 2006.

22 M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, 1992.

23 W.-K. Sung. *Algorithms in Bioinformatics: A Practical Introduction.* Chapman & Hall/CRC, Boca Raton, Florida, 2010.

24 S. J. Willson. Constructing rooted supertrees using distances. *Bulletin of Mathematical Biology*, 66(6):1755–1783, 2004.

25 C. Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2013), pages 1757–1769. SIAM, 2013.

Quantum Communication Complexity of Distributed Set Joins

Stacey Jeffery[1] **and François Le Gall**[2]

1 **Institute for Quantum Information and Matter, California Institute of Technology, Pasadena, USA**
`sjeffery@caltech.edu`

2 **Graduate School of Informatics, Kyoto University, Kyoto, Japan**
`legall@i.kyoto-u.ac.jp`

Abstract

Computing set joins of two inputs is a common task in database theory. Recently, Van Gucht, Williams, Woodruff and Zhang [PODS 2015] considered the complexity of such problems in the natural model of (classical) two-party communication complexity and obtained tight bounds for the complexity of several important distributed set joins.

In this paper we initiate the study of the *quantum* communication complexity of distributed set joins. We design a quantum protocol for distributed Boolean matrix multiplication, which corresponds to computing the *composition join* of two databases, showing that the product of two $n \times n$ Boolean matrices, each owned by one of two respective parties, can be computed with $\widetilde{O}(\sqrt{n}\ell^{3/4})$ qubits of communication, where ℓ denotes the number of non-zero entries of the product. Since Van Gucht et al. showed that the classical communication complexity of this problem is $\widetilde{\Theta}(n\sqrt{\ell})$, our quantum algorithm outperforms classical protocols whenever the output matrix is sparse. We also show a quantum lower bound and a matching classical upper bound on the communication complexity of distributed matrix multiplication over $\mathbb{F}_2$.

Besides their applications to database theory, the communication complexity of set joins is interesting due to its connections to direct product theorems in communication complexity. In this work we also introduce a notion of *all-pairs* product theorem, and relate this notion to standard direct product theorems in communication complexity.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Quantum communication complexity, Distributed quantum computing, Database joins

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.54

1 Introduction

Background

Set joins are basic operations in relational database theory. The notion of set join was introduced to the database community more than forty years ago by Codd [6] to express operations combining two tables in relational databases. This seminal paper considered, in particular, the *composition join*: given two (relational) databases A and B, A represented as a subset of $\{1,\ldots,m\}\times\{1,\ldots,n\}$ and B as a subset of $\{1,\ldots,n\}\times\{1,\ldots,m\}$, the composition join of A and B is the set $\{(i,j) \mid \exists k : (i,k) \in A \text{ and } (k,j) \in B\} \subseteq \{1,\ldots,m\} \times \{1,\ldots,m\}$. Many other join operations have been defined so far and have found many applications (see, e.g., [3, 6, 9, 15, 17, 18, 19, 23]).

The computational complexity of join operations is naturally an important issue. Very recently Van Gucht, Williams, Woodruff and Zhang [23] have investigated this question in the two-party communication complexity model where one party owns the first database, the second party owns the second database, and both parties collaborate to compute the join of these two databases using as little communication as possible. This model is interesting for two main reasons. First, it models the natural and practical task of distributed computation of join operations. Second, in the communication complexity setting it is possible to show strong lower bounds on the complexity of problems. Indeed, one of the main contributions of [23] was to show quantitative differences between the (communication) complexities of various join operations.

Many join operations studied in database theory actually correspond to fundamental and well-studied computational tasks in other areas of computer science. The composition join mentioned above, in particular, corresponds to Boolean matrix multiplication, one central problems in theoretical computer science: if we represent the database A by an $m \times n$ matrix M_A and B by an $n \times m$ matrix M_B (such that $M_A[i,j] = 1$ if and only if $(i,j) \in A$, and similarly for M_B), the matrix representation of the composition join of A and B is precisely the output of the Boolean matrix multiplication of M_A and M_B (i.e., the $m \times m$ matrix C such that $C[i,j] = \bigvee_{k=1}^{n} M_A[i,k] \wedge M_B[k,j]$). The result by Van Gucht et al. on the communication complexity of the composition join [23] shows that the communication complexity of Boolean matrix multiplication is $\widetilde{\Theta}(n\sqrt{\ell})$ for the square case $m = n$ (a more complicated formula is also given for the rectangular case), where ℓ denotes the number of non-zero entries in the product C. Since the parameter ℓ represents the sparsity of the output matrix, algorithms and communication protocols with complexity depending explicitly on ℓ are sometimes called output-sensitive and have been studied in several settings other than communication complexity as well [2, 5, 11, 16].

Our Results

In this paper we initiate the study of the *quantum* communication complexity of distributed set joins. Our main result is about the set joins related to matrix multiplication. We first show that the quantum communication complexity of the composition join (i.e., Boolean matrix multiplication) is $O(\sqrt{n}\ell^{3/4}\log m)$ (Theorem 7). This is better than the best possible classical protocol, which costs $\Omega(n\sqrt{\ell})$ as mentioned above. We also consider matrix multiplication over the binary field and show that its quantum communication complexity (and actually even its classical communication complexity) is $\widetilde{O}(n\sqrt{\ell})$ (Theorem 8). We give a matching lower bound as well (Theorem 11).

These bounds are also interesting since they confirm and substantiate our current understanding of the power of quantum algorithms for problems related to matrix multiplication. Indeed, while matrix multiplication over a field seems harder than Boolean matrix multiplication for quantum computers, we currently do not have any technique to prove such a statement in the time complexity setting. Our results prove this statement in the communication complexity setting, for instances with sparse output matrices.

In addition to these concrete results, this work presents several interesting new open problems. An *OR lemma* is a composition lemma that says that the quantum communication complexity of the function $f^{\vee m}(a_1, \ldots, a_m, b_1, \ldots, b_m) = \bigvee_{i=1}^{m} f(a_i, b_i)$ is at least $\Omega(\sqrt{m}Q(f))$, where $Q(f)$ is the quantum query complexity of f. We show that our upper bound for composition join is tight up to logarithmic factors assuming the problem of Boolean matrix multiplication satisfies an OR lemma (Proposition 14). We give further evidence that our upper bound is indeed tight by showing that it is tight at extreme values of ℓ, when $\ell = O(1)$ (Proposition 12) and when $\ell = \Omega(n^2)$ (Proposition 13).

We believe that proving lower bounds on set joins is a very interesting area of future research, as doing so may give insight into direct product theorems in communication complexity, as well as lower bounds in quantum query complexity for problems that involve *read-many formulas*, in which different parts of the input are used multiple times, which makes it difficult to prove lower bounds using standard composition theorems.

Organization

The remainder of this paper is organized as follows. In Section 2, we give the necessary preliminaries, including quantum communication complexity, and the groundwork for studying the quantum communication complexity of set joins. In Section 3, we present our communication protocol for composition join. In Section 4, we present our classical communication protocol and matching quantum lower bound for matrix multiplication over $\mathbb{F}_2$. Finally, in Section 5, we give some evidence that our upper bound for composition join is tight, by reducing a matching lower bound to a plausible OR lemma.

2 Preliminaries

2.1 Notation

Let $2^{[n]}$ denote the set of subsets of $[n]$. A subset S of $[n] := \{1, \dots, n\}$ can be represented by an n-bit string, and we will sometimes conflate these two notions. Let $S[i]$ denote the i-th bit of the string corresponding to S, so $S[i] = 1$ if and only if $i \in S$. For any $x \in \{0,1\}^n$, we let $|x|$ denote the Hamming weight, which is the size of the corresponding subset of $[n]$. Similarly, for a Boolean matrix A (i.e., a matrix with entries in $\{0,1\}$), let $|A|$ denote the number of 1s in A. Given an $m \times n$ Boolean matrix A and an $n \times m$ Boolean matrix B, we write the Boolean product $A * B$ and let AB denote their product over the finite field $\mathbb{F}_2 = \{0,1\}$. Morever, for any integer $k \in [n]$ we let $A[\cdot, k]$ denote the k-th column of A and $B[k, \cdot]$ the k-th row of B.

2.2 Quantum Communication Complexity

A communication problem is a function $f : A \times B \to Y$ whose input has two parts, $a \in A$, which we call Alice's input, and $b \in B$, which we call Bob's input. In the model of communication complexity, first defined by Yao [25], Alice and Bob want to run a protocol such that, at the end of the protocol, Alice and Bob both output $f(a, b)$ with high probability, and they want to minimize the number of bits they need to communicate in order to achieve this.

In the model of quantum communication, also introduced by Yao [26], Alice and Bob are allowed a quantum communication channel (for a detailed introduction to the theory of quantum information, see [24]) and they want to minimize the number of quantum bits (qubits) they need to communicate in order to compute the function. More precisely, a quantum communication protocol consists of finite inner product spaces $\mathcal{X}$ and $\mathcal{Y}$, a measurement $\{\Pi, I - \Pi\}$ on $\mathcal{Y}$, and unitary operators $\{U_i\}_{i=1}^T$, such that for odd i, U_i acts on $\mathcal{X} \otimes \mathcal{C}$, where $\mathcal{C} = \mathbb{C}^2$ is a single-qubit system, and for even i, U_i acts on $\mathcal{C} \otimes \mathcal{Y}$. The protocol is said to have quantum communication complexity T. The protocol is said to compute f with bounded error $1/3$ if for all $(a, b) \in A \times B$, there exist states ρ_a and ρ_b on $\mathcal{X}$ and $\mathcal{Y}$ respectively, such that

$$|\operatorname{Tr}\left((I_{\mathcal{X}} \otimes I_{\mathcal{C}} \otimes \Pi) U_T \dots U_1 (\rho_a \otimes |0\rangle\langle 0| \otimes \rho_b)\right) - f(a, b)| \leq 1/3.$$

That is, Alice begins the protocol in some state ρ_a depending on her input, and Bob begins the protocol in some state ρ_b depending on his input, and Alice also has an additional system, $\mathcal{C}$, initialized to $|0\rangle$,which will be used for communication with Bob. Alice applies U_1 to $\mathcal{X}\otimes\mathcal{C}$, and then sends $\mathcal{C}$ to Bob, who applies U_2 to $\mathcal{C}\otimes\mathcal{Y}$, before sending $\mathcal{C}$ back to Alice. They continue until they have applied all T unitaries, at which point, Bob measures $\mathcal{Y}$, and the outcome determines $f(a,b)$ with error at most $1/3$.

The *bounded error quantum communication complexity* of f, denoted $Q(f)$, is the minimum T such that there exists a quantum communication protocol computing f with bounded error $1/3$ with quantum communication complexity T. We will also consider the bounded error quantum communication complexity of partial functions $f : D \to \{0,1\}$ for $D \subseteq A \times B$.

There are many variants of this model, including the setting of *one-way* communication complexity, in which Alice can send messages to Bob, but Bob cannot send messages to Alice, and only Bob is required to output the correct answer. We let $Q^1(f)$ denote the one-way communication complexity of f.

An important problem in the study of quantum communication complexity is the problem of set disjointness, which is defined as follows.

Set Disjointness, DISJ_n
Alice's input: $a \in \{0,1\}^n$
Bob's input: $b \in \{0,1\}^n$
Output: $\mathsf{DISJ}_n(a,b) = \bigvee_{i=1}^n a_i b_i$

It is well known that $Q(\mathsf{DISJ}_n) = \Theta(\sqrt{n})$ [4, 10, 1, 21], beating the classical communication complexity of $\Theta(n)$ [12, 20]. When one of the two input sets is small, we can do even better as shown in the following elementary lemma.

▶ **Lemma 1** (Set disjointness for small sets). *The bounded error quantum communication complexity of* $\mathsf{DISJ}_n(a,b)$ *is* $O\left(\sqrt{\frac{\min\{|a|,|b|\}}{|a\cap b|+1}}\log n\right)$. *Furthermore, if* $\mathsf{DISJ}_n(a,b)=1$*, then the protocol also returns a uniform random* $i \in a\cap b$.

Proof. To begin the protocol, Alice sends Bob $|a|$ using $\lceil \log_2 n\rceil$ bits of communication, and Bob sends Alice $|b|$ using $\lceil \log_2 n\rceil$ bits of communication. If $|a|<|b|$, Alice and Bob perform Grover search on the set $S_A = \{i \in [n] : a_i = 1\}$ for an index $i \in S_A$ such that $f_B(i)=1$, where $f_B(i) = b_i$. They do this as follows. Alice computes $|\pi(S_A)\rangle = \sum_{i\in S_A}\frac{1}{\sqrt{|a|}}|i\rangle$. In order to perform the search, Alice and Bob must alternate $R_A = 2|\pi(S_A)\rangle\langle\pi(S_A)| - I$ and $R_B = \sum_{i\in[n]}(-1)^{b_i}|i\rangle\langle i|$, $O\left(\sqrt{\frac{|a|}{|a\cap b|+1}}\right)$ times. Clearly Alice can implement R_A, and Bob can implement R_B, so they can implement this algorithm using $O\left(\sqrt{\frac{|a|}{|a\cap b|+1}}\right)$ rounds of communication, each time communicating a $\lceil \log_2 n\rceil$-qubit state. Bob measures some $i\in[n]$, and sends $i, f_B(i)$ to Alice. Both Alice and Bob output $f_B(i)$. If $|a| \geq |b|$, they do the protocol obtained by reversing Alice and Bob's roles. ◀

The algorithm in Lemma 1 actually finds a witness $i \in a\cap b$, which is slightly stronger than what is required to solve DISJ. We will also consider the problem of finding the entire intersection:

Find-all Set Intersection, $\mathsf{DISJall}_n$
Alice's input: $a \in \{0,1\}^n$
Bob's input: $b \in \{0,1\}^n$
Output: $\mathsf{DISJall}_n(a,b) = \{i\in[n] : a_i = b_i = 1\}$

In this case, we also have an advantage when a or b is small, as shown in the following lemma.

▶ **Lemma 2** (Find-all set intersection for small sets). *The bounded error quantum communication complexity of* $\mathsf{DISJall}_n(a,b)$ *is* $O(\sqrt{|a\cap b|\min\{|a|,|b|\}}\log n)$.

Proof. Alice and Bob run the following protocol.

1. $S \leftarrow \emptyset$, $\tilde{a} \leftarrow a$, $\tilde{b} \leftarrow b$.
2. Repeat:
   - **a.** Use the protocol for $\mathsf{DISJ}_n(\tilde{a},\tilde{b})$ to obtain $i \in \tilde{a}\cap\tilde{b}$. If $\mathsf{DISJ}_n(\tilde{a},\tilde{b}) = 0$, output S.
   - **b.** $S \leftarrow S\cup\{i\}$, $\tilde{a}_i \leftarrow 0$, $\tilde{b}_i \leftarrow 0$.

This protocol has communication complexity

$$\sum_{i=1}^{|a\cap b|}\sqrt{\frac{\min\{|a|,|b|\}}{|a\cap b|-i+1}}\log n = \Theta\left(\sqrt{|a\cap b|\min\{|a|,|b|\}}\log n\right)$$

qubits. ◀

2.3 Set Joins and Direct Product Theorems

In this paper, we consider various *set join* problems. For any predicate $\mathcal{P}_n : 2^{[n]}\times 2^{[n]} \to \{0,1\}$, we can define a set join, as follows.

$\mathcal{P}$-Set Join, $\mathcal{P}_n^{\otimes m}$

Alice's input: $\mathcal{A} = (A_1,\dots,A_m)$, $A_i \subseteq [n]$

Bob's input: $\mathcal{B} = (B_1,\dots,B_m)$, $B_i \subseteq [n]$

Output: $\{(i,j)\in[m]\times[m] : \mathcal{P}_n(A_i,B_j) = 1\}$

When $\mathcal{P}_n$ is the predicate such that $\mathcal{P}_n(A,B) = 1$ if and only if $A\cap B \neq \emptyset$, the resulting join is called the *composition join* or sometimes *set-intersection-non-empty join*. As mentioned in the introduction, this join is equivalent to Boolean matrix multiplication, where we consider $A_1,\dots,A_m$ to be the rows of a matrix $A\in\{0,1\}^{m\times n}$, and $B_1,\dots,B_m$ to be the columns of a matrix $B\in\{0,1\}^{n\times m}$.

Consider a related construction: the direct product.

Direct product, $\mathcal{P}_n^{(m)}$

Alice's input: $\mathcal{A} = (A_1,\dots,A_m)$, $A_i \subseteq [n]$

Bob's input: $\mathcal{B} = (B_1,\dots,B_m)$, $B_i \subseteq [n]$

Output: $\{i\in[m] : \mathcal{P}_n(A_i,B_i)\}$

Unlike set joins, such problems are well-studied, and much is known. Clearly, we have $Q(\mathcal{P}_n^{(m)}) = O(m\mathcal{P}_n\log m)$ for any predicate $\mathcal{P}_n$. Intuitively, one can usually expect that the resources needed to solve m instances of $\mathcal{P}_n$ scale as at least m times the resources needed to solve one instance, that is: $Q(\mathcal{P}_n^{(m)}) = \Omega(mQ(\mathcal{P}_n))$. This is called a *(weak) direct product theorem* for $\mathcal{P}_n$. In fact, we can sometimes prove a stronger statement: that even solving $\mathcal{P}_n^{(m)}$ with success probability 2^{-m} requires $\Omega(mQ(\mathcal{P}_n))$ quantum communication. Such a statement is called a *strong direct product theorem*. Although such a statement likely holds for many problems in quantum communication complexity, it can be very difficult to prove (see, e.g., [22] and the references therein).

In the case of set joins, it is also easy to see that $Q(\mathcal{P}_n^{\otimes m}) = O(m^2Q(\mathcal{P}_n)\log m)$, however, unlike the case of direct products, this naive upper bound is often not tight. For example, let $Q^1(\mathcal{P}_n)$ denote the *one-way communication complexity* of $\mathcal{P}_n$. Then we have the following:

▶ **Theorem 3.** *For any predicate $\mathcal{P}_n$, $Q(\mathcal{P}_n^{\otimes m}) \leq O(mQ^1(\mathcal{P}_n)\log m)$.*

Proof. Consider an optimal one-way quantum communication protocol for $\mathcal{P}_n$. Let $\rho(A)$ be the mixed state on at most $Q^1(\mathcal{P}_n)$ qubits that Alice sends Bob and let $U(B)$ be the unitary that Bob applies to $\rho(A) \otimes |0\rangle\langle 0|_{\mathcal{W}} \otimes |0\rangle\langle 0|_{\mathcal{A}}$, for some workspace $\mathcal{W}$ and single-qubit answer register $\mathcal{A}$, so that he measures $\mathcal{P}_n(A, B)$ in the answer register with probability at least $2/3$.

We construct a (one-way) protocol for $\mathcal{P}_n^{\otimes m}$ as follows. Let Alice have input $A_1, \dots, A_m$, and Bob $B_1, \dots, B_m$. For every $i \in [m]$, Alice sends Bob $(\rho(A_i))^{\otimes c \log m}$, where c is a large enough constant. For each $i, j \in [m]$, Bob applies $U(B_j)^{\otimes c \log m}$ to $(\rho(A_i) \otimes |0\rangle\langle 0|_{\mathcal{W}} \otimes |0\rangle\langle 0|_{\mathcal{A}})^{\otimes c \log m}$. He then computes the majority of the answer registers in a new single-qubit register, which he measures. Let $\rho(A_i, B_j) := U(B_j)(\rho(A_i) \otimes |0\rangle\langle 0|_{\mathcal{W}} \otimes |0\rangle\langle 0|_{\mathcal{A}})U(B_j)^\dagger = \sum_{b,b' \in \{0,1\}} \rho_{b,b'} \otimes |b\rangle\langle b'|$, so the state Bob measures is (up to permuting registers):

$$\sum_{x,x' \in \{0,1\}^\ell} \bigotimes_{i=1}^{\ell} \rho_{x_i, x_i'} \otimes |x\rangle\langle x'| \otimes |\mathrm{maj}(x)\rangle\langle \mathrm{maj}(x')|,$$

where $\mathrm{maj}(x) = 1$ if $|x| \geq \ell/2$ and 0 otherwise. Assume that $\mathcal{P}_n(A_i, B_j) = 1$, as the 0 case is nearly identical. Then the probability of success in a single round is $\mathrm{Tr}(\rho_{1,1}) \geq 2/3$, so the probability of success upon measuring the majority register is:

$$\sum_{\substack{x \in \{0,1\}^\ell: \\ |x| \geq \ell/2}} \Pi_{i=1}^{\ell} \mathrm{Tr}(\rho_{x_i,x_i}) = \sum_{k=0}^{\lfloor \ell/2 \rfloor} \binom{\ell}{k} \mathrm{Tr}(\rho_{1,1})^k (1 - \mathrm{Tr}(\rho_{1,1}))^{\ell-k} \geq 1 - e^{-\Omega(\ell)} = 1 - m^{-\Omega(1)},$$

where the inequality follows from Hoeffding's inequality, and the constant in $\Omega(1)$ depends on c. Thus, Bob gets the correct answer with high probability, but furthermore, this measurement causes negligible damage to the state $\rho(A_i, B_j)^{\otimes \ell}$, so Bob can apply $(U(B_j)^\dagger)^{\otimes \ell}$ to recover $\rho(A_i)^{\otimes \ell}$, to be used again. The error in the state remains negligible as long as Bob does this no more than $m^{O(1)}$ times. ◀

Call a theorem of the form $Q(\mathcal{P}_n^{\otimes m}) = \Omega(\min\{mn, m^2 Q(\mathcal{P}_n)\})$ a *(weak) all-pairs product theorem*. The $\min\{mn, \cdot\}$ is to account for the fact that we always have a trivial upper bound of mn, and so if we did not include this, the statement would always be false for some values of m and n. In this work, we give an example of a set-join for which an all-pairs direct product theorem does not hold — in particular, in Section 3 we will give an upper bound of $O(m^{3/2}\sqrt{n})$ for the composition join, showing that this problem does not satisfy an all-pairs product theorem. Although we show that such a statement holds for matrix multiplication over $\mathbb{F}_2$, in that case, we have $\min\{mn, m^2 Q(\mathcal{P}_n)\} = mn$ for all m and n, so the best strategy is always for Alice to send her whole input to Bob, rather than for Alice and Bob to compute m^2 instances of $\mathcal{P}_n$. It is an open question whether or not there exists a predicate for which an all-pairs product theorem holds in a non-trivial sense — that is, the best strategy is to compute m^2 instances of $\mathcal{P}_n$.

3 Composition Join (Boolean Matrix Multiplication)

In this section, we give an upper bound on the communication complexity of Boolean matrix multiplication (equivalent to computing the composition join), proving our main theorem. As in [23], we consider the following promise version of the problem, in which the output has at most ℓ ones.

Boolean Matrix Multiplication, $\mathsf{BMM}_{m,n,\ell}$

Alice's input: $A \in \{0,1\}^{m\times n}$
Bob's input: $B \in \{0,1\}^{n\times m}$
Promise: $|A * B| \leq \ell$
Output: $\mathsf{BMM}_{m,n,\ell}(A,B) = A*B = \{(i,j)\in[m]\times[m] : \exists k\in[n], A[i,k]=B[k,j]=1\}$

The communication protocol we give is inspired by the query-optimal quantum algorithm for Boolean matrix multiplication given in [11]. The algorithm of [11] is based on a subroutine for a problem called *graph collision*. For any family of bipartite graphs G on n vertices, the communication version of graph collision on G is as follows.

Graph Collision, GC_G

Alice's input: $f_A \in \{0,1\}^n$
Bob's input: $f_B \in \{0,1\}^n$
Output: $\mathsf{GC}_G(f_A,f_B) = \bigvee_{(i,j)\in G} f_A(i)f_B(j)$

An efficient protocol for this problem can easily be constructed in the communication complexity setting:

▶ **Lemma 4** (Graph collision). *$Q(\mathsf{GC}_G(f_A,f_B)) = O(\sqrt{\min\{|f_A|,|f_B|\}})$ for any family of bipartite graphs G.*

Proof. Alice sends Bob $|f_A|$, and Bob sends Alice $|f_B|$ using $2\log n$ bits of communication. If $|f_A|\leq|f_B|$, Alice sets $a=f_A$ and Bob sets $b=\{i\in[n] : \exists j\in[n],(i,j)\in G, f_B(j)=1\}$. Otherwise, Alice sets $a=\{j\in[n] : \exists i\in[n],(i,j)\in G, f_A(i)=1\}$ and Bob sets $b=f_B$. They finally compute $\mathsf{DISJ}(a,b)$. ◀

When we solve graph collision as a subroutine, we will actually want to additionally find *all* graph collisions in a particular instance. That is, we will want to solve the following problem.

Find All Graph Collisions, GCall_G

Alice's input: $f_A \in \{0,1\}^n$
Bob's input: $f_B \in \{0,1\}^n$
Output: $\mathsf{GCall}_G(f_A,f_B) = \{(i,j)\in G : f_A(i)=f_B(j)=1\}$

The following upper bound for GCall_G is a corollary of the previous lemma (its proof is similar to the proof of Lemma 2).

▶ **Corollary 5** (Find all graph collisions). *$Q(\mathsf{GCall}_G(f_A,f_B)) = O(\sqrt{\lambda\min\{|f_A|,|f_B|\}})$, where $\lambda = |\{(i,j)\in G : f_A(i)=f_B(j)=1\}|$.*

The final ingredient we need before presenting our quantum communication protocol for Boolean matrix multiplication is a quantum communication protocol that searches for a 1-instance among n independent instances of a communication problem. Its proof is fairly straightforward and simply combines quantum search with the original communication protocol.

▶ **Lemma 6** (Search over communication instances). *Let $f : X\times Y\to\{0,1\}$ be a communication problem with bounded error quantum communication complexity $Q(f)$. Let $F : X^n\times Y^n\to\{0,1\}$ be the problem of finding some $i\in[n]$ such that $f(x_i,y_i)=1$. Then $Q(F)=O(\sqrt{\frac{n}{t}}Q(f)\log n)$, where $t=|\{i\in[n] : f(x_i,y_i)=1\}|$.*

Proof. Alice creates $|\pi\rangle = \sum_{i\in[n]} \frac{1}{\sqrt{n}}|i\rangle$. Alice and Bob implement quantum search by repeating the reflections

$$R_1 = 2|\pi\rangle\langle\pi| - I \quad \text{and} \quad R_2 = \sum_{i\in[n]} (-1)^{f(x_i,y_i)}|i\rangle\langle i|$$

$O(\sqrt{n/t})$ times. Each implementation of R_2 is accomplished as follows. Let Alice's state be $\sum_{i\in[n]} \alpha_i|i\rangle$. Alice performs the mapping $|i\rangle \mapsto |i,i\rangle$, to get $\sum_{i\in[n]} \alpha_i|i,i\rangle$ and sends half of the state to Bob. Conditioned on their quantum state, Alice and Bob perform the protocol for f using input (x_i, y_i), that is, they perform the protocol on a superposition of inputs. This leaves the state $\sum_{i\in[n]} \alpha_i|i, f(x_i, y_i)\rangle_A|i, f(x_i, y_i)\rangle_B$ (here we assume, without loss of generality, that the final state in the protocol for f does not contain any garbage). Alice then maps this state to $\sum_i(-1)^{f(x_i,y_i)}|i, f(x_i, y_i)\rangle_A|i, f(x_i, y_i)\rangle_B$. They run the protocol in reverse to uncompute $f(x_i, y_i)$, leaving $\sum_i \alpha_i(-1)^{f(x_i,y_i)}|i\rangle_A|i\rangle_B$. Bob sends his half to Alice, so she can uncompute it, leaving the state $\sum_i \alpha_i(-1)^{f(x_i,y_i)}|i\rangle$, and thus implementing R_2. ◀

We are now ready to state and prove our main theorem.

▶ **Theorem 7** (Upper bound for Boolean matrix multiplication). *For all $\ell \in \{1, \ldots, m^2\}$,*

$$Q(\mathsf{BMM}_{m,n,\ell}) = O(\sqrt{n}\ell^{3/4} \log m).$$

Proof. Alice and Bob run the following communication protocol.

1. Alice and Bob individually store the all-zero matrix C of size $m \times m$.
2. Repeat:
   a. Alice and Bob jointly find $k \in [n]$ such that $\mathsf{GC}_{\overline{C}}(A[\cdot, k], B[k, \cdot]) = 1$. If none exists, Alice and Bob output C.
   b. Alice and Bob jointly compute $S \leftarrow \mathsf{GCall}_{\overline{C}}(A[\cdot, k], B[k, \cdot])$.
   c. Alice and Bob individually compute $C \leftarrow C + S$.

In this protocol Alice and Bob each maintain a matrix C containing the 1s of the product $A * B$ found by the protocol so far. They repeatedly search for a new $k \in [n]$ such that $f^k_A = A[\cdot, k]$ and $f^k_B = B[k, \cdot]$ have graph collisions with respect to the graph given by the complement of C. When they find such a k, they compute all graph collisions. Suppose they find $\{k_1, \ldots, k_t\}$ before there are no more k to be found, and let λ_i be the number of ones found at round i. By Lemma 4 and Lemma 6, in round i step 2a costs $O\left(\sqrt{\frac{n}{t-i+1}\min\{|f^{k_i}_A|, f^{k_i}_B|\}\log m}\right)$. By Corollary 5, in round i step 2b costs $O\left(\sqrt{\lambda_i \min\{|f^{k_i}_A|, |f^{k_i}_B|\}\log m}\right)$. Thus, the total cost is at most:

$$\sum_{i=1}^{t}\left(\sqrt{\frac{n}{t-i+1}}\sqrt{\min\{|f^{k_i}_A|, |f^{k_i}_B|\}} + \sqrt{\lambda_i \min\{|f^{k_i}_A|, f^{k_i}_B|\}}\right)\log m.$$

Note that for any k, every (i, j) such that $f^k_A(i) = f^k_B(j) = 1$ implies that $(A * B)[i, j] = 1$, so we necessarily have $\ell \geq |f^k_A| \cdot |f^k_B|$. We therefore have $\min\{|f^k_A|, |f^k_B|\} \leq \sqrt{\ell}$ for all $k \in [n]$, and thus, the total cost is at most (up to constants):

$$\begin{aligned}\ell^{1/4}\sum_{i=1}^{t}\left(\sqrt{\frac{n}{t-i+1}} + \sqrt{\lambda_i}\right)\log m \quad &\leq \quad \ell^{1/4}\left(\sqrt{nt} + \sqrt{t\sum_{i=1}^{t}\lambda_i}\right)\log m \\ &\leq \quad \ell^{1/4}(\sqrt{nt} + \sqrt{t\ell})\log m,\end{aligned}$$

where in the first line we use the fact that $\sum_{i=1}^{t} i^{-1/2} = \Theta(\sqrt{t})$ and Cauchy-Schwartz inequality, and in the second line we use the fact that $\sum_{i=1}^{t} \lambda_i = \ell$, since ℓ is the total number of ones we find over all rounds. Finally, observe that since t is the number of distinct witnesses $k \in [n]$ found, $t \leq n$, and since we find at least one new 1 in every round except the last, we also have $t \leq \ell + 1$. Thus, the total communication is at most

$$(\ell^{1/4}\sqrt{n} + \ell^{3/4})\sqrt{\min\{n,\ell\}}\log m = O\left(\ell^{3/4}\sqrt{n}\log m\right),$$

as claimed. ◀

4 Matrix Multiplication over Finite Fields

In this section we consider matrix multiplication over finite fields and give tight bounds (up to possible polylogarithmic factors) on its communication complexity. We work out here only the case of square matrices over the binary field. Formally, the problem we consider is the following.

Square matrix multiplication over $\mathbb{F}_2$, $\mathsf{MM}_{n,\ell}$
Alice's input: $A \in \mathbb{F}_2^{n\times n}$
Bob's input: $B \in \mathbb{F}_2^{n\times n}$
Promise: $|AB| \leq \ell$
Output: the matrix $AB \in \mathbb{F}_2^{n\times n}$

The main result of this section is the following upper bound on the classical (and thus quantum) communication complexity of this problem.

▶ **Theorem 8** (Upper bound for matrix multiplication over $\mathbb{F}_2$). *The classical communication complexity of $\mathsf{MM}_{n,\ell}$ is $\widetilde{O}(n\sqrt{\ell})$.*

We will need two lemmas to prove Theorem 8. The first lemma is a finite-field version of a result related to compressed sensing used in [23]. The proof of this finite-field version can be found in [7].

▶ **Lemma 9.** *For any positive integer n and any integer $\kappa \in \{1,\dots,n\}$, there are a distribution on random matrices $M \in \mathbb{F}_2^{O(\kappa)\times n}$ and a reconstruction function $\mathsf{Rec}(\cdot)$ such that for any vector $x \in \mathbb{F}_2^n$ with at most κ non-zero entries the inequality*

$$\Pr_M\left[\mathsf{Rec}(Mx) = x\right] > 0.99.$$

holds (i.e., $\mathsf{Rec}(\cdot)$ applied on Mx returns x with high probability).

The second lemma shows how to use Freivalds' technique to detect non-zero columns of a matrix product. Similar ideas were used in [8].

▶ **Lemma 10.** *Let m and n be two positive integers. Consider the setting where Alice has for input a matrix $A \in \mathbb{F}_2^{m\times n}$ and Bob has for input a matrix $B \in \mathbb{F}_2^{n\times n}$. Alice and Bob can detect, with high probability, which columns of AB contain at least one non-zero entry with $\widetilde{O}(n)$ communication.*

Proof. Consider the following procedure: Alice takes a vector v uniformly at random in $\mathbb{F}_2^m$; Alice sends the row-vector $v^T A \in \mathbb{F}_2^n$ to Bob; Bob sends the row-vector $v^T AB \in \mathbb{F}_2^n$ to Alice. This procedure has communication complexity $2n$ and, for each column of AB, enables Alice and Bob to decide with probability at least 1/2 whether this column contains at least one non-zero entry. By repeating this procedure a logarithmic number of times, Alice and Bob are able to find, with high probability, which columns of AB contain at least one non-zero entry. ◀

We are now ready to prove Theorem 8.

Proof of Theorem 8. We assume for convenience that both $\sqrt{\ell}$ and $n/\sqrt{\ell}$ are integers (the general case is handled similarly). We will say that a column of AB is dense if it contains at least $0.9\sqrt{\ell}$ non-zero entries, and say that a column of AB is sparse if it contains at most $1.1\sqrt{\ell}$ non-zero entries (note that a column can be both sparse and dense). The protocol is as follows.

1. Alice and Bob partition the columns of AB into dense columns and sparse columns: they compute a set of indexes $S \subseteq \{1, \dots, n\}$ such that, for any $j \in \{1, \dots, n\}$, the j-th column of AB is dense if $j \in S$ and sparse if $j \notin S$.
2. Alice and Bob compute all entries of all columns of AB with index in S.
3. Alice and Bob compute all entries of all the columns of AB with index in $[n] \setminus S$.

Step 1 can be done probabilistically with $\widetilde{O}(n)$ bits of communication by repeating the following procedure: Alice constructs a $(n/\sqrt{\ell}) \times n$ matrix A' by selecting $n/\sqrt{\ell}$ rows of A uniformly at random; Alice and Bob then use the protocol of Lemma 10 (with A' as Alice's input and B as Bob's input) to decide which columns of $A'B$ have more than one non-zero entry. Repeating this procedure a logarithmic number of times enables Alice and Bob to decide, with high probability, which columns of AB are not dense: for a non-dense column of AB (i.e., a column with less than $0.9\sqrt{\ell}$ non-zero entries) the corresponding column of $A'B$ will not contain any non-zero entry with high probability (on the choice of A'). The indices of the other columns are collected in S. The indices in S thus correspond only to dense columns of AB. While the set S may not contain the indices of all the dense columns of AB, it can be seen from a similar argument that all non-sparse columns of AB (i.e., the columns with at least $1.1\sqrt{\ell}$ non-zero entries) will be put in S, which means that all indices in $[n] \setminus S$ correspond to columns of AB that are sparse.

Step 2 can be done with $O(|S|n) = O(\sqrt{\ell}n)$ bits of communication (note that $|S| \leq \frac{1}{0.9}\sqrt{\ell}$ since AB has only at most ℓ non-zero entries): Bob simply sends the entries $B[i,j]$ for all $(i,j) \in \{1, \dots, n\} \times S$, and then Alice computes $AB[i,j]$ for all $(i,j) \in \{1, \dots, n\} \times S$.

Step 3 can be done with $\widetilde{O}(n\sqrt{\ell})$ bits of communication using Lemma 9 with $\kappa = \lceil 1.1\sqrt{\ell} \rceil$, by repeating the following procedure a logarithmic number of times: Alice chooses a random matrix M as in Lemma 9 and sends MA to Bob; for each $j \in \{1, \dots, n\} \setminus S$, Bob computes $\mathsf{Rec}(MAz)$ where z denotes the j-th column of B. ◀

We now show a lower bound on the quantum (and thus also classical) communication complexity of matrix multiplication over $\mathbb{F}_2$, which matches the upper bound of Theorem 8 up to polylogarithmic factors.

▶ **Theorem 11.** *The quantum communication complexity of* $\mathsf{MM}_{n,\ell}$ *is* $\Omega(n\sqrt{\ell})$.

Proof. Assume for convenience that $\sqrt{\ell}$ is an integer (the general case is handled similarly). Let $x^1, \dots, x^{\sqrt{\ell}}, y^1, \dots, y^{\sqrt{\ell}}$ be $2\sqrt{\ell}$ vectors in $\mathbb{F}_2^n$. Let $x \in \mathbb{F}_2^{n\sqrt{\ell}}$ be the vector obtained by concatenating $x^1, \dots, x^{\sqrt{\ell}}$, and $y \in \mathbb{F}_2^{n\sqrt{\ell}}$ be the vector obtained by concatenating $y^1, \dots, y^{\sqrt{\ell}}$.

Construct the $n \times n$ matrix A by putting the vector x^i as its i-th row, for each $i \in \{1, \dots, \sqrt{\ell}\}$, and setting the next $n - \sqrt{\ell}$ rows to zero (observe that $\sqrt{\ell} \leq n$ since $\ell \leq n^2$). Construct the $n \times n$ matrix B by putting the vector y^j as its j-th column, for each $j \in$

$\{1, \dots, \sqrt{\ell}\}$, and setting the next $n - \sqrt{\ell}$ columns to zero. Observe that $|AB| \leq \ell$ and the parity of the diagonal entries of the matrix product AB is equal to

$$\bigoplus_{i=1}^{\sqrt{\ell}} x^i \cdot y^i = x \cdot y.$$

We thus obtain a reduction from computing the inner product of two vectors in $\mathbb{F}_2^{n\sqrt{\ell}}$ to solving $\mathsf{MM}_{n,\ell}$. Since the quantum communication complexity of the former problem is $\Omega(n\sqrt{\ell})$, as shown in [14], we obtain the same lower bound for $\mathsf{MM}_{n,\ell}$. ◀

5 Lower Bounds for Boolean Matrix Multiplication

An important open problem of this work is to prove a tight lower bound on the bounded error quantum communication complexity of Boolean matrix multiplication, i.e., to show that the upper bound of Theorem 7 is tight. Let us focus on the square case (i.e., $m = n$). We are able to prove two lower bounds, each of which is tight for one extreme value of ℓ: $\ell = O(1)$ or $\ell = \Omega(n^2)$, but neither is tight for the range $\ell \in (\omega(1), o(n^2))$. We further show that assuming a plausible OR-lemma, our upper bound is indeed tight, up to logarithmic factors.

▶ **Proposition 12.** *For all $\ell \in \{1, \dots, n^2\}$, $Q(\mathsf{BMM}_{n,n,\ell}) = \Omega(\sqrt{n\ell})$. In particular, when $\ell = O(1)$, then $Q(\mathsf{BMM}_{n,n,\ell}) = \Omega(\sqrt{n}\ell^{3/4})$.*

Proof. We can embed $\sqrt{\ell} \leq n$ instances $\{(a^{(i)}, b^{(i)})\}_{i=1}^{\sqrt{\ell}}$ of DISJ_n in an instance of $\mathsf{BMM}_{n,n,\ell}$ as follows. Let A have $a^{(i)}$ in the i-th row for $i = 1, \dots, \sqrt{\ell}$, and all zeros elsewhere, and let B have $b^{(i)}$ in the i-th column for $i = 1, \dots, \sqrt{\ell}$, and zeros elsewhere. Then AB is 0 except in the upper left $\sqrt{\ell} \times \sqrt{\ell}$ submatrix, so $|AB| \leq \ell$, and $(AB)[i,i] = \mathsf{DISJ}_n(a^{(i)}, b^{(i)})$, so AB encodes $\mathsf{DISJ}_n^{(\sqrt{\ell})}(a^{(1)}, \dots, a^{(\sqrt{\ell})}, b^{(1)}, \dots, b^{(\sqrt{\ell})})$. The result follows from the $\Omega(\sqrt{\ell}\sqrt{n})$ lower bound on $Q(\mathsf{DISJ}_n^{(\sqrt{\ell})})$ shown in [13]. ◀

▶ **Proposition 13.** *For all $\ell \in \{1, \dots, n^2\}$, $Q(\mathsf{BMM}_{n,n,\ell}) = \Omega(\ell)$. In particular, when $\ell = \Omega(n^2)$, then $Q(\mathsf{BMM}_{n,n,\ell}) = \Omega(\ell^{3/4}\sqrt{n})$.*

Proof. We can embed an instance (a, b) of the inner product function IP_ℓ in an instance of $\mathsf{BMM}_{n,n,\ell}$ as follows. Let $B = I$ be the identity matrix, and let A contain the ℓ-bit string a in the first ℓ positions. Then Alice and Bob jointly compute $AB = A$, and Bob can compute $\mathsf{IP}(a, b)$ and send the resulting bit to Alice. Since $Q(\mathsf{IP}_\ell) = \Omega(\ell)$, we have $Q(\mathsf{BMM}_{n,n,\ell}) = \Omega(\ell)$. ◀

▶ **Proposition 14.** *Suppose computing the entrywise-OR of k independent instances of BMM_{n,n,n^2} has bounded error quantum communication complexity $\Omega(\sqrt{k}Q(\mathsf{BMM}_{n,n,n^2}))$. Then for any $\ell \in [n^2]$, $Q(\mathsf{BMM}_{n,n,\ell}) = \Omega(\ell^{3/4}\sqrt{n})$.*

Proof. Let $(A_1, B_1), \dots, (A_k, B_k)$ be independent instances of $\mathsf{BMM}_{\sqrt{\ell},\sqrt{\ell},\ell}$, for $k = \frac{n}{\sqrt{\ell}}$. Define A and B as follows:

$$A = \begin{bmatrix} A_1 & \dots & A_k \\ 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} B_1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ B_k & 0 & \dots & 0 \end{bmatrix}.$$

Then $A * B$ has $\bigvee_{i=1}^{k} A_i * B_i$ in the top left corner, and zeros elsewhere. So $|A * B| \leq \ell$, and computing $A * B$ costs at least $\sqrt{k}Q(\mathsf{BMM}_{\sqrt{\ell},\sqrt{\ell},\ell}) \geq \sqrt{\frac{n}{\sqrt{\ell}}}\ell = \ell^{3/4}\sqrt{n}$. ◀

The above proposition actually holds equally true for $\mathsf{BMM}_{m,n} = \mathsf{BMM}_{m,n,m^2}$, the non-promise problem, and would imply $Q(\mathsf{BMM}_{m,n}) = \Omega(m^{3/2}\sqrt{n})$.

Acknowledgments. The authors are grateful to Troy Lee and Ronald de Wolf for helpful discussions. FLG is supported by the Grant-in-Aid for Young Scientists (A) No. 16H05853 and the Grant-in-Aid for Scientific Research (A) No. 16H01705 of the Japan Society for the Promotion of Science, and the Grant-in-Aid for Scientific Research on Innovative Areas No. 24106009 of the Ministry of Education, Culture, Sports, Science and Technology in Japan. SJ is supported by the Institute for Quantum Information and Matter, an NSF Physics Frontiers Center (NFS Grant PHY-1125565) with support of the Gordon and Betty Moore Foundation (GBMF-12500028).

References

1 S. Aaronson and A. Ambainis. Quantum search of spatial regions. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science*, pages 200–209, 2003.

2 R. R. Amossen and R. Pagh. Faster join-projects and sparse matrix multiplications. In *Proceedings of the International Conference on Database Theory*, pages 121–126, 2009.

3 A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 918–929, 2006.

4 H. Buhrman, R. Cleve, and A. Wigderson. Quantum vs. classical communication and computation. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 63–68, 1998.

5 H. Buhrman and R. Špalek. Quantum verification of matrix products. In *Proceedings of the 17th ACM-SIAM symposium on Discrete algorithm*, pages 880–889, 2006.

6 E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

7 S. C. Draper and S. Malekpour. Compressed sensing over finite fields. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 669–673, 2009.

8 L. Gasieniec, C. Levcopoulos, and A. Lingas. Efficiently correcting matrix products. In *Proceedings of the 25th International Symposium on Algorithms and Computation*, pages 53–64, 2014.

9 S. Helmer and G. Moerkotte. Evaluation of main memory join algorithms for joins with set comparison join predicates. In *Proceedings of 23rd International Conference on Very Large Data Bases*, pages 386–395, 1997.

10 P. Høyer and R. de Wolf. Improved quantum communication bounds for disjointness and equality. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 299–310, 2002.

11 S. Jeffery, R. Kothari, F. Le Gall, and F. Magniez. Improving quantum query complexity of Boolean matrix multiplication using graph collision. *Algorithmica*, 2016. To appear.

12 B. Kalyanasundaram and G. Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

13 H. Klauck, R. Spalek, and R. de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM Journal on Computing*, 36(5):1472–1493, 2007. doi:10.1137/05063235X.

14 I. Kremer. Quantum communication. Master's thesis, The Hebrew University of Jerusalem, 1995.

15 D. Leinders and J. Van den Bussche. On the complexity of division and set joins in the relational algebra. *Journal of Computer and System Sciences*, 73(4):538–549, 2007.

16 A. Lingas. A fast output-sensitive algorithm for Boolean matrix multiplication. *Algorithmica*, 61(1):36–50, 2011.

17 N. Mamoulis. Efficient processing of joins on set-valued attributes. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 157–168, 2003.

18 S. Melnik and H. Garcia-Molina. Adaptive algorithms for set containment joins. *ACM Transactions on Database Systems*, 28:56–99, 2003.

19 K. Ramasamy, J. M. Patel, J. F. Naughton, and R. Kaushik. Set containment joins: The good, the bad and the ugly. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 351–362, 2000.

20 A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106:385–390, 1992.

21 A. Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya of the Russian Academy of Science, mathematics*, 67:159–176, 2003.

22 A. A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM Journal on Computing*, 41(5):1122–1165, 2012.

23 D. Van Gucht, R. Williams, D. P. Woodruff, and Q. Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 199–212, 2015.

24 J. Watrous. Course notes for *Theory of Quantum Information*. Available at: `https://cs.uwaterloo.ca/~watrous/CS766/`, 2013.

25 A. C.-C. Yao. Some complexity questions related to distributive computing. In *Proceedings of the 11th ACM Symposium on Theory of Computing*, pages 209–213, 1979.

26 A. C.-C. Yao. Quantum circuit complexity. In *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science*, pages 352–360, 1993.

On the Voting Time of the Deterministic Majority Process[*]

Dominik Kaaser[1], Frederik Mallmann-Trenn[2], and Emanuele Natale[3]

1 **University of Salzburg, Salzburg, Austria**
dominik@cosy.sbg.ac.at

2 **École normale supérieure, Paris, France, and Simon Fraser University, Burnaby, Canada**
fmallman@sfu.ca

3 **Sapienza Università di Roma, Rome, Italy**
natale@di.uniroma1.it

Abstract

In the deterministic binary majority process we are given a simple graph where each node has one out of two initial opinions. In every round, each node adopts the majority opinion among its neighbors. It is known that this process always converges in $\mathrm{O}(|E|)$ rounds to a two-periodic state in which every node either keeps its opinion or changes it in every round.

It has been shown by Frischknecht, Keller, and Wattenhofer (2013) that the $\mathrm{O}(|E|)$ bound on the convergence time of the deterministic binary majority process is even for dense graphs tight. However, in many graphs such as the complete graph the process converges in just a constant number of rounds from any initial opinion assignment.

We show that it is NP-hard to decide whether there exists an initial opinion assignment for which it takes more than k rounds to converge to the two-periodic state, for a given integer k. We then give a new upper bound on the voting time of the deterministic binary majority process. Our bound can be computed in linear time by carefully exploiting the structure of the potential function by Goles and Olivos. We identify certain modules of a graph G to obtain a new graph G^{Δ}. This new graph G^{Δ} has the property that the worst-case convergence time of G^{Δ} is an upper bound on that of G. Our new bounds asymptotically improve the best known bounds for various graph classes.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases distributed voting, majority rule

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.55

1 Introduction

Distributed voting is a fundamental problem in distributed computing. We are given a network of players modeled as a graph. Each player in the network starts with one initial opinion out of a set of possible opinions. Then the voting process runs either synchronously in discrete rounds or asynchronously according to some activation mechanism. During these rounds in the synchronous case, or upon activation in the asynchronous case, the players are allowed to communicate with their direct neighbors in the network with the main goal to eventually agree on one of the initial opinions. If all nodes agree on one opinion, we say this

[*] Research was supported by Austrian Science Fund (FWF): P 27613

opinion *wins* and the process *converges*. Usually, voting algorithms are required to be simple, fault-tolerant, and easy to implement [22, 24].

In this paper, we study the *deterministic binary majority process* which is defined as follows. We are given a graph $G = (V, E)$ where each node has one out of two opinions. The process runs synchronously in discrete rounds where each node in every round computes and adopts the majority opinion among all of its neighbors. It is known that this process always converges to a two-periodic state. The *convergence time* of a given graph for a given initial opinion assignment is the time required until this two-periodic state is reached. In this work we improve the bounds on the convergence time for given initial opinions and then we analyze the *voting time* of the process, which is the maximum convergence time over all possible initial opinion assignments.

In distributed computing, various variants of the majority process are used in fault-tolerant distributed consensus algorithms. In the analysis of structures of large networks, the deterministic binary majority process has widespread applications in the study of so-called *influence networks* [15]. Early applications can be found in distributed databases [16]. Further fields include sensor networks [6], the analysis of opinions in social networks [32], social behavior in game theory [12], chemical reaction networks [14], neural and automata networks [18], and cells' behavior in biology [7]. Variants of the deterministic binary majority process have been used in the area of distributed community detection [38, 28, 10]. In this context, the proposed community detection protocols exhibit a convergence time which can be bounded by the voting time of the deterministic binary majority process.

Among its many probabilistic variants that have been previously considered, plenty of work concerns *randomized voting* where in each step every node is allowed to contact a random sample of its neighbors and updates its current opinion according to the majority opinion in that sample [1, 5, 9, 13, 22, 23, 29, 30, 31, 33].

In an algorithmic game theoretic setting, the deterministic binary majority process can be seen as the simplest *discrete preference games* [8]. In this game theoretic perspective, the existence of *monopolies* has been investigated [2]. A monopoly in a graph is a set of nodes which start with the same opinion and cause all other nodes to eventually adopt this opinion. In the distributed computing area, a lot of research has been done to find small monopolies, see for example [34]. It has also been shown that there exist families of graphs with constant-size monopolies [4]. More recently, classes of graphs which do not have small monopolies have been investigated [35].

Many of these results relate to the voting time of the deterministic binary majority process. It was proven independently by Goles and Olivos [20], and Poljak and Sůra [36] with the same potential function argument that the deterministic binary majority process always converges to a two-periodic state. They later (independently) refined and generalized the potential function argument in several directions [17, 19, 21, 37]. Their proof was popularized in the *Puzzled* columns of Communications of the ACM [41, 42]. Recently, the same problem has been studied on infinite graphs w.r.t. a given probability distribution on the initial opinion assignments [3]. In [40], the authors provide a bound on the number of times a node in a given bounded-degree graph changes its opinion. Both [3] and [40] also investigate the probability that in the two-periodic state all nodes hold the same opinion.

As for the maximum time it takes for the process to converge over all initial opinion assignments, Frischknecht et al. [15] note that the potential argument by Goles et al. [20, 36, 42] can be used to prove an $\mathrm{O}(|E|)$ upper bound. They furthermore show that this upper bound is tight in general, by designing a class of graphs in which the deterministic binary majority process takes at least $\Omega\left(|V|^2\right)$ rounds to converge from a given initial opinion assignment. This construction has later been extended to prove lower bounds for weighted and multi-edges graphs by Keller et al. [27].

Once the process converges to the two-periodic state, each node stays either with its own opinion or changes its opinion in every round. A lot of attention has been given to the opinions to which the deterministic binary majority process converges. However, regarding the voting time, besides the $\mathrm{O}(|E|)$ upper bound that follows from the result by Goles et al. [20, 36, 42], no further upper bound on the voting time that holds for any initial opinion assignment has been proved. Still, one can observe that in many graphs the voting time is much smaller than $\mathrm{O}(|E|)$. For example, the voting time of the complete graph is one.

We show that for the deterministic binary majority process the question whether the voting time is greater than a given number is NP-hard. While for many generalizations of the deterministic binary majority process many decision problems are known to be NP-hard, at the best of our knowledge this is the first NP-hardness proof that does not require any additional mechanisms besides the bare majority rule of the deterministic binary majority process. However, as we show in the rest of the paper, it is possible to obtain upper bounds on the voting time which can be computed in linear time. A module of a graph is a subset of vertices S such that for each pair of nodes $u, v \in S$ it holds that $N(u) \setminus S = N(v) \setminus S$, where $N(u)$ denotes the set of neighbors of a node u. By carefully exploiting the structure of the potential function by Goles et al. we leverage the particular behavior that certain modules, which we call *families*, exhibit and prove that the voting time of a graph can be bounded by the voting time of a smaller graph that can be constructed in linear time by contracting suitable vertices.

We obtain a new upper bound that asymptotically improves the previous $\mathrm{O}(|E|)$ bound on graph classes which are characterized by a high number of modules that are either cliques or independent sets. An example for such graphs is the Turán graph $T(n, r)$, formed by partitioning a set of n vertices into r subsets of (almost) equal sizes and connecting two vertices by an edge whenever they belong to different subsets. For the convergence time of the Turán graph $T(n, r)$ we obtain an $\mathrm{O}\left(r^2\right)$ bound, compared to the previously best known bound of $\mathrm{O}\left(n^2\right)$. Also, for the convergence time of full d-ary trees we get an $\mathrm{O}(|V|/d)$ bound, compared to $\mathrm{O}(|V|)$ originating from the $\mathrm{O}(|E|)$ bounds. Further examples include the clique and the star graph, for which our bound gives a constant $\mathrm{O}(1)$ convergence time. Our bound relies on a well-known graph contraction technique based on identifying *equivalent nodes*. This technique is used in other related disciplines as well, including parallel and distributed computing. See, for example, the notion of *identical nodes* in the work by Sarıyüce et al. [39].

1.1 Preliminaries

We are given a graph $G = (V, E)$ and an initial opinion assignment defined as follows.

▶ **Definition 1.** An opinion assignment f_t in round $t \geq 0$ is a function $f_t : V \to \{0, 1\}$ which assigns for each $v \in V$ one out of two possible opinions. We will also denote opinion 1 as *white* and opinion 0 as *black*. The opinion assignment at time $t = 0$ is called *initial opinion assignment*.

The deterministic binary majority process can be defined as follows. Let v be an arbitrary but fixed vertex and $N(v)$ the set of neighbors of v. To compute $f_{t+1}(v)$ the node v computes the majority opinion of all of its neighbors in $N(v)$. In the case of a tie the node behaves *lazily*, that is, v stays with its own opinion. Otherwise, there is a *clear majority* and the node adopts the majority opinion. This leads to the following definition.

▶ **Definition 2.** Let $G = (V, E)$ be a graph and let f_0 be an initial opinion assignment such that $f_0 : V \to \{0, 1\}$. The deterministic binary majority process is the series of opinion assignments that satisfy the rule

$$f_{t+1}(v) = \begin{cases} 0 & \text{if } |\{u \in N(v) : f_t(u) = 0\}| > |\{u \in N(v) : f_t(u) = 1\}| \\ 1 & \text{if } |\{u \in N(v) : f_t(u) = 0\}| < |\{u \in N(v) : f_t(u) = 1\}| \\ f_t(v) & \text{otherwise.} \end{cases}$$

Note that the pair (G, f_0) completely determines the behavior of the system according to the majority process. We now define the main object of this work, the voting time.

▶ **Definition 3.** Given a graph $G = (V, E)$ and any initial opinion assignment f_0 on V, the *convergence time* $\mathfrak{T}$ of the majority process on G w.r.t. f_0 is $\mathfrak{T} = \mathfrak{T}(G, f_0) = \min \{t : \forall v \; f_{t+2}(v) = f_t(v)\}$. The *voting time* of G is defined as $\max_{f_0 \in \{0,1\}^V} \mathfrak{T}(G, f_0)$.

Observe that $\mathfrak{T}$ is indeed the number of steps until the process converges to a two-periodic state. This holds since the process is completely determined by the current opinion assignment. Thus $f_{t+2}(v) = f_t(v)$ also implies that $f_{t+3}(v) = f_{t+1}(v)$ for all nodes v.

In the following we assume without loss of generality that G is connected. For disconnected graphs the deterministic binary majority process runs independently in each connected component. Therefore, the resulting upper bounds on the voting time can be replaced by the maximum over the corresponding bounds in the individual connected components of G.

1.2 Our Contribution

First we define the *voting time decision problem* VTDP and show that it is NP-complete.

▶ **Definition 4** (voting time decision problem VTDP)**.** For a given graph G and an integer k, is there an assignment of initial opinions such that the voting time of G is at least k?

▶ **Theorem 5.** *Given a general simple graph G,* VTDP *is NP-complete.*

In Section 3 we extend known approaches to derive upper bounds on the voting time, which are tight for general graphs. In Section 3.2, we identify the following subsets of nodes that play a crucial role in determining the voting time of the deterministic binary majority process.

▶ **Definition 6.** A set of nodes S is called a *family* if and only if for all pairs of nodes $u, v \in S$ we have $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. We say that a family S is *proper* if $|S| > 1$.

The set of families of a graph forms a partition of the nodes into equivalence classes. Our main contribution is a proof that the voting time of the deterministic binary majority process is bounded by that of a new graph obtained by contracting its families into one or two nodes, as stated in the following theorem.

▶ **Definition 7.** Given a graph $G = (V, E)$, its *asymmetric graph* $G^{\Delta} = (V^{\Delta}, E^{\Delta})$ is the subgraph of G induced by the subset $V^{\Delta} \subseteq V$ constructed by contracting every family of odd-degree non-adjacent nodes to one node, and any other proper family to two nodes.

Let in the following V_{even} be the set of even-degree vertices in V and, analogously, let V_{odd} be the set of odd-degree vertices. Based on above definition of G^{Δ}, we give the following bound on the voting time.

▶ **Theorem 8.** *Given any initial opinion assignment on a graph $G = (V, E)$, the voting time of the deterministic binary majority process is at most*

$$1 + \min\left\{|E^\Delta| - \frac{|V^\Delta_{odd}|}{2}, \frac{|E^\Delta|}{2} + \frac{|V^\Delta_{even}|}{4} + \frac{7}{4} \cdot |V^\Delta|\right\} .$$

Furthermore, this bound can be computed in $\mathrm{O}(|E|)$ *time.*

As mentioned before, this bound becomes $\mathrm{O}\left(r^2\right)$ for the Turán graph $T(n, r)$ and $\mathrm{O}(|V|/d)$ for d-ary trees. Finally, in Section 3 we also give some insight into further interesting computational properties of the deterministic binary majority process. For example, we disprove a monotonicity of the convergence time w.r.t. the potential function and argue that the voting time is not, at least straightforwardly, bounded by the diameter of the graph.

2 NP-Completeness

If it was possible to efficiently compute the worst-case voting time, there would have been not much interest in investigating good upper bounds for it. In this section, we show that this is unlikely to be the case. We prove Theorem 5 by reducing 3SAT to the voting time decision problem. Given $\Phi \in$ 3SAT, we construct a graph $G = G(\Phi)$ such that the deterministic binary majority process on G simulates the evaluation of Φ. The graph G consists of h layers where $h = 3 + 4 \cdot n$. The first layer represents an assignment of the variables in Φ, the remaining layers represent Φ and ensure that the assignment of variables in Φ is valid. We will show that if Φ is satisfiable, then there exists an initial assignment of opinions for which the convergence time is exactly $h + 1$. If, however, Φ is not satisfiable, then any assignment of opinions will result in a convergence time strictly less than $h + 1$. We now give the formal proof.

Let $\Phi \in$ 3SAT be a Boolean formula in 3-conjunctive normal form. Let n be the number of variables of Φ. Let m be the number of clauses of Φ. The Boolean formula is of the form $\Phi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge \cdots \wedge (l_{m,1} \vee l_{m,2} \vee l_{m,3})$, where $l_{i,j} \in \{x_1, \overline{x}_1, x_2, \overline{x}_2, \cdots, x_n, \overline{x}_n\}$ is a literal for $1 \leq i \leq m$ and $1 \leq j \leq 3$.

We construct a graph G to simulate the evaluation of Φ as follows. Let $\ell = 10 \cdot (m+n)+1$. The graph consists of several layers. On the first layer, we place so-called literal cliques of size ℓ, and on the layers above we place the gates. In our reduction, we use OR-gates, an AND-gate, and 2/3-gates. Each gate consists of one or several nodes. Additionally, we have two so-called *mega-cliques* K_{white} and K_{black} of size ℓ.

Let g be an arbitrary but fixed gate. We denote a node on a layer below g which does not belong to g but is connected to g as *input node* to g. Additionally, we will denote a node that belongs to g and is connected to another gate on a layer above g as *output node* of g.

In the following, we assume that opinion 1, white, corresponds to Boolean TRUE and 0, black, corresponds to FALSE. The main idea of the construction is to show that an *activation signal* is transmitted from the bottom up through all layers. If the current assignment of opinions on the literal cliques corresponds to a satisfying assignment of Boolean values to Φ, then the process requires $h + 1$ steps. The main purpose of the OR-gates and the AND-gate is to evaluate Φ. The 2/3-gates check whether the opinion assignment to literal nodes is valid. That is, we need to enforce that the corresponding literal nodes for x_i and $\overline{x}_i$ are of opposite colors for every variable x_i of Φ. If either this condition is violated and variables x_i exist for which $x_i = \overline{x}_i$ or the current assignment of opinions on the literal cliques does not corresponds to a satisfying assignment of Boolean values to Φ, the construction enforces that the process stops prematurely after strictly fewer than $h + 1$ steps.

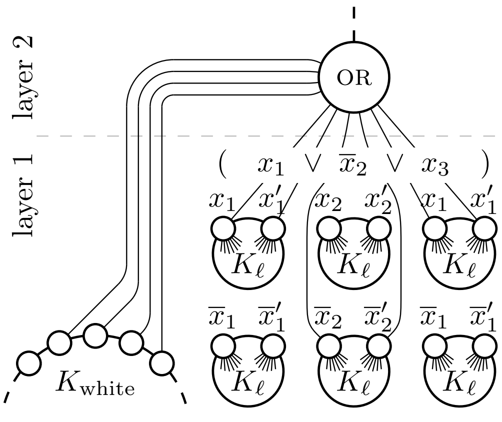

layer 2
layer 1
OR
(x_1 ∨ $\overline{x}_2$ ∨ x_3)
x_1 x_1' $\overline{x}_2$ x_2' x_1 x_1'
K_ℓ
$\overline{x}_1$ $\overline{x}_1'$ $\overline{x}_2$ $\overline{x}_2'$ $\overline{x}_1$ $\overline{x}_1'$
K_{white}

Figure 1 Literal nodes and OR-gate.

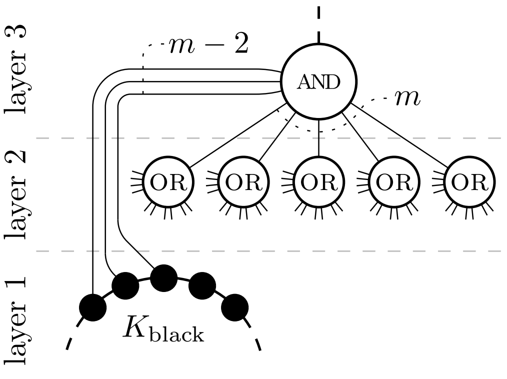


Figure 2 AND-gate.

Layer 1: Literal Cliques. We represent each variable x_i with two cliques, one for x_i and one for $\overline{x}_i$. Each clique has a size of ℓ which is defined above. Note that ℓ is odd. Additionally, we distinguish three so-called *representative nodes* in each of these cliques. Furthermore, we add two cliques of size ℓ to the graph which we call mega-cliques. Intuitively, these mega-cliques represent the Boolean values TRUE and FALSE. We will show that they cannot have the same color in order to achieve a long convergence time. The mega-cliques are used in all other gates.

Layer 2: Parallel OR-Gates. The OR-gates are placed on layer 2 and consist of one node v which is also the output node. There is one OR-gate for every clause. Fix a clause $(l_{j,1} \vee l_{j,2} \vee l_{j,3})$. Input nodes are three pairs of nodes (v_1, v_1'), (v_2, v_2'), and (v_3, v_3'), where (v_1, v_1') are two representative nodes of the literal clique for $l_{j,1}$, (v_2, v_2') are representatives of $l_{j,2}$, and (v_3, v_3') are representatives of $l_{j,3}$. That is, for each literal in the clause we connect the OR-gate on layer 2 to two of the three representative nodes of the corresponding literal clique on layer 1. The output node v is additionally connected to 4 nodes of the K_{white} mega-clique. Intuitively, we use the OR-gates to verify that for each clause at least one literal is TRUE. All clauses are evaluated simultaneously using an OR-gate for each clause. The OR-gate is shown in Figure 1.

Layer 3: AND-Gate. There is exactly one AND-gate on layer 3. This AND-gate consists of one output node denoted u_0, which has the following input nodes. It is connected to every output node of the OR-gates on layer 2 and to $m-2$ distinct nodes of the K_{black} mega-clique. Intuitively, the AND-gate is used toverify that every clause is satisfied. It is shown in Figure 2.

Layers 4 to 3 + 4n: 2/3-Gates. The 2/3-gates consist of a path v_1, v_2, v_3, and v_4. Each node of this path is connected to two distinct nodes of the K_{white}. The output node of the gate is v_4. The node v_1 of the first 2/3-gate on layer 4 is connected to the AND-gate on layer 3. The node v_1 of each of the following 2/3-gates is connected to the node v_4 of the previous 2/3-gate. Additionally, the input node of the i-th 2/3-gate is connected to three distinct nodes of the literal clique representing x_i and to three distinct nodes of the literal clique representing $\overline{x}_i$ on layer 1. The output node of the final 2/3-gate is connected to K_{black}. An example is shown in Figure 3. The 2/3-gates are used to verify that we do not have variables x_i in Φ for which the literal cliques of x_i and $\overline{x}_i$ have the same color. Observe that 2/3-gates span over 4 layers, and we have n such 2/3-gates.

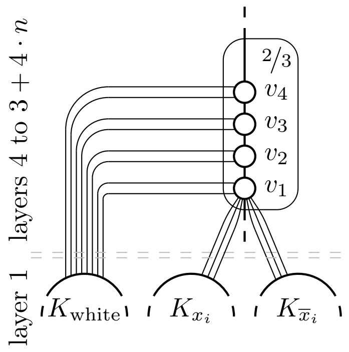


Figure 3 The 2/3-gate.

Literal cliques, OR-gates, and the AND-gate use only one layer, and 2/3-gates span over 4 layers. Therefore, the total number of layers is $h = 3 + 4 \cdot n$, which results from one layer for the literal cliques, one layer for the OR-gates, one layer for the AND-gate, and $4 \cdot n$ layers containing n concatenated 2/3-gates. A detailed example for such a graph G is given in [26]. Based on above description of G we prove the following lemmas, which are then used to show Theorem 5.

▶ **Lemma 9.** *If Φ is satisfiable, then there exists an assignment of opinions such that the convergence time in G is at least $h + 1$.*

To show Theorem 9, we construct an initial opinion assignment for which the gates change from black to white one layer after the other, assuming Φ is satisfiable. The full proof can be found in [26].

It remains to show that if Φ is not satisfiable, then the voting time in G is strictly less than $h + 1$. Recall that the voting time is the maximum of the convergence time over all possible initial opinion assignments.

▶ **Lemma 10.** *If Φ is not satisfiable, then there is no assignment of opinions such that the convergence time in G is at least $h + 1$.*

Before we prove this lemma, we establish several auxiliary lemmas which require the following definitions. Let u_0 denote the output node of the AND-gate. Consider the graph G' induced by the nodes of the AND-gate and the nodes of the 2/3-gates. Let u_i be the node at distance i to u_0 in G'. We observe that G' is a path $u_0, \ldots, u_\kappa$ consisting of the $4 \cdot n + 1$ top layers of the graph G. Consequently, $\kappa = 4 \cdot n$ and u_i is the i-th node on this path.

▶ **Definition 11** (Stable Time). We define the *stable time* $s(v)$ for any node $v \in V$ to be the first time step such that v does not change its opinion in any subsequent time step $t' > s(v)$ over all possible initial configurations. That is,

$$s(v) = \min \left\{ t : \forall f_0 \in \{0,1\}^V \ \ \forall t' \geq t \quad f_{t'}(v) = f_t(v) \right\} .$$

Accordingly, let for any subset $V' \subseteq V$ be $s(V')$ defined as $s(V') = \max \{s(v) : v \in V'\}$.

In the following, let V_K be the set of nodes of all cliques in $G(\Phi)$, that is, the nodes contained in the literal cliques and in the mega-cliques on layer 1. Furthermore, let $V_{K^{\text{R}}}$ be the set of representatives of the cliques and $V_{K^-} = V_K \setminus V_{K^{\text{R}}}$. That is, every clique K on layer 1 consists of $K^- \cup K^{\text{R}}$. Finally, let V_{OR} be the set of all output nodes of OR-gates. The following lemma shows that the layers become stable one after the other.

▶ **Lemma 12.** *It takes at most* 3 *time steps for the layers* 1 *and* 2 *consisting of literal cliques and* OR*-gates to become stable. Precisely, we have* (i) $s\left(V_{K^-}\right) = 1$*,* (ii) $s\left(V_{K^{\mathrm{R}}}\right) = 2$*, and* (iii) $s\left(V_{\mathrm{OR}}\right) = 3$*.*

The above lemma gives bounds on the stable time of layers 1 and 2. In the following, we argue that whenever a node changes its opinion in any step t after time step 3, it will not change its color in any subsequent time step $t' \geq t$ any more. We therefore define the so-called *activation time* of a node $v \in G'$ as follows.

▶ **Definition 13** (Activation Time)**.** Let c be the color of the K_{black} mega-clique at time 2 and let f_0 be an arbitrary but fixed initial opinion assignment. We define the *activation time* of a node $v \in G'$ to be the first time step after time step 3 in which the node v adopts opinion c. That is, $a(v) = \min\{t \geq 3 : f_t(v) = c\}$. If v does not change its color after time step 3 we write $a(v) = 3$.

We now use the above definition to state the following lemma, which describes that every node $u_i \in G'$ with $i \geq 1$ changes its color at most once after time step 3. Note that this covers the nodes of the 2/3-gates.

▶ **Lemma 14.** *Let f_0 be an arbitrary but fixed initial opinion assignment. Let t be the activation time w.r.t. f_0 of the node $u_i \in G'$ with $i \geq 1$ such that $t = a(u_i)$. Then for all $t' \geq t$ we have $f_{t'}(u_i) = f_t(u_i)$.*

Proof. By Theorem 12, all nodes $u \in V_{K^{\mathrm{R}}}$ are stable at $t' \geq 2$. We now distinguish two cases.

Case 1: $i \mod 4 \neq 1$. Observe that u_i can only change its color at time $t = a(u_i)$, if it had a different color than K_{white} in the previous round. This holds, since every node u_i with $i \mod 4 \neq 1$ has the same number of connections to K_{white} than to nodes in $V \setminus K_{\text{white}}$. Since furthermore the process behaves lazy, any node u_i which has the same color as K_{white} cannot change its opinion back to the opposite color any more.
Case 2: $i \mod 4 = 1$. The node u_i is a v_1 node of the j-th 2/3-gate with $j = \lceil i/4 \rceil$. Therefore it is connected to three representatives of each literal clique for x_j and $\overline{x}_j$. The literal representatives of x_j and $\overline{x}_j$ are stable at time $t' \geq 2$. Now if x_j and $\overline{x}_j$ have the same color c, then u_i has $6 > |N(u_i)|/2$ edges to nodes of color c. Therefore, the node does not change its color any more after time step 3. That is, we have $a(u_i) = 3$ and also $f_{t'}(u_i) = c$ for any consecutive time step $t' \geq 3$. If, however, x_j and $\overline{x}_j$ do not have the same color, these edge *cancel* each other out and the color of node u_i is determined by u_{i-1}, u_{i+1}, and K_{white}. Therefore, the same argument as in the first case holds. ◀

In the following we examine the behavior of layer 3 which contains only the AND-gate. Recall that u_0 is the output node of the AND-gate. The next lemma describes the following fact. The AND-gate u_0 can only change its color in a time step $t \geq 4$ if u_1 changed its color in time step $t - 1$. After this change at time t, the node u_0 cannot change its color again.

▶ **Lemma 15.** *Let f_0 be an arbitrary but fixed initial opinion assignment and let furthermore t be the round after node u_1 has been activated such that $t = a(u_1) + 1$. For all consecutive rounds $t' \geq t$ we have $f_{t'}(u_0) = f_t(u_0)$. That is, the* AND*-gate does not change its opinion any more once the node u_1 has become stable.*

The following lemma implies that in order to reach a convergence time of $h + 1$ the gates on the path $u_0, \ldots, u_\kappa$ in G' have to activate one after the other starting with u_0 at time 4. Recall that $\kappa = 4 \cdot n$.

▶ **Lemma 16.** *Let f_0 be an arbitrary but fixed initial opinion assignment and let $u_i \in G'$ be a node with $0 \le i \le \kappa$. If $a(u_i) < i + 4$ w.r.t. f_0, then $\mathfrak{T}(G(\Phi), f_0) < h + 1$.*

In the following two lemmas, we enforce that initial opinion assignments which do not represent valid assignments of Boolean values to literal cliques result in premature termination of the deterministic binary majority process in $G(\Phi)$. An assignment is called *illegal* if there exist literal cliques such that the majority of x_i and the majority of $\overline{x}_i$ have the same initial color.

▶ **Lemma 17.** *For any illegal initial opinion assignment f_I to $G(\Phi)$, the convergence time $\mathfrak{T}(G(\Phi), f_I)$ is strictly less than $h + 1$.*

▶ **Lemma 18.** *If after two time steps K_{white} and K_{black} have the same color, the process stops after strictly fewer steps than $h + 1$.*

From above lemmas we conclude that Theorem 10 holds, and together with Theorem 9, Theorem 10 yields Theorem 5. The full proofs can be found in [26].

Proof of Theorem 5. It is easy to see that VTDP is in NP. Furthermore, we can polynomially reduce 3SAT to VTDP. The correctness proof of the reduction follows from Theorem 9 and Theorem 10. Therefore we conclude that VTDP is NP-complete. ◀

3 Bounds on the Voting Time

Since the problem is NP hard, we cannot hope to calculate the voting time of a graph efficiently. Nevertheless, in this section we show, that it is possible to obtain non-trivial upper bounds on the voting time that are easy to compute. This section is dedicated to proving our upper bound on the voting time, Theorem 8. The main contribution of this theorem is the influence of symmetry which is studied in Section 3.2.

We start by giving a formal version of the potential function argument [20, 36] as conceived in [42]. In the following we assume that each edge in $\{x, y\} \in E$ can be replaced by two directed edges (x, y) and (y, x). The main idea is based on so-called *bad arrows* defined as follows.

▶ **Definition 19.** Let $G = (V, E)$ be a graph with initial opinion assignment f_0. Let v denote an arbitrary but fixed node and $u \in N(v)$ a neighbor of v. Let t denote an arbitrary but fixed round. The directed edge (v, u) is called *bad arrow* if and only if the opinion of u in round $t + 1$ differs from the opinion of v in round t. We will also denote the bad arrows which have their tail at round $t = 0$ as *initial bad arrows.*

Intuitively, each of these directed edges (v, u) can be seen as *advice* given from v to u in the voting process. In the case of a bad arrow the advice was not followed by u since it has a different opinion in the following round than v. Observe that each bad arrow is incident at exactly two nodes and thus we say it is *outgoing* in the node at its tail and incoming in the node at its head. An example of such a bad arrow can be seen in Figure 4.

▶ **Theorem 20.** *Let $G = (V, E)$ be a graph which contains only vertices of odd degree. The voting time of the deterministic binary majority process on G is at most $1 + W_{bad}$ where W_{bad} is an upper bound on the number of initial bad arrows for any initial opinion assignment on G. In particular, the voting time of G is at most $2 \cdot |E| + 1$.*

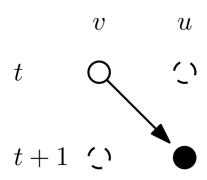


Figure 4 A *bad arrow* from node v to node u in round t.

The idea of the proof is to define a potential function ϕ_t that is strictly monotonically decreasing over the time. Let f_0 be any initial opinion assignment. The potential function ϕ_t is simply the number of bad arrows defined in Theorem 19, that is

$$\phi_t = \phi_t(G, f_t) = |\{(v, u) \in E : f_{t+1}(u) \neq f_t(v)\}| \quad .$$

For the full proof, see [26]. Note that in Theorem 20 it is assumed that all nodes of the graph have odd-degree. In the following we show how to remove this assumption.

▶ **Definition 21.** Let $G = (V, E)$ be a graph. The graph $G^* = (V, E^*)$ is the graph obtained by adding a self loop to every node of even degree in G. More formally,

$$E^* = E \ \cup \bigcup_{v \in V_{\text{even}}} (v, v) \quad .$$

From the definition it follows that $|E^*| = |E| + |V_{\text{even}}|$.

▶ **Theorem 22.** *The voting time of the deterministic binary majority process on any graph $G = (V, E)$ is at most $1 + W_{bad}$, where W_{bad} is an upper bound on the number of initial bad arrows in G^*.*

The proof is based on the fact that for every node $v \in V$ the sequence of opinions, $(f_t(v))$, is exactly the same for the deterministic binary majority process in G as for the deterministic binary majority process in G^*. For the full proof, see [26].

Observe, that while the number of bad arrows is used in the potential function, the convergence time is, however, not monotone w.r.t. the number of initial bad arrows.

▶ **Lemma 23.** *The convergence time is not monotone w.r.t. the number of initial bad arrows.*

The upper bound on the voting time considered in [27] follows from the $2 \cdot |E|$ upper bound on the number of bad arrows of Theorem 20. Clearly, this result can be improved by a factor of 2 by simply applying the observation that the number of initial bad arrows in G^* is at most $|E| - |V_{odd}|/2$. Therefore, from Theorem 22 we obtain the following corollary.

▶ **Corollary 24.** *The voting time of the deterministic binary majority process on any graph $G = (V, E)$ is at most $1 + |E| - |V_{odd}|/2$.*

▶ Remark. Theorem 24 is tight for general graphs up to an additive constant of 1. Indeed, consider a path with an initial opinion assignment on which the opinions alternate except for the last two nodes, which share the same opinion.

Suppose that, instead of specifying the initial opinion assignment, we decide in advance what bad arrows are there. We can do that by deciding for each ordered pair (u, v) for which $\{u, v\} \in E$ whether we want to have a bad arrow going from u to v. We formalize this notion by means of the following definitions.

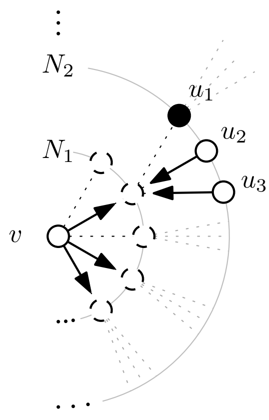


Figure 5 The opinions of each second neighborhood are uniquely determined.

▶ **Definition 25.** Let $G = (V, E)$ be a graph and $\beta : V \times V \to \{0, 1\}$ denote a characteristic function on $V \times V$. Then β is a bad arrows assignment on G if there exists an opinion assignment f on G that determines β such that β is the indicator function of the bad arrows we have on G w.r.t. the opinion assignment f.

In proving upper bounds on the voting time we consider the bad arrows assignment determined by the initial opinion assignment. One may wonder whether in doing so we are *losing information*. In the following lemma we show that, given a valid bad arrows assignment, we can reconstruct the initial opinion assignment up to exchanging black and white (and up to two more possibilities in bipartite graphs).

▶ **Lemma 26.** *Let G be a connected graph and let β be a valid bad arrows assignment on G. If the graph is not bipartite, there are exactly two opinion assignments, otherwise there are exactly four opinion assignments that determine β.*

Proof. Let $v \in V$ denote an arbitrary but fixed vertex. We now denote the set $\{v\}$ as N_0 and the set of direct neighbors of v as N_1 to define the i-th neighborhood N_i for $i \geq 2$ as

$$N_i = \left(\bigcup_{u \in N_{i-1}} N(u) \right) \setminus \left(\bigcup_{j=1}^{i-1} N_j \right) .$$

We now show by an induction on $k = 0, 1, 2, \ldots$ that the colors of all nodes in $N_{2 \cdot k}$ are determined by the color of v. The base-case is trivial since for $k = 0$ we have $N_0 = \{v\}$. For the induction step we observe that according to the induction hypothesis the color of each node in $N_{2 \cdot k}$ is determined. We now observe that the color at time 1 of each node in $N_{2 \cdot k+1}$ is determined by β and the colors at time 0 of the nodes in $N_{2 \cdot k}$. Vice versa, also the colors at time 0 of nodes in $N_{2 \cdot (k+1)}$ are determined by β and the colors at time 1 of each node in $N_{2 \cdot k+1}$. This concludes the induction.

An example is shown in Figure 5. In this figure it is clear that v and, e.g., u_1 must have a different color, for the following reason. Since u_1 does not have a bad arrow to its neighbor in N_1, it has the same color in the next round as this neighbor. But this neighbor's color in the next round is different to the current color of v because of the bad arrow assignment.

Observe that from above induction the lemma follows immediately for bipartite graphs. We can fix the colors for two arbitrary nodes, one from each of the two sets of non-adjacent nodes, to determine all other nodes' colors. This gives us four possible opinion assignments for a given bad arrow assignment β. If the graph is not bipartite there must exist a cycle of odd length. The opinion assignments for all nodes of this cycle are determined by β with the same argument as in above induction. Therefore, not only the colors of even neighborhoods N_{2k} are determined, but also of odd neighborhoods N_{2k+1}. This leaves us with exactly two possible initial opinion assignments, which concludes the proof. ◀

3.1 Improved Bounds for Dense Graphs

We observe that Theorem 24 is (almost) tight, and it gives us a voting time linear in the number of vertices for sparse graphs where $|E| = \mathrm{O}(|V|)$. However, for dense graphs with, e.g., $|E| = \Omega\left(|V|^2\right)$ there is room for improvement. Now the main goal in this following subsection is to reduce the dominant term of the voting time even further, which leads us to the following theorem which is formally shown in [26].

▶ **Theorem 27.** *Let $G = (V, E)$ be a graph. For any initial opinion assignment f_0 on G, the convergence time of the deterministic binary majority process is at most $1 + \frac{|E|}{2} + \frac{|V_{\text{even}}|}{4} + \frac{7}{4} \cdot |V|$.*

▶ Remark. One might intuitively assume that the voting time is bounded by the diameter of the network. However, this is not true, at least straightforwardly, as there exist graphs G where the convergence time w.r.t. a given initial opinion assignments f_0 is asymptotically larger than the diameter of the network, that is, $\mathfrak{T}(G, f_0) \gg \operatorname{diam}(G)$.

3.2 The Influence of Symmetry

We observe that the majority process is much faster on graphs that exhibit certain types of symmetry, such as the star graph, the complete graph and many other graphs in which several nodes share a common neighborhood. We investigate this feature of the process to further improve the bounds obtained so far. We recall that a set of nodes S is called a *family* if and only if for all nodes $u, v \in S$ we have $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. The key fact is that these nodes of any family will behave in a similar way after the first step.

▶ **Definition 28.** Let $\operatorname{fam}(u)$ denote the family of u. We write $u \sim v$ if $\operatorname{fam}(u) = \operatorname{fam}(v)$.

▶ **Lemma 29.** *The relation $\sim$ defines an equivalence class. In particular, all nodes in the same family either form a clique or a stable set, and they all have the same degree in G.*

▶ **Corollary 30.** *For any graph G, its asymmetric graph G^{Δ} is well-defined.*

Proof. According to Theorem 29, the set of families is a partition of the nodes of G. By construction of G^{Δ}, every family S in G is replaced by one or two nodes in G^{Δ}. Therefore, there is a bijection between the families in G and the corresponding node or pair of nodes in G^{Δ}. Hence G^{Δ} is well-defined. ◀

We are now ready to prove Theorem 8.

▶ Remark. While we show that for the voting time we have $\max_f \mathfrak{T}(G^{\Delta}, f) \geq \max_f \mathfrak{T}(G, f)$, in general it is not the case that $\mathfrak{T}(G^{\Delta}, f) \geq \mathfrak{T}(G, f)$ for every opinion assignment f. A formal statement along with a counterexample is given in [26].

Proof of Theorem 8. Let v and v' be two nodes of the same family $\operatorname{fam}(v) = \operatorname{fam}(v')$, having the same color at time t. Since v and v' observe the same opinions in their respective neighborhood, v and v' will also have the same color anytime after t. It follows that if at some time t there is a bad arrow going from v to some neighbor u (or from u to v), then there will also be a bad arrow from v' to u (or from u to v'). In particular, this implies that whenever the number of bad arrows adjacent to v is decreased by some amount c, also the identical number of bad arrows adjacent to v' will be decrease by the same amount c.

Recall the proofs of Theorem 24 and Theorem 27. An estimate of the voting time is obtained by upper bounding the number of bad arrows that can possibly disappear during the process. The main argument is the following. It suffices to only consider the bad-arrows adjacent to v in $G^{\triangle}$, since the corresponding bad arrows adjacent to v' will disappear whenever those adjacent to v do.

Let v and v' be two nodes with $\text{fam}(v) = \text{fam}(v')$ having a different color at time t. We can divide every such family that contains nodes of different opinions into two sets S_0 and S_1 according to their initial opinion in the first round. Note that all nodes in either set behave identically. In particular, an adjacent bad arrow from a node u to all nodes of either set disappears at the same time. Since there is bijection between the families of G and the pairs of nodes and singletons of $G^{\triangle}$, and by applying Theorem 24 and Theorem 27 we can bound the voting time by bounding the bad arrows in $G^{\triangle}$. This yields the first part of the claim. Using [11], one can obtain the modular decomposition of G in $\mathrm{O}(|E|)$ time steps. In another $\mathrm{O}(|E|)$ time steps one can select from the modular decomposition those modules that form a family, using that all nodes of a family have the same degree. Hence, $G^{\triangle}$ can be constructed in linear time. ◀

Acknowledgement. We would like to thank our supervisors Petra Berenbrink, Andrea Clementi, Robert Elsässer, and Claire Mathieu for helpful discussions and important hints.

References

1 D. Aldous and J. Fill. Reversible Markov Chains and Random Walks on Graphs, 2002. Unpublished. `http://www.stat.berkeley.edu/~aldous/RWG/book.html`.

2 V. Auletta, I. Caragiannis, D. Ferraioli, C. Galdi, and G. Persiano. Minority Becomes Majority in Social Networks. In *Proc. WINE '15*, pages 74–88, 2015.

3 I. Benjamini, S.-O. Chan, R. O'Donnell, O. Tamuz, and L.-Y. Tan. Convergence, unanimity and disagreement in majority dynamics on unimodular graphs and random graphs. *CoRR*, abs/1405.2486, 2014. URL: `http://arxiv.org/abs/1405.2486`.

4 E. Berger. Dynamic Monopolies of Constant Size. *Journal of Combinatorial Theory, Series B*, 83(2):191–200, 2001.

5 S. Brahma, S. Macharla, S.P. Pal, and S.K. Singh. Fair Leader Election by Randomized Voting. In *Proc. ICDCIT '04*, pages 22–31, 2004.

6 F. Bénézit, P. Thiran, and M. Vetterli. Interval consensus: From quantized gossip to voting. In *Proc. ICASSP '09*, pages 3661–3664, 2009.

7 L. Cardelli and A. Csikász-Nagy. The Cell Cycle Switch Computes Approximate Majority. *Scientific Reports*, 2(656), 2012.

8 F. Chierichetti, J.M. Kleinberg, and S. Oren. On Discrete Preferences and Coordination. In *Proc. EC '13*, pages 233–250, 2013.

9 C. Cooper, R. Elsässer, H. Ono, and T. Radzik. Coalescing Random Walks and Voting on Connected Graphs. *SIAM Journal on Discrete Mathematics*, 27(4):1748–1758, 2013.

10 G. Cordasco and L. Gargano. Community Detection via Semi–Synchronous Label Propagation Algorithms. In *Proc. BASNA '10*, pages 1–8, 2010.

11 A. Cournier and M. Habib. A New Linear Algorithm for Modular Decomposition. In *Proc. 19th Colloquium on Trees in Algebra and Programming (CAAP '94)*, pages 68–84, 1994.

12 X. Deng and C. Papadimitriou. On the Complexity of Cooperative Solution Concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

13 P. Donnelly and D. Welsh. Finite particle systems and infection models. *Mathematical Proceedings of the Cambridge Philosophical Society*, 94(01):167–182, 1983.

14 David Doty. Timing in Chemical Reaction Networks. In *Proc. SODA '14*, pages 772–784, 2014.

15 S. Frischknecht, B. Keller, and R. Wattenhofer. Convergence in (Social) Influence Networks. In *Proc. DISC '13*, pages 433–446, 2013.

16 D. Gifford. Weighted Voting for Replicated Data. In *Proc. SOSP '79*, pages 150–162, 1979.

17 E. Goles. Local Graph Transformations Driven by Lyapunov Functionals. *Complex Systems*, 3(1):173–184, 1989.

18 E. Goles and S. Martínez. *Neural and Automata Networks.* Kluwer, 1990.

19 E. Goles and A.M. Odlyzko. Decreasing Energy Functions and Lengths of Transients for Some Cellular Automata. *Complex Systems*, 2(5):501–507, 1988.

20 E. Goles and J. Olivos. Periodic behaviour of generalized threshold functions. *Discrete Mathematics*, 30(2):187–189, 1980.

21 E. Goles-Chacc, F. Fogelman-Soulié, and D. Pellegrin. Decreasing energy functions as a tool for studying threshold networks. *Discrete Applied Mathematics*, 12(3):261–277, 1985.

22 Y. Hassin and D. Peleg. Distributed Probabilistic Polling and Applications to Proportionate Agreement. *Information and Computation*, 171(2):248–268, 2001.

23 R. Holley and T. Liggett. Ergodic Theorems for Weakly Interacting Infinite Systems and the Voter Model. *The Annals of Probability*, 3(4):643–663, 1975.

24 Barry W. Johnson, editor. *Design & Analysis of Fault Tolerant Digital Systems.* Addison-Wesley, 1989.

25 D. Kaaser, F. Mallmann-Trenn, and E. Natale. Brief Announcement: On the Voting Time of the Deterministic Majority Process. In *Proc. DISC '15*, 2015.

26 D. Kaaser, F. Mallmann-Trenn, and E. Natale. On the Voting Time of the Deterministic Majority Process. *CoRR*, abs/1508.03519, 2015. URL: `http://arxiv.org/abs/1508.03519`.

27 B. Keller, D. Peleg, and R. Wattenhofer. How Even Tiny Influence Can Have a Big Impact! In *Proc. FUN '14*, pages 252–263, 2014.

28 K. Kothapalli, S. Pemmaraju, and V. Sardeshmukh. On the Analysis of a Label Propagation Algorithm for Community Detection. In *Proc. ICDCN '13*, pages 255–269, 2013.

29 N. Lanchier and C. Neuhauser. Voter model and biased voter model in heterogeneous environments. *Journal of Applied Probability*, 44(3):770–787, 2007.

30 T. Liggett. *Interacting Particle Systems.* Springer, 1985.

31 F. Mallmann-Trenn. Bounds on the voting time in terms of the conductance. Master's thesis, Simon Fraser University, 2014. Master's thesis. `http://summit.sfu.ca/item/14502`.

32 E. Mossel and O. Tamuz. Opinion Exchange Dynamics. *CoRR*, abs/1401.4770, 2014. URL: `http://arxiv.org/abs/1401.4770`.

33 R. Oliveira. On the coalescence time of reversible random walks. *Transactions of the American Mathematical Society*, 364(4):2109–2128, 2012.

34 D. Peleg. Local majorities, coalitions and monopolies in graphs: a review. *Theoretical Computer Science*, 282(2):231–257, 2002.

35 D. Peleg. Immunity against Local Influence. In *Language, Culture, Computation. Computing - Theory and Technology*, volume 8001 of *LNCS*, pages 168–179. Springer, 2014.

36 S. Poljak and M. Sůra. On periodical behaviour in societies with symmetric influences. *Combinatorica*, 3(1):119–121, 1983.

37 S. Poljak and D. Turzík. On an application of convexity to discrete systems. *Discrete Applied Mathematics*, 13(1):27–32, 1986.

38 U. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

39 A. Sarıyüce, E. Saule, K. Kaya, and U. Çatalyürek. Shattering and Compressing Networks for Betweenness Centrality. In *Proc. SDM '13*, pages 686–694, 2013.
40 O. Tamuz and R. J. Tessler. Majority Dynamics and the Retention of Information. *Israel Journal of Mathematics*, 206(1):483–507, 2015.
41 P. Winkler. Puzzled: Delightful Graph Theory. *Communications of the ACM*, 51(8):104, 2008.
42 P. Winkler. Puzzled: Solutions and Sources. *Communications of the ACM*, 51(9):103, 2008.

Space-Efficient Biconnected Components and Recognition of Outerplanar Graphs

Frank Kammer[1], **Dieter Kratsch**[2], **and Moritz Laudahn**[3]

1 **Institut für Informatik, Universität Augsburg, Augsburg, Germany**
`kammer@informatik.uni-augsburg.de`
2 **LITA, Université de Lorraine, Metz, France**
`dieter.kratsch@univ-lorraine.fr`
3 **Institut für Informatik, Universität Augsburg, Augsburg, Germany**
`moritz.laudahn@informatik.uni-augsburg.de`

Abstract

We present space-efficient algorithms for computing cut vertices in a given graph with n vertices and m edges in linear time using $O(n + \min\{m, n \log\log n\})$ bits. With the same time and using $O(n + m)$ bits, we can compute the biconnected components of a graph. We use this result to show an algorithm for the recognition of (maximal) outerplanar graphs in $O(n \log\log n)$ time using $O(n)$ bits.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases graph algorithms, space efficiency, cut vertices, maximal outerplanar graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.56

1 Introduction

Nowadays the use of small mobile devices like tablets and smartphones is ubiquitous. Typically they will not be equipped with large memory and common actions like storing (many) pictures may even decrease the available memory significantly. This triggers the interest in data structures and algorithms being space-efficient. (Time) Efficient algorithms are a classical subject in computer science. The corresponding algorithms course is often based on the textbook of Cormen et al. [8]. There is also a long tradition in the development of algorithms that use as few bits as possible; famous results are the two results of Savitch [19] and Reingold [18] on reachability in directed and undirected graphs, respectively. However, the running times of their algorithms are far away from the fastest algorithms for that problem and are therefore of small practical interest. Moreover, Edmonds et al. [10] have shown in the so-called NNJAG model that only a slightly sublinear working-space bound is possible for an algorithm that solves the reachability problem when required to run in polynomial time. This motivates the recent interest in *space-efficient algorithms*, i.e., algorithms that use as few working space as possible under the condition that their running time (almost) matches the running time of the best algorithm(s) without any space restrictions.

A useful model of computation for the development of algorithms in that context is the word RAM with a read-only input, a read-write working memory, and a write-only output. As usual, we assume that for a given instances of size n, the word size is $\Omega(\log n)$. One of the first problems considered for space-efficient algorithms is sorting [16], which was finally solved to optimality [5, 17]. Other researchers considered problems in geometry [1, 3, 4].

Recent research started to focus on space-efficient graph algorithms [2, 11]. However, there is still only a very short list of problems with space-efficient graph algorithms: depth-first search, breadth-first search, (strongly) connected components, topological sorting, shortest path.

We continue this work on space-efficient graph algorithms and consider the basic problems to compute the cut vertices and to decompose a given undirected graph into its biconnected components. Tarjan's linear time algorithm [20] solving this problem has been implemented in almost any usual programming language. However the algorithm requires $\Omega(n \log n)$ bits on n-vertex graphs. The idea of our algorithm is to classify the edges via a DFS as tree and back edges and to mark those tree edges that build a cycle with a back edge. The marking allows us subsequently to determine the cut vertices and the biconnected components. Given a graph with n vertices and m edges, the whole algorithm runs in $O(n+m)$ time using $O(n + \min\{m, n \log\log n\})$ bits, which is $O(n)$ in sparse graphs. Due to the lower bound of Edmonds et al. [10], there is not much hope for an algorithm that uses $o(n)$ bits.

Finally we study the recognition of outerplanar graphs, i.e., those graphs having a planar embedding with all vertices on the outer face. The problem has been studied in various settings and linear time algorithms have been given, e.g., by Mitchell [15] and Wiegers [21]. However, both algorithms modify the given graph by removing vertices of degree 2, which is not possible in our model. An easy solution would be to copy the given graph in the working memory, but this requires $\Omega(n \log n)$ bits for a graph with n vertices. Another problem is that if the neighbors of a removed vertex are not adjacent, then both algorithms above want to add a new edge connecting the neighbors. Storing all these new edges also can require $\Omega(n \log n)$ bits. Our algorithm runs in time $O(n \log\log n)$ and uses $O(n)$ bits, and determines if the input graph is outerplanar, as well as if it is maximal outerplanar. To obtain our algorithm, we can not simply remove vertices of degree 2. With each removed vertex v we have to remove the so-called chain of vertices of degree 2 that contains v and we have to choose the chains carefully such that we have only very few new edges at a time in our graph.

2 Preliminary

For graph-theoretic notions not defined in the paper we refer to the monograph of Diestel [9]. For basic notions in construction and analysis of algorithms and a large collection of fundamental algorithms like, e.g., depth-first search (DFS) we refer to the textbook of Cormen et al. [8]. To develop space-efficient algorithms, the details of the representation of an input graph are more important than in the classic setting because it is rarely possible to modify and store a given representation. We use the terminology of [11]. In particular, if we say that a graph is represented via *adjacency arrays*, then we assume that, given a vertex u and an index i, we can determine the ith edge $\{u, v\}$ of u in constant time. Moreover, *cross pointers* allow us to determine the index of $\{u, v\}$ in the adjacency array of v in constant time. As usual, we always assume that an n-vertex graph has vertices $V = \{1, \ldots, n\}$.

Our algorithms make use of *rank-select data structures*. A rank-select data structure is initialized on a bit sequence $B = (b_1, \ldots, b_n)$ and then supports the following two queries.
$\boldsymbol{rank_B(j)}$ $(j \in \{0, \ldots, n\})$: Return $\sum_{i=1}^{j} b_i$
$\boldsymbol{select_B(k)}$ $(k \in \{1, \ldots, \sum_{i=1}^{n} b_i\})$: Return the smallest $j \in \{1, \ldots, n\}$ with $rank_B(j) = k$.
Rank-select data structures for bit sequences of length n that support rank and select queries in constant time and occupy $O(n)$ bits can be constructed in $O(n)$ time [7].

Assume that $d_1, \ldots, d_n \in \mathbb{N}_0$ and that it is desired to allocate n bit strings $A_1, \ldots, A_n$ such that, for $k = 1, \ldots, n$, A_k consists of d_k bits and (the beginning of) A_k can be located in constant time. Take $N = \sum_{j=1}^{n} d_j$. We say that $A_1, \ldots, A_n$ are stored with *static space allocation* if we allocate $A_1, \ldots, A_n$ within an array of size N. In $O(n+N)$ time, we can compute the sums $s_k = k + \sum_{j=1}^{k-1} d_j$ for $k = 1, \ldots, n$ and a rank-select data structure for the bit vector B of size $n+N$ whose ith bit, for $i = 1, \ldots, n+N$, is 1 exactly if $i = s_k$ for some $k \in \{1, \ldots, n\}$. This allows us, given a $k \in \{1, \ldots, n\}$ to compute the number of bits used by the arrays $A_1, \ldots, A_{k-1}$ and thus the location of A_k in constant time by evaluating $select_B(k) - k$. One application of static space allocation—as already shown by [13]—is to store data for each vertex v consisting of $O(\deg(v))$ bits where $\deg(v)$ is the degree of v. Given a vertex, we then can locate its data in constant time and the whole data can be stored with $O(n+m)$ bits.

To maintain subsets of vertices or of edge indices we use a data structure by Hagerup and Kammer [12, Lemma 4.4] that is called a choice dictionary.

▶ **Theorem 1.** *Let $n \in I\!N$. A choice dictionary is a data structure that maintains an initially empty subset S of $\{1, \ldots, n\}$ under insertion, deletion, membership queries, and an operation called choice that returns an arbitrary element of S. The data structure can be initialized in $O(1)$ time using $O(n)$ bits of working space and supports each of its operations in constant time. The choice dictionary can also be extended by an operation called iteration that returns all elements in S in a time linear in $|S|$.*

We also use a simplified version of the ragged dictionary that was introduced by of Elmasry et al. [11, Lemma 2.1] and named in [12]. Missing proofs can be found in [14].

▶ **Theorem 2.** *For every fixed $n \in I\!N = \{1, 2, \ldots\}$ as well as integers $b = O(\log n)$ and $\kappa = O(n/\log n)$, there is a dictionary that can store a subset A of $\{1, \ldots, n\}$ with $|A| \leq \kappa$, each $a \in A$ with a string h_a of satellite data of b bits, in $O(n)$ bits such that the following operations all run in $O(\log \log n)$ time: h_a can be inspected for each $a \in A$ and elements with their satellite data can be inserted in and deleted from A.*

3 Cut Vertices

A *cut vertex* of a connected undirected graph G is a vertex v such that $G - v$ is disconnected. Furthermore a graph is *biconnected* if it is connected and does not have a cut vertex. We first show how to compute cut vertices in $O(n+m)$ time using $O(n+m)$ bits on an undirected graph $G = (V, E)$ with $n = |V|$ and $m = |E|$. Afterwards, we present a second algorithm that has the same running time and uses $O(n \log \log n)$ bits.

We start with the description of an algorithm, but for the time being, do not care on the running time and the amount of working space. Using a single DFS we are able to classify all edges as either tree edges or back edges. During the execution of a DFS we call a vertex white if it has not been reached by the DFS, gray if the DFS has reached the vertex, but has not yet retracted from it and black if the DFS has retracted from the vertex. W.l.o.g., we assume that all our DFS runs are deterministic such that every DFS run explores the edges of G in the same order. Let T denote the *DFS tree* of G, i.e., the subgraph of G consisting only of tree edges, which we always assume to be rooted at the start vertex of the DFS. We call a tree edge $\{u, v\}$ of T with u being the parent of v *full marked* if there is a back edge from a descendant of v to a strict ancestor of u, *half marked* if it is not full marked and there exists a back edge from a descendant of v to u, and *unmarked*, otherwise. Then one can easily prove the next lemma.

▶ **Lemma 3.** *Let T denote a DFS tree of a graph G with root r, then the following holds:*
1. *Every vertex $u \neq r$ is a cut vertex of G exactly if at least one of the edges from u to one of its children is either an unmarked edge or a half marked edge.*
2. *The vertex r is a cut vertex of G exactly if it has at least two children in T.*

Since we want to use the lemma above, we have to mark the tree edges as full marked, half marked or unmarked. Therefore, we run two DFS. In the first run, we classify all tree edges, which we initially unmark. During the second, whenever we discover a back edge $\{w, u\}$ with w being a descendant of u, it becomes evident that both u and w belong to the same biconnected component as do all vertices that are both, descendants of u and ancestors of w, since they induce a cycle C. Let v be the ancestor of w that is child of u. We mark the edge from u to v as half marked and all the other tree edges on C as full marked. If the edge $\{u, v\}$ has already been full marked in a previous step, this will not be overwritten. Note that if $\{u, v\}$ is half marked and u has a back edge to one of its ancestors such that the edge connecting u to its parent p becomes full marked, v and p do not belong to the same biconnected component, but u belongs to both, the biconnected component containing v and the biconnected component containing p. The notion to distinguish between half marked and full marked is to indicate this gap between biconnected components.

After the second DFS each cut vertex can be determined by the edge markings of the tree edges connecting it to its children. For a space-efficient implementation, the first DFS can be taken from [11]. The second one with making the markings is described in the next sections and depends on the used working space.

3.1 Cut Vertices with $O(n + m)$ bits

We use static space allocation to address $O(\deg(v))$ bits for each vertex v of degree $\deg(v)$. Within these bits, we store for every edge $\{u, v\}$ adjacent to v if (1) it is a tree or back edge, (2) u or v is closer to the root of the DFS tree, and (3) its markings in case it is a tree edge. Additionally, using $O(\log deg(v))$ bits, when a vertex v is encountered by the DFS for the first time, we store the position of the tree edge $\{u, v\}$ in the adjacency array of v, where u denotes the parent of v in the DFS tree. This information can later be used, when the DFS retreats from v, such that we can perform the DFS without explicitly having to store the DFS stack. (This idea to obtain a DFS in $O(n + m)$ time using $O(n + m)$ bits was already described by Hagerup et al. [13].) To bound the time of marking the tree edges during the second DFS by $O(n+m)$, we perform the following steps. Whenever we encounter a vertex u for the first time during the second DFS via a tree edge, we scan its entire adjacency array for back edges that lead to descendants w of u. For each such back edge $\{u, w\}$, we perform the following substep. As long as there is a tree edge from w to its parent $v \neq u$ that is not full marked, we mark $\{v, w\}$ as full marked and continue with v becoming the new w. If we encounter an edge $\{v, w\}$ that is already marked as full marked, we terminate the substep without marking any more edges. If at some point the parent v of w becomes u, we mark $\{v, w\}$ as half marked if it has not been full marked, yet, and terminate the substep thereafter.

It is easy to see that this procedure results in the correct markings for every edge. The order in which back edges are worked on assures whenever a tree edge $\{v, w\}$ is already full marked so are all tree edges between v and the child of u. Because we store, for each vertex, the position of the edge that connects it with its parent in T, the time of each substep is at most the number of tree edges that are marked plus additional constant time. Since each tree edge is marked at most twice, once as half marked and once as full marked, and the number of back edges is at most m, altogether we use $O(m)$ time.

3.2 Cut Vertices with $O(n \log \log n)$ bits

The key ideas to make the algorithm more space-efficient are to perform the classification of edges as tree edges or back edges on the fly whenever an edge is explored by the DFS, to use a second stack $\mathcal{U}$, and to apply the stack restoration technique by Elmasry et al. [11] to both stacks. We thus reconsider that paper. To obtain a DFS with a linear running time using $O(n \log \log n)$ bits, the stack is partitioned into $O(\log n)$ segments of $\Theta(n/\log n)$ entries, each consisting basically of a vertex. During the DFS, only the $O(1)$ latest segments are kept in the working memory; the remaining are thrown away. Whenever a segment that was thrown away is needed, a restoration recovers it in a time linear in the number of vertices of the segment. To restore a segment in a time linear to its size, the vertices of the ith segment have a *hue* (value) i. For a slight modification of that algorithm, one can easily see that it is not important that the segments consist of $\Omega(n/\log n)$ vertices stored in the stack, as long as their number is bounded by $O(\log n)$ and the vertices with the same hue form a connected subsequence of the sequence of elements in the stack. Therefore, we build segments not based on the entries of the stack; instead, we define the first $\Theta(n/\log n)$ vertices that are visited by the DFS as the first segment, the next $\Theta(n/\log n)$ vertices as the next segment, etc. The hue values are determined via an extra DFS at the beginning. They are stored in addition to the colors white, gray and black that are used during a "standard" DFS. The space consumption of the algorithm sums up to $O(n)$ for the segments plus $O(n \log \log n)$ bits for the hue.

A normal DFS stack $\mathcal{S}$ contains at any moment during the execution of a DFS the vertices on the path within T from its root to the vertex that is currently explored, which are all the vertices that are currently gray. The depth dp_w of a vertex w is the number of edges connecting the vertices on the path from w to the root r of T that consists solely of tree edges. The second stack $\mathcal{U}$ contains those gray vertices u that have a gray child v such that $e_u = \{u, v\}$ is not full marked. Hence, the vertices in $\mathcal{U}$ denote a subset of the vertices in $\mathcal{S}$. More exactly, each entry of $\mathcal{U}$ is a tuple of a vertex u and its depth within T. When v is explored, v is pushed onto $\mathcal{S}$ and its parent u (together with its depth) is pushed onto $\mathcal{U}$. When the DFS retreats from v, it is popped from $\mathcal{S}$ and u is popped from $\mathcal{U}$ if it is still present in $\mathcal{U}$. A second way for u to be removed from $\mathcal{U}$ is when the edge e_u becomes full marked.

Let w denote the vertex at the top of $\mathcal{S}$. Whenever the DFS tries to explore a vertex u that is gray, then $\{w, u\}$ is a back edge. Under the assumption that we know the depth of every vertex, we can mark edges as follows: While there is a vertex u' on the top of $\mathcal{U}$ whose depth is higher than dp_u, we full mark the edge $e_{u'} = \{u', v'\}$ that connects u' with its gray child v' and pop u' from $\mathcal{U}$. This loop stops if either u becomes the vertex at the top of $\mathcal{U}$ or a vertex with a lower depth than u becomes the top vertex of $\mathcal{U}$. In the first case, we half mark e_u. In the second case, we do not mark the edge $\{u, v\}$ that connects u with its gray child v since u is not on $\mathcal{U}$ because the edge $\{u, v\}$ must have been full marked during the processing of a previous back edge.

We now discuss the computation of cut vertices. When retreating from a vertex v to its parent u that is not the root of T, we check if $\{u, v\}$ is half marked or unmarked. If that is the case, we output u as a cut vertex. For the root vertex r, we maintain a counter that indicates if at least two children of r have been explored during the DFS. If so, r is outputted. Using a bit vector over the vertices, we can avoid outputting a cut vertex more than once.

For the implementation of the algorithm, the remaining problems are maintaining both stacks $\mathcal{S}$ and $\mathcal{U}$ as well as determining the depth of u, whenever processing a back edge $\{w, u\}$ with u being the ancestor of w in the DFS tree T. For the first problem, we store for every vertex its hue and use the stack restoration techniques introduced by Elmasry et al. [11], but use it with the modified size as described above. Restorations of segments in $\mathcal{S}$ and $\mathcal{U}$ are performed independently of each other.

The second problem is more complicated and considered now. Let k be the hue of vertices that we currently process, and let Z be the set consisting of every hue $i \neq k$ that is present in $\mathcal{U}$. Our goal is to store the depth of all vertices in $\mathcal{U}$ with a hue in $\max(Z) \cup \{k\}$ such that it can be addressed by the vertex. The idea is to use one array A addressed with static space allocation to store the depth for all vertices in $\mathcal{U}$ of one segment, i.e., one hue. This allows us to build and destroy the arrays for each hue independently. However, the rank-select structure used by the static space allocation is to slow in its construction since we want a running time of $O(n/\log n)$. Therefore, we define *blocks* where block $i \in \{0, \ldots, \lceil n/\lceil(\log n)/2\rceil\rceil - 1\}$ consist of the vertices $1 + i\lceil(\log n)/2\rceil, \ldots, (i+1)\lceil(\log n)/2\rceil$—the last block may be smaller. In an auxiliary array B addressed with static space allocation, for each non-empty block b, we store a pointer to the first entry in A for a vertex in b. Within each block we use table lookup to find the exact position in A where the depth of a vertex is stored. This allows us to store the depth of the vertices in the upcoming segment in an array with static space allocation. We also restore the depth of the vertices of a segment in $\mathcal{U}$ whenever we restore it.

When processing a back edge $\{u, w\}$ (w having a hue k) with u being the ancestor of w and having a hue i there are two possibilities: If $i \in \max(Z) \cup \{k\}$, then the depth of $\mathcal{U}$ can be determined in constant time. Otherwise, $i < j = \max(Z)$ and, we iteratively restore those segments in $\{j-1, j-2, \ldots, i\}$ that have a vertex in $\mathcal{U}$ and process the vertices within these segments that are present in $\mathcal{U}$ as described above until we finally restore the segment of vertices with hue i together with their depth.

We summarize the space bound as follows. For every vertex we store in $O(n)$ bits its current color (white, gray, black), and if it yet has been outputted as a cut vertex. The set Z can be easily maintained with $O((\log n)^2)$ bits. Using $O(n \log\log n)$ bits we can store for every vertex its hue. We use additional $O(n)$ bits to store a constant number of stack segments of $O(n/\log n)$ vertices each. Since the depths of those vertices are stored compactly in arrays, each such array A together with its auxiliary array B can be implemented with $O(n)$ bits. Moreover, we use two bits for each vertex u on the stack to store if the edge connecting u and its child on the stack, if any, has been half or fully marked. Thus, the overall space bound is $O(n \log\log n)$ bits.

We finally determine the time bounds. As analyzed in [11], the DFS including the stack restorations of $\mathcal{S}$, but without the extra computations for the back edges runs in $O(n+m)$ time as do the intermediate computations in total. Assume for the moment, that we do not throw away and restore segments of $\mathcal{U}$. Then, the total time to mark the tree edges due to the back edges is $O(n+m')$ where m' is the number of back edges since each tree edge is marked at most twice using the stack $\mathcal{U}$. The tests if a vertex is a cut vertex can be performed in total time $O(m)$. It remains to bound the time for the restorations of $\mathcal{U}$. Whenever we restore a segment of $\mathcal{U}$, a hue value is removed from the set Z and never returns. This means that we have only $O(\log n)$ restorations of a segment of $\mathcal{U}$, which can be done in total time $O(n)$.

Combining the algorithms of Section 3.1 and this section, we obtain our first theorem.

▶ **Theorem 4.** *There is an algorithm that, given an n-vertex m-edge graph G in an adjacency array representation with cross pointers, runs in time $O(n+m)$ and uses $O(n + \min\{m, n\log\log n\})$ bits of working space, and determines the cut vertices of G.*

3.3 Biconnected Components

We next show that we can compute the biconnected components of an undirected graph. Recall that a graph is biconnected if it is connected and has no cut vertices. A biconnected component of a graph G is a maximal biconnected induced subgraph of G. Whenever we say in the following theorem and proof that we output an edge $\{x, y\}$, then we mean that we output the index of the edge in the adjacency array of both x and y.

▶ **Theorem 5.** *There is a data structure that, given an n-vertex m-edge graph G in an adjacency array representation with cross pointers, runs $O(n + m)$ initialization time and uses $O(n + m)$ bits of working space and that afterwards, given an edge e, computes the vertices and/or the edges of the biconnected component B of G with B containing e in a time that is linear in the number of vertices and edges that are output.*

Proof. To initialize the data structure, first run our algorithm from Section 3.1 to compute a DFS tree T with a root r, for each edge $\{u, v\}$ the ancestor-descendant relationship in T between u and v as well as the markings of the tree edges. Also store for each vertex $v \neq r$ the index of its edge connecting v to its parent. Then build a rank-select data structure for each vertex v that allows us to iterate over the fully marked tree edges and the back edges to ancestors of v in $O(1)$ time per edge. It is easy to see that the initialization runs in $O(n+m)$ time and all information can be stored with $O(n + m)$ bits using static-space allocation.

Afterwards, given an edge $e = \{u, v\}$ with u being an ancestor of v, we can output the biconnected component B containing e by running a DFS from v and traversing only tree edges (to both directions, parent and children) such that we never explore new vertices from a vertex that was reached by a half marked or unmarked edge and such that we never use a half marked or unmarked edge moving from a parent to its child via such an edge. During the DFS, we output all visited vertices as well as all traversed tree edges. Since a back edge $\{x, y\}$ with x ancestor of y always belongs to the same biconnected component as y, we output $\{x, y\}$ only if we visit y. In this case, we output the edge as an edge of y and, using cross pointers, also of x. The algorithm can easily be modified such that it only outputs the vertices/edges of the biconnected component. Using the rank-select data structures, this can be done in the time stated in the lemma.

To see that each outputted B is indeed a biconnected component observe that B is connected. Assume that B has a cut vertex v with a child u that cuts off the subtree T_u with root u, and we want to output the component containing the edge connecting v and its parent. In that case the edge $\{v, u\}$ is half marked or not marked. Hence, such an edge is not used to go from a parent to a child. On the other hand, if we want to output a component B for which a vertex v is a cut vertex disconnecting B and r, then the edge connecting v with the rest of B is half marked. Hence, we output v, but we do not visit other vertices. Finally, B is maximal by construction since all tree edges that are in B are fully marked except for the "highest" edge in T that is half marked; thus all vertices of B are found by the DFS. ◀

4 Outerplanar Graphs

Outerplanar graphs and maximal outerplanar graphs are well-studied subclasses of planar graphs. For their structural properties we refer to the monograph of Brandstädt et al. [6].

Given a biconnected outerplanar graph $G = (V, E)$, we call an edge that is incident to the outer face an *outer edge* and an edge that is incident to two inner faces an *inner edge*. We now describe a set of well-known properties for outerplanar graphs that help us to describe and prove our algorithm of Section 4.1. Every maximal outerplanar graph with at least three vertices is biconnected and, for every biconnected outerplanar graph G, the set of outer edges induce a unique Hamiltonian cycle that contains all vertices of G. Every biconnected outerplanar graph $G = (V, E)$ with $V = \{1, \dots, n\}$ and $|E| > n$ contains at least one inner edge. Let the vertices of G be labeled according to their position on the Hamiltonian cycle of G, by $1, \dots, n$, and let $\{u, v\}$ denote an inner edge that connects the vertices u and v. W.l.o.g., let $u < v$. Then the graph $G' = G[\{u, \dots, v\}]$ is biconnected and outerplanar. Since $\{u, v\}$ is an inner edge, $1 < v - u < n - 1$ holds and there are exactly $v - u - 1 > 0$ vertices between u and v on the path part of the Hamiltonian cycle that belongs to G'. This path together with the edge $\{v, u\}$ forms the Hamiltonian cycle of G'.

Usually, one decomposes an outerplanar graph by repeatedly removing a vertex v of degree 2. However, this needs to test if the neighbors of v are connected by an edge and, if not, to add such an edge. Because the test is too time consuming and storing all such edges needs too much space, we search instead for a closed or good chain defined next.

We define a *chain* in an outerplanar graph G as either a cycle that consists solely of vertices of degree 2 or a path that contains at least three pairwise distinct vertices with the property that its first and its last vertex have a degree larger than 2 while the rest must have degree 2. We denote the first and the last vertex of a chain as its *endpoints* unless the chain C is a cycle, in which case the endpoints can be chosen arbitrarily as long as they are adjacent to each other. Furthermore, we call a cycle a *loop* if it contains one vertex of degree larger than 2 and all other vertices have a degree 2. A chain is called a *good chain* if one of its endpoints has a degree of at most 4. Let us call a face F *induced* by a chain C if the endpoints u and v of C are adjacent to each other and C together with the edge $\{u, v\}$ is the boundary of F. We denote a chain C that induces a face F as a *closed chain*. For simplicity, we sometimes consider a chain also as a set of edges.

▶ **Lemma 6.** *Let $G = (\{1, \dots, n\}, E)$ denote a biconnected outerplanar graph with $n \geq 4$ vertices. Then G contains a good closed chain C.*

The next two lemmas are used to show the correctness of our algorithm.

▶ **Lemma 7.** *Let $G = (V, E)$ be a biconnected outerplanar graph, then the following properties hold:*

(i) *For every chain C in G with endpoints v_1 and v_2, the graph $G' := G[(E \setminus C) \cup \{\{v_1, v_2\}\}]$ induced by the edge set $(E \setminus C) \cup \{\{v_1, v_2\}\}$ is biconnected and outerplanar.*

(ii) *For all $v_1, v_2 \in V$, there are at most two internal vertex-disjoint paths with at least two edges each.*

Proof. To prove the first part note that G' is a minor of G. Thus, G' is outerplanar. For all vertices u_1, u_2 of G', there are two internal vertex-disjoint paths in G. Since C is a chain, it can only be a complete part of such a path and therefore replaced by the edge $\{v_1, v_2\}$. It follows there are two internal vertex-disjoint paths between u_1 and u_2 in G' and G' is biconnected. We prove the second part by contradiction. Assume that Prop. ii does not hold. Then $K_{2,3}$ is a minor of G. This is a contradiction to G being outerplanar. ◀

▶ **Lemma 8.** *Let $G = (V, E)$ be a graph for which the following properties hold.*

(1) *There is a chain C in G with endpoints v_1 and v_2 such that the graph $G' := G[(E \setminus C) \cup \{\{v_1, v_2\}\}]$ is biconnected and outerplanar.*

(2) *There are at most two internal vertex-disjoint paths with at least 2 edges each in G that connect v_1 and v_2.*

Then G is biconnected and outerplanar.

Proof. The existence of a chain follows from Lemma 6. G' is biconnected outerplanar because of Prop. 1. Because of Prop. 2 there is at most one internal vertex-disjoint path connecting v_1 and v_2 in $G[E \setminus C]$. Hence, the edge $\{v_1, v_2\}$ is part of the outer face of G'. It follows that we can embed C in the outer face of an embedding of G' to yield an outerplanar embedding of G. Because $\{v_1, v_2\}$ is part of the outer face of G', it is part of the Hamiltonian cycle of G'. We can extend the Hamiltonian cycle of G' by replacing $\{v_1, v_2\}$ with C to get a Hamiltonian cycle of G. Thus G is biconnected. ◀

4.1 Our Algorithm on Biconnected Outerplanar Graphs

For the time being, we assume that the given graph is biconnected. Our algorithm works in two phases and can be sketched as follows. Before our actual algorithm starts we test whether $m \leq 2n - 3$. If not, the graph is not outerplanar and we terminate immediately. Otherwise we start our algorithm that modifies the input graph into a smaller outerplanar graph as described in Lemma 8 Prop. 1 while checking Prop. 2 of Lemma 8. Lemma 7 guarantees that it does not matter which chain we take, the modification is always possible and that the check never fails unless G is not biconnected outerplanar. A main obstacle is to handle the edges replacing the chains that we call subsequently *artificial edges*.

To check Prop. 2 of Lemma 8, we keep in both phases counters P_e for every (original or artificial) edge e to count the number of internal vertex-disjoint paths with at least 2 edges that connect the endpoints of e. Whenever we remove a chain with both endpoints in e, we increment P_e. If P_e at any time exceeds 2, the graph can not be outerplanar by Lemma 7 (ii). We check this after every incrementation of an counter P_e and terminate eventually.

We start to sketch the two phases of our algorithm. The details of the phases are given subsequently. Let $G = (V, E)$ denote the input graph, which is located in read-only input space, and G' denote the subgraph of G that we are currently considering in our algorithm. Initially, $G' := G$. Thereafter, within our algorithm the edges of chains are either removed from G' if they induce a face or replaced by artificial edges that connect the endpoints of the chain directly. Consequently, G' is always a minor of G.

The purpose of the first phase is to limit the number of artificial edges that are required for the second phase to $O(n/\log n)$. The first phase consists of $\Theta(\log\log n)$ *rounds*, in each of which we iterate over all chains, but only remove the closed ones.

In the second phase, we repeatedly take a vertex of degree 2 and determine its chain. Depending on the kind of the chain, we proceed: Good closed chains are removed from G'. The edges of chains that turn out to be good, but not closed are replaced by an artificial edge connecting its endpoints. The counter of vertex-disjoint paths with the same endpoints $\{u, v\}$ for a newly created artificial edge $\{u, v\}$ is initialized with 1 to account for the chain that has been replaced by $\{u, v\}$. When processing a chain C that turns out to be not good, we implement a *shortcut* that is a pair of pointers, each one addressed by the vertex of degree 2 in C that is next to a endpoint of C and pointing to the vertex of degree 2 in C that is adjacent to the other endpoint. This way, when the degree of either one of the endpoints is lowered to 2 by the removal of adjacent chains and thus C becomes connected with an other chain, C does not have to be traversed again to check if the new chain is good.

If the input graph is outerplanar, the algorithm terminates either in Phase 1 or in Phase 2 as soon as the last good chain, which is a cycle, is processed and a single edge that is the edge between its endpoints remains. Otherwise, if the input graph is not outerplanar, at some moment one of the checks fails and the algorithm stops and answers that the input is not biconnected outerplanar. Possible conditions for a failed check are if no good chain remains and the graph has at least vertices, if the counter that counts the internal vertex-disjoint paths with at least 2 edges between the endpoints of an edge exceeds 2 for some edge, if some loop is detected, or if a vertex turns out to be incident to at least three vertices of degree 2.

To lower the time that is required to iterate through the adjacency array of a vertex v, we initialize, for each vertex $v \in V$ with degree $deg_G(v)$ in G, a choice dictionary with universe $\{1, \ldots, deg_G(v)\}$ that represents the edges that are adjacent to v and still present within the current subgraph. The choice dictionary contains i with $i \in \{1, \ldots, deg_G(v)\}$ exactly if the ith entry of the adjacency array of v represents an edge of the input graph G that is still present in the current graph G'. Thus, future iterations through the adjacency array of v can be performed within a time that is linear in the number of edges that are incident on v in G'. Hence, the time to determine the adjacency of two endpoints v and w of a chain is bound by $O(\min\{deg_{G'}(v), deg_{G'}(w)\})$.

Phase 1: At the beginning of each round we unmark all vertices and insert all vertices of degree 2 of the current graph G' into a choice dictionary D.

For a more detailed description we subdivide the $\Theta(\log \log n)$ rounds into stages. As long as there is a vertex u in D, we extract it from D and start a new stage. Let C denote the chain that u belongs to. A stage works as follows. If C contains a vertex that is marked as *tried* or C is not closed, the stage stops. Otherwise, we increment the counter P_e for every edge e on C as well as for the edge connecting its endpoints and remove the vertices and edges of C from the graph. If the current chain C is not the first chain processed in the current stage so far, we mark every vertex of degree 2 that has been processed within the current stage and still remains in the graph as *tried*. If one or both of the endpoints of C thereby become a vertex of degree 2, the procedure is repeated immediately for the new chain C' that incorporates both endpoints of C. The stage ends if no new chain is found. At the end of the stage we remove the vertices of degree 2 that have been processed within the current stage from D. A round ends if D is empty.

Phase 2: We start to initialize a choice dictionary D consisting of one vertex of degree 2 for each good closed chain. (During Phase 2, vertices of other chains may be added into D.) Take q as the initial number of vertices in D. As we prove in Corollary 11, the number of chains that induce a face is at most $O(n/\log n)$ after Phase 1 unless G is not outerplanar. If G' contains more such chains, we can terminate immediately. Otherwise, we can store a constant number of artificial edges and shortcuts for each such chain. Whenever we process a vertex u from D, we determine the chain C that u is part of, the endpoints v and w of C, and then distinguish three cases.

- If C is good and closed, we increment the counter P_e for every edge e that is part of C and for the edge that connects its endpoints. Finally, we remove the edges of C from G'.
- If C is good and not closed, we insert an artificial edge e_a that connects the endpoints of C, increment the counter P_e for every edge e in C by 1 and remove all the edges and inner vertices of C from G'. Finally, the counter P_{e_a} of e_a is initialized with 1.
- Otherwise, C is not good. We remove all vertices of C from D and insert a shortcut between the vertices of degree 2 in C that are adjacent to the endpoints of C. If there are old shortcuts connecting other inner vertices of C, they become obsolete and are removed.

Whenever the degree of v or w decreases, we perform the following subroutine. If $deg_{G'}(v) \in \{3, 4\}$, we insert all vertices u of degree 2 that are adjacent to v into D. Note that u and w are neighbors of v on the unique Hamiltonian Cycle of G'. Thus, if G is biconnected outerplanar, there is at most one such vertex u. If $deg_{G'}(v) = 2$, we insert v into D. We proceed with w analogously. We so guarantee that each good closed chain of G' has a vertex in D. As shown by Lemma 12, if the number of artificial edges and shortcuts that are simultaneously in use exceeds $2q$, G is not outerplanar and we stop.

Since the removal of chains is performed in accordance with Lemma 7 and Lemma 8, to show the correctness of the algorithm, it remains to verify that the checks on the counters P_e are correct and sufficient.

▶ **Lemma 9.** *The counter P_e counts for any edge $e = \{v_1, v_2\}$ that belongs at some moment during our algorithm to the current graph G' the number of internal vertex-disjoint paths with at least 2 edges between v_1 and v_2 that have been removed from G so far.*

By our counters P_e for all original and artificial edges, we count the number of internal vertex-disjoint paths between vertices that have been endpoints of a removed chain. Thus, our tests are sufficient by Lemma 8. By Lemma 7 (ii) the counting of the internal vertex-disjoint paths between the endpoints of other edges and terminating if one of these counters exceeds 2 is correct.

4.2 Space-Efficient Implementation and Space Bounds

Before any allocation of space is performed the first check of the algorithm is to verify that the number m of edges within the input graph G is at most $2n - 3$, where n denotes the number of vertices in G. From now on, we assume that $m = O(n)$. During Phase 1 we use a bit vector of n bits that allows us to mark vertices as *tried*.

The current graph G' is represented as follows. We use a choice dictionary of O(n) bits, where i is in the set represented by the choice dictionary exactly if the vertex i is still part of G'. In addition, for each vertex v of initial degree $deg_G(v)$, we store $\Theta(deg_G(v))$ bits. Within this space we use a choice dictionary C_v with universe $\{1, \ldots, deg_G(v)\}$ as already described above and a counter that maintains the current degree of v in G'. It follows that, ignoring artificial edges and shortcuts, a representation of the current graph G' with one choice dictionary for each vertex fits in $O(n)$ bits of working space.

We next want to bound the number of artificial edges and shortcuts. We start to bound the number of chains that induce a face after Phase 1. For this purpose, let us define the *dual tree* T_G of a biconnected and outerplanar graph G as the dual graph of G minus the vertex that represents the outer face of G. Since G is biconnected and outerplanar, T_G is a tree. The leaves of T_G correspond to those faces of G that are induced by chains. Thus, the removal of a closed chain C that induces a face F in G', which results in the merging of F with the outer face, corresponds to the removal of the leaf F in $T_{G'}$.

▶ **Lemma 10.** *If the input graph G with n vertices is biconnected outerplanar, then the number of chains that induce a face in the current graph G' after the tth round of Phase 1 is at most $n/2^t$.*

Proof. It is easy to see that the initial number of chains that induce a face before the first round is at most n. Recall that, in each stage of Phase 1, we consider a vertex u of degree 2 and test whether it is part of a closed chain C within our current graph G'. If so, we remove the chain C from G' and thereafter continue recursively on one endpoint v of C if the removal of C results in v becoming a vertex of degree 2. Let us analyze the modifications of the algorithm in the dual tree. Whenever we remove a chain, this means that we remove a leaf and then recursively try if its parent thereby has become a leaf and can be removed as well until a node of degree at least 2 is encountered.

Vertices of a chain C that is incident to a face F are marked as *tried* if F could not be merged with the outer face after a face that was incident to F has been successfully merged with the outer face by the removal of the chain that induced it. The removal of F fails only if F is not induced by C and there remain at least two faces in $T_{G'}$ that are incident on F. We conclude that F had degree at least 3 at the beginning of the current round. Since in every round leaves are removed until a node is encountered that had at the beginning of the round a degree of at least 3, the number of leaves is at least halved in every round. ◀

▶ **Corollary 11.** *If the input graph G with n vertices is biconnected outerplanar, the number of closed chains after Phase 1 in the current graph G' is $O(n/\log n)$.*

We next bound the number of artificial edges and shortcuts.

▶ **Lemma 12.** *The initial number q of chains at the beginning of Round 2 that have vertices in D only doubles during Phase 2, and the number of artificial edges and shortcuts that are simultaneously in use by the algorithm is $2q = O(n/\log n)$.*

Proof. By the corollary above, $q = O(n/\log n)$ at the beginning of Phase 2. One may consider these chains cutting the Hamilton cycle of G' into q parts. Since we add new vertices into D at the end of Phase 2 only if we were in Case 1 before, a new chain C' is always neighbored to an old chain C, which is then deleted from G' before a vertex of C' is added into D. Roughly speaking, the chains at the beginning of Phase 2 are fires that can spread to new chains on the left and on the right along the Hamilton cycle, but the fire can never return and ends immediately at the moment when a new chain starts to burn. It is easy to see that at most $2q$ chains are on fire. If we define that a chain that is processed with Case 2 or 3 is still on fire, then we have to store artificial edges or shortcuts only for chains that are on fire, and thus, their number is bounded by $2q = O(n/\log n)$. ◀

We can use a bit vector of $O(n)$ bits to store for every vertex v the information whether an artificial edge or shortcut exists at v or not. As a consequence of the last lemma, we can store all artificial edges as well as all shortcuts in a ragged dictionary and the total space bound of the algorithm is $O(n)$ bits.

▶ **Lemma 13.** *There is an algorithm that, given an n-vertex biconnected graph G in an adjacency array representation with cross pointers, runs in $O(n \log\log n)$ time and uses $O(n)$ bits of working space, and determines whether G is biconnected outerplanar.*

4.3 Algorithm for General Outerplanar Graphs

We next sketch the generalization of our recognition algorithm of biconnected outerplanar graphs to general outerplanar graphs. Since a graph is outerplanar exactly if all of its biconnected components are outerplanar, we can iterate over the biconnected components of a given graph G using our framework of Section 3.3—to avoid using $\Omega(m)$ time or bits if a non-outerplanar, dense graph is given, one should initially check that $m \leq 2n - 3$. For each biconnected component, we first stream all its vertices and build an initially empty choice dictionary C_v with $\deg_G(v)$ keys for each such vertex v, then stream the edges of the biconnected component and fill in the indices of each edge in the choice dictionaries of its endpoints. In addition, we compute and store for each vertex of the biconnected component its degree.

Recall that our subgraph G' is represented by a choice dictionary that contains those vertices of G that are still part of G' and a choice dictionary C_v for every vertex v such that $i \in \{1, \ldots, deg_G(v)\}$ is present in C_v exactly if the ith entry of the adjacency array of v contains an edge that is still present G'. We initialize these choice dictionaries with the values that are streamed from the algorithm that determines biconnected components to initialize a representation of a biconnected subgraph. The time to test if the subgraph is biconnected outerplanar is bound by the number of vertices and edges of the subgraph alone.

Finally note that a check if a outerplanar graph $G = (V, E)$ is maximal outerplanar can be easily performed by checking if $2|V| - 3 = |E|$.

▶ **Theorem 14.** *There is an algorithm that, given an n-vertex graph G in an adjacency array representation with cross pointers, runs in time $O(n \log\log n)$ and uses $O(n)$ bits of working space, and determines if G is (maximal) outerplanar.*

References

1 Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Reprint of: Memory-constrained algorithms for simple polygons. *Comput. Geom. Theory Appl.*, 47(3, Part B):469–479, 2014. `doi:10.1016/j.comgeo.2013.11.004`.

2 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $O(n)$ bits. In *Proc. 25th International Symposium on Algorithms and Computation (ISAAC 2014)*, volume 8889 of *LNCS*, pages 553–564. Springer, 2014. `doi:10.1007/978-3-319-13075-0_44`.

3 Tetsuo Asano, Wolfgang Mulzer, Günter Rote, and Yajun Wang. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.*, 2(1):46–68, 2011.

4 Luis Barba, Matias Korman, Stefan Langerman, Rodrigo I. Silveira, and Kunihiko Sadakane. Space-time trade-offs for stack-based algorithms. In *Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPIcs*, pages 281–292. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. `doi:10.4230/LIPIcs.STACS.2013.281`.

5 Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991. `doi:10.1137/0220017`.

6 A. Brandstädt, V. Le, and J. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999. `doi:10.1137/1.9780898719796`.

7 David Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1996.

8 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

9 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

10 Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for *st*-connectivity on the NNJAG model. *SIAM J. Comput.*, 28(6):2257–2284, 1999. `doi:10.1137/S0097539795295948`.

11 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient basic graph algorithms. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *LIPIcs*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. `doi:10.4230/LIPIcs.STACS.2015.288`.

12 Torben Hagerup and Frank Kammer. Succinct choice dictionaries. *Computing Research Repository (CoRR)*, arXiv:1604.06058 [cs.DS], 2016.

13 Torben Hagerup, Frank Kammer, and Moritz Laudahn. Space-efficient euler partition and bipartite edge coloring. Talk given on the theory days of computer science of the Gesellschaft für Informatik in Speyer, Germany, 2015.

14 Frank Kammer, Dieter Kratsch, and Moritz Laudahn. Space-efficient biconnected components and recognition of outerplanar graphs. *Computing Research Repository (CoRR)*, arXiv:1606.04679 [cs.DS], 2016.

15 Sandra L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Inf. Process. Lett.*, 9(5):229–232, 1979. doi:10.1016/0020-0190(79)90075-9.

16 J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12(3):315–323, 1980. doi:10.1016/0304-3975(80)90061-4.

17 Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, pages 264–268. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743455.

18 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, September 2008. doi:10.1145/1391289.1391291.

19 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

20 Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010.

21 Manfred Wiegers. Recognizing outerplanar graphs in linear time. In *Proc. International Workshop on Graphtheoretic Concepts in Computer Science (WG 1986)*, volume 246 of *LNCS*, pages 165–176. Springer, 1986. doi:10.1007/3-540-17218-1_57.

Multi-Party Protocols, Information Complexity and Privacy

Iordanis Kerenidis[*1], **Adi Rosén**[†2], **and Florent Urrutia**[‡3]

1 **CNRS and Université Paris Diderot, Paris, France**
jkeren@liafa.univ-paris-diderot.fr
2 **CNRS and Université Paris Diderot, Paris, France**
jadiro@liafa.univ-paris-diderot.fr
3 **CNRS and Université Paris Diderot, Paris, France**
urrutia@liafa.univ-paris-diderot.fr

Abstract

We introduce the new measure of *Public Information Complexity* (PIC), as a tool for the study of multi-party computation protocols, and of quantities such as their communication complexity, or the amount of randomness they require in the context of information-theoretic private computations. We are able to use this measure directly in the natural asynchronous message-passing *peer-to-peer* model and show a number of interesting properties and applications of our new notion: the Public Information Complexity is a lower bound on the Communication Complexity and an upper bound on the Information Complexity; the difference between the Public Information Complexity and the Information Complexity provides a lower bound on the amount of randomness used in a protocol; any communication protocol can be compressed to its Public Information Cost; an explicit calculation of the zero-error Public Information Complexity of the k-party, n-bit Parity function, where a player outputs the bit-wise parity of the inputs. The latter result establishes that the amount of randomness needed for a private protocol that computes this function is $\Omega(n)$.

1998 ACM Subject Classification F.0 [Theory of computation] General, E.4 Data, Coding and Information Theory

Keywords and phrases multi-party protocols, information theory, communication complexity, multi-party private computation (MPC), randomness

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.57

1 Introduction

Communication complexity, originally introduced by Yao [44], is a prolific field of research in theoretical computer science that yielded many important results in various fields. Informally, it answers the question "How many bits must the players transmit to solve a distributed problem ?" The study of the two-party case has produced a large number of interesting and important results, upper and lower bounds, with many applications in other areas in theoretical computer science such as circuit complexity, data structures, streaming algorithms and distributed computation (see, e.g., [35, 36, 24, 40, 23]).

* Research supported in part by ERC grant QCC and ANR grant RDAM.
† Research supported in part by ANR grant RDAM.
‡ Research supported in part by ERC grant QCC and ANR grant RDAM.

A powerful tool recently introduced for the study of two-party communication protocols is the measure of *Information Complexity* (or *cost*). This measure, originally defined in [1] and [14], extends the notions of information theory, originally introduced by Shannon [42], to interactive settings. It quantifies, roughly speaking, the amount of information about their respective inputs that Alice and Bob must leak to each other in order to compute a given function f of their inputs. Information complexity (IC) has been used in a long series of papers to prove lower bounds on communication complexity and other properties of (two-party) communication protocols (e.g., [2, 3, 11, 6]). An interesting property of information complexity is that it satisfies a direct sum. The *direct sum* question, one of the most interesting questions in complexity theory, asks whether solving n independent copies of the same problem must cost (in a given measure) n times the cost of solving a single instance. In the case of communication complexity, this question has been studied in, e.g., [20, 14, 41, 31, 29, 3, 33, 30] and in many cases it remains open whether a direct sum property holds.

Another important question in communication complexity is the relation between the information complexity of a function and its communication complexity. We would like to know if is it possible to compute a function by sending a number of bits which is not (too much) more than the information the protocol actually has to reveal. Put differently, is it always possible to *compress* the communication cost of a protocol to its information cost? For the two-party case it is known that perfect compression is not possible [26, 27]. Still, several interesting compression results are known. The equality between information cost and amortized communication cost is shown in [11, 6], and other compression techniques are given in [3, 4, 12, 37]. It remains open if one can compress interactive communication up to some small loss (for example logarithmic in the size of the input).

When trying to study the *multi-party* (i.e., where at least 3 players are involved) communication setting using similar information-theoretic methods, such as IC, one encounters a serious problem. The celebrated results on information-theoretic private computation [5, 18] state that if the number of players is at least 3, then *any function* can be computed by a randomized protocol such that *no information* about the inputs is revealed to the other players (other than what is implied by the value of the function and their own input). Thus, in the multi-party case, the IC of any function f is 0 (or only the entropy of f, depending on the definition of IC), and cannot serve to study multi-party protocols.

For this reason, information complexity has rarely been used in the multi-party setting, where most results have been obtained via combinatorial techniques. Among the interesting works on multi-party setting are [38, 43] which introduce the techniques of *symmetrization* and *composition*, and [16, 17] which study the influence of the topology of the network. One notable exception is the work of Braverman et al. [7] which studies the *set-disjointness* problem using information theoretic tools. Braverman et al. provide almost tight bounds in the so-called coordinator model (that differs from the more natural peer-to-peer model) by analyzing the information leaked between the players but also the information obtained by the coordinator itself. The set disjointness problem is maybe one of the most extensively studied problem in communication complexity (cf. [6, 2, 13, 28, 32, 9]). This line of research was followed by [15] which also uses information complexity to obtain tight bounds on the communication complexity of the function *Tribes* in the coordinator model. Information complexity is also used in [10] to study set-disjointness in the broadcast model.

A number of sub-models have been considered in the literature for the multi-party computation protocols setting: the *number in hand model* (NIH), where each player has a private input, is maybe the most natural one, while in the *number on the forehead* model

(NOF), each player i knows all inputs x_j, $j \neq i$, i.e., the "inputs" of all players except its own. As to the communication pattern, a number of variants exist as well: in the *blackboard* model, the players communicate by broadcasting messages (or writing them on a "blackboard"); in the *message passing* model, each pair of players is given a private channel to mutually communicate (for more details on different variants see [35]). Most of the results obtained in multi-party communication complexity were obtained for the NOF model and/or the blackboard model. The present paper studies, however, the NIH, message passing (peer to peer) model, which is also the most closely related to the work done on message passing protocols in the distributed computing and networking communities.

1.1 Our contributions

Our main goal is to introduce novel information-theoretical measures for the study of number in hand, message-passing multi-party protocols, coupled with a natural model that, among other things, allows private protocols (which is not the case for, e.g., the coordinator model)

We define the new measure of *Public Information Complexity* (PIC), as a tool for the study of multi-party computation protocols, and of quantities such as their communication complexity, or the amount of randomness they require in the context of information-theoretic private computations. Intuitively, our new measure captures a combination of the amount of information about the inputs that the players leak to other players, and the amount of randomness that the protocol uses. By proving lower bounds on PIC for a given multi-party function f, we are able to give lower bounds on the communication complexity of f and on the amount of randomness needed to privately compute f. The crucial point is that the PIC of functions, in our multi-party model, is not always 0, unlike their IC.

Our new measure works in a model which is a slight restriction of the most general asynchronous model, where, for a given player at a given time, the set of players from which that player waits for a message can be determined by that player's own local view. This allows us to have the property that for any protocol, the information which is leaked during the execution of the protocol is at most the communication cost of the protocol. Note that in the multi-party case, the information cost of a protocol may be higher than its communication cost, because the identity of the player from which one receives a message might carry some information. This is another issue when trying to use IC in the peer-to-peer multi-party setting. We are able to define our measure and use it directly in a natural asynchronous *peer-to-peer* model (and not, e.g., in the coordinator model used in most works studying the multi-party case, c.f. [19]). The latter point is particularly important when one is interested in privacy, since our model allows for private protocols, while this is not necessarily the case for other models, including the coordinator model. Furthermore, if one seeks to accurately understand the natural peer-to-peer model, suppressing polylog-factor inaccuracies, one has to study directly the peer-to-peer model (see the comparison of models in subsection 2.1).

We go on to show a number of interesting properties and applications of our new notion:

- The Public Information Complexity is a lower bound on the Communication Complexity and an upper bound on the Information Complexity. In fact, it can be strictly larger than the Information Complexity.
- The difference between the Public Information Complexity and the Information Complexity provides a lower bound on the amount of randomness used in a protocol.
- We compress any communication protocol to their PIC (up to logarithmic factors), by extending to the multi-party setting the work of Brody et al. [12] and Pankratov [37].
- We show that one can approach the central question of direct sum in communication complexity by trying to prove a direct sum result for PIC. Indeed, we show that a direct sum property for PIC implies a certain direct sum property for communication complexity.

- We explicitly calculate the zero-error *Public Information Complexity* of the k-party, n-bit Parity function (Par), where a player outputs the bit-wise parity of the inputs. We show that the PIC of this function is $n(k-1)$. This result is tight and it also establishes that the amount of randomness needed for a private protocol that computes this function is $\Omega(n)$. While this sounds a reasonable assertion no previous proof for such claim existed.

The paper is organized as follows. In Section 2 we define the communication model and a number of traditional complexity measures. In Section 3 we define the new measure PIC that we introduce in the present paper, and in Section 4 we discuss its relation to randomness and multi-party private computation. In section 5 we discuss the existence of a direct sum property for PIC, and in Section 6 we give tight bounds for Par using PIC. We conclude the paper in Section 7. All proofs are deferred to the full version of the paper.

2 The model

We start by defining a number of notations. We denote by k the number of players. We often use n to denote the size (in bits) of the input to each player. Calligraphic letters will be used to denote sets. Upper case letters will be used to denote random variables, and given two random variables A and B, we will denote by AB the joint random variable (A, B). Given a string (of bits) s, $|s|$ denotes the length of s. Using parentheses we denote an ordered set (family) of items, e.g., (Y_i). Given a family (Y_i), Y_{-i} denotes the sub-family which is the family (Y_i) *without* the element Y_i. The letter X will usually denote the input to the players, and we thus use the shortened notation X for (X_i), *i.e.* the input to all players. π will be used to denote a protocol. We use the term *entropy* to talk about binary entropy.

We now define a natural communication model which is a slight restriction of the most general asynchronous peer-to-peer model. Its restriction is that for a given player at a given time, the set of players from which that player waits for a message can be determined by that player's own local view. This allows us to define information theoretical tools that pertain to the transcripts of the protocols, and at the same time to use these tools as lower bounds for communication complexity. This restriction however does not exclude the existence of private protocols, as other special cases of the general asynchronous model do. We observe that without such restriction the information revealed by the execution of a protocol might be higher than the number of bits transmitted and that, on the other hand, practically all multi-party protocols in the literature are implicitly defined in our model. We also compare our model to the general one and to other restricted ones and explain the usefulness and logic of our specific model.

2.1 Definition of the model

We work in the *multi-party number in hand peer-to-peer* model. Each player has unbounded local computation power and, in addition to its input X_i, has access to a source of private randomness R_i. We will use the notation R for (R_i), *i.e.*, the private randomness of all players. A source of public randomness R^p is also available to all players. The system consists of k players and a family of k functions $f = (f_i)_{i \in [\![1,k]\!]}$, with $\forall\ i \in [\![1,k]\!], f_i : \prod_{l=1}^{k} \mathcal{X}_l \to \mathcal{Y}_i$, where $\mathcal{X}_l$ denotes the set of possible inputs of player l, and $\mathcal{Y}_i$ denotes the set of possible outputs of player i. The players are given some input $x = (x_i) \in \prod_{i=1}^{k} \mathcal{X}_i$, and for every i, player i has to compute $f_i(x)$. Each player has a special write-only output tape.

We define the communication model as follows, which is the asynchronous setting, with some restrictions. To make the discussion simpler we assume a global time which is *unknown* to the players. Every pair of players is connected by a bidirectional communication link that allows them to send messages in both directions. There is no bound on the delivery time of a message, but every message is delivered in finite time, and the communication link maintains FIFO order in each of the two directions. Given a specific time we define the *view* of player i, denoted D_i, as the input of that player, X_i, its private randomness, R_i, and the messages received so far by player i. The protocol of each player i runs in *local* rounds. In each round, player i sends messages to some subset of the other players. The identity of these players, as well as the content of these messages, depend on the current view of player i. The player also decides whether to write a (nonempty) string on its output tape. Then, the player waits for messages from a certain subset of the other players, where this subset is also determined by the current view of the player. Then the (local) round of player i terminates[1]. To make it possible for the player to identify the arrival of the *complete* message that it waits for, we require that each message sent by a player in the protocol is self-delimiting.

Denote by D_i^j the view of player i at the end of local round j, $j \geq 0$, where the beginning of the protocol is considered round 0. Formally, a protocol π is defined by a sequence of functions for each player i, parametrized by the *local* round j, $j \geq 1$:

- $\overline{S_i^j} : D_i^{j-1} \to 2^{\{1,\ldots,k\}\setminus\{i\}}$, defining the set of players to which player i sends the messages.
- $m_{i,q}^j : D_i^{j-1} \to \{0,1\}^*$, for all $q \in \overline{S_i^j}(D_i^{j-1})$, defining the content of the messages player i sends. Each such message has to be self-delimiting.
- $O_i^j : D_i^{j-1} \to \{0,1\}^*$, defining what the player writes on the output tape. Each player can write on its output tape a non-empty string only once.[2]
- $S_i^j : D_i^{j-1} \to 2^{\{1,\ldots,k\}\setminus\{i\}}$, defining the set of players from which player i waits for a message.

We note that the model does not impose "coherence" between the players. That is, the model does not preclude the possibility that a certain player waits indefinitely for a message that is never sent to it.

We define the transcript of the protocol of player i, denoted Π_i, as the concatenation of the messages read by player i from the links of the sets $S_i^1, S_i^2, \ldots$, ordered by local round number, and within each round by, say, the index of the player. We denote by $\overleftrightarrow{\Pi}_i$ the concatenation of Π_i together with a similar concatenation of the messages sent by player i to the sets $\overline{S}_i^0, \overline{S}_i^1, \ldots$. We denote by $\Pi_{i \to j}$ the concatenation of the messages sent by player i to player j during the course of the protocol. The transcript of the (whole) protocol, denoted Π, is obtained by concatenating all the Π_i ordered by, say, player index.

We will give most of the definitions for the case where all functions f_i are the same function, that we denote by f. The definitions in the case of family of functions are similar.

▶ **Definition 1.** For $\epsilon \geq 0$, a protocol π *ϵ-computes* a function f if for all $x \in \prod_{i=1}^{k} \mathcal{X}_i$:

1. For all possible assignments for the random sources R_i, $1 \leq i \leq k$, and R^p, every player eventually (i.e., in finite time) writes on its output tape (a non-empty string).

[1] The fact that the receiving of the incoming messages comes as the last step of the (local) round comes only to emphasize that the sending of the messages and the writing on the output tape are a function of only the messages received in previous (local) rounds.

[2] We require that each player writes only once on its output tape so that the local view of the player determines the local output of the protocol (i.e., so that players itself "knows" the output). This requirement is needed since a player may not know locally that the protocol ended.

2. With probability at least $1 - \epsilon$ (over all random sources) the following event occurs: each player i writes on its output tape the value $f(x)$, i.e., the correct value of the function.

For simplicity we also assume that a protocol must eventually stop. That is, for all possible inputs and all possible assignments for the random sources, eventually (i.e., in finite time) there is no message in transit.

2.2 Comparison to other models

The somewhat restricted model (compared to the general asynchronous model) that we work with allows us to define a measure similar to information cost that we will later show to have desirable properties and to be of use. Notice that the general asynchronous model is problematic in this respect since one bit of communication can bring $\log(k)$ bits of information, as not only the content of the message but also the identity of the sender may reveal information. Thus, information cannot be used as a lower bound on communication. In our case, the sets S_i^l and $\overline{S_i^l}$ are determined by the current view of the player, (Π_i) contains only the content of the messages, and thus the desirable relation between the communication and the information is maintained. On the other hand, our restriction is natural, does not seem to be very restrictive (practically all protocols in the literature adhere to our model), and does not exclude the existence of private protocols.

To exemplify the above mentioned issue in the general asynchronous model consider the following simple example of a deterministic protocol, for 4 players A, B and C, D, which allows A to transmit to B its input bit x, but where all messages sent in the protocol are the bit 0, and the protocol generates only a single transcript over all possible inputs.

A: If $x = 0$ send 0 to C; after receiving 0 from C, send 0 to D.
If $x = 1$ send 0 to D; after receiving 0 from D, send 0 to C

B: After receiving 0 from a player, send 0 back to that player.

C,D: After receiving 0 from A send 0 to B. After receiving 0 from B send 0 to A.

It is easy to see that B learns the value of x from the order of the messages it gets.

There has been a long series of works about multi-party communication protocols in different variants of models, for example [13, 28, 32, 38, 16, 17]. In [7], Braverman et al. consider a restricted class of protocols working in the *coordinator model*: an additional player with no input can communicate privately with each player, and the players can only communicate with the coordinator.

We first note that the coordinator model does not yield exact bounds for the multi-party communication complexity in the peer-to-peer model (neither in our model nor in the most general one). Namely, a protocol in the peer-to-peer model can be transformed into a protocol in the coordinator model with an $O(\log k)$ multiplicative factor in the communication complexity, by sending any message to the coordinator with a $O(\log k)$-bit label indicating its destination. This factor is sometimes necessary, e.g., for the q`-index` function, where players P_i, $0 \leq i \leq k-1$, each holds an input bit x_i, player P_k holds q indices $0 \leq j_\ell \leq k-1$, $1 \leq \ell \leq q$, and P_k should learn the vector $(x_{j_1}, x_{j_1}, \ldots, x_{j_q})$: in the coordinator model the communication complexity of this function is $\Theta(\min\{k, q \log k\})$, while in both peer-to-peer models there is a protocol for this function that sends only (at most) $\min\{k, 2q\}$ bits, where P_k just queries the appropriate other players. But this multiplicative factor between the complexities in the two models is not always necessary: the communication complexity of the parity function Par is $\Theta(k)$ both in the peer-to-peer models and in the coordinator model.

Moreover, when studying private protocols in the peer-to-peer model, the coordinator model does not offer any insight. In the (asynchronous) coordinator model, described in [19] and used for instance in [7], if there is no privacy requirement with respect to the coordinator, it is trivial to have a private protocol by all players sending their input to the coordinator, and the coordinator returning the results to the players. If there is a privacy requirement with respect to the coordinator, then if there is a random source shared by all the players (but not the coordinator), privacy is always possible using the protocol of [21]. If no such source exists, privacy is impossible in general. This follows from the results of Braverman et al. [7] who show a non-zero lower bound on the total internal information complexity of all parties (including the coordinator) for the function *Disjointness* in that model.

Note also that the private protocols described in [5, 18] (and further work) are defined in the synchronous setting, and thus can be adapted to our communication model (the sets $\overline{S_i^j}$ and S_i^j are always all the players and hence even independent of the current views).

In the sequel we also use a special case of our model, where the sets $\overline{S_i^j}$ and S_i^j are a function only of i and j, and not of the entire current view of the player. This is a natural special case for protocols which we call *oblivious protocols*, where the communication pattern is fixed and is not a function of the input or randomness. Clearly, the messages themselves remain a function of the view of the players. This model also allows for private protocols.

2.3 Communication complexity and information complexity

Communication complexity, introduced in [44], measures how many bits of communication are needed in order for a set of players to compute with error ϵ a given function of their inputs. The allowed error ϵ, implicit in many of the contexts, will be written explicitly as a superscript when necessary.

▶ **Definition 2.** The *communication cost* of a protocol π, $\mathsf{CC}(\pi)$, is the maximal length of the transcript of π over all possible inputs, private randomness and public randomness.

▶ **Definition 3.** $\mathsf{CC}(f)$ denotes the communication cost of the best protocol computing f.

Information complexity measures the amount of information that must be transmitted so that the players can compute a given function of their joint inputs. One of its main uses is to provide a lower bound on the communication complexity of the function. In the two-party setting, this measure led to interesting results on the communication complexity of various functions, such as *AND* and *Disjointness*. We now focus on designing an analogue to the information cost, for the multi-party setting. The notion of internal information cost for two-party protocols (c.f. [14, 2, 6]) can be easily generalized to any number of players:

▶ **Definition 4.** The *internal information cost* of a protocol π for k players, with respect to input distribution μ, is the sum of the information revealed to each player about the inputs of the other players: $\mathsf{IC}_\mu(\pi) = \sum_{i=1}^{k} I(X_{-i}; \Pi_i \mid X_i R_i R^p)$.

Intuitively, the information cost of a protocol is the amount of information each player learns about the inputs of the other players during the protocol. The definition we give above, when restricted to two players is the same as in [6], even though they look slightly different. This is because we explicit the role of the randomness, which will allow us to later bound the amount of randomness needed for private protocols in the multi-party setting.

The *internal information complexity* of a function f with respect to input distribution μ, as well as the *internal information complexity* of a function f, can be defined for the multi-party case based on the information cost of a protocol, just as in the 2-party case.

▶ **Definition 5.** The *internal information complexity* of a function f, with respect to input distribution μ, is the infimum of the internal information cost over all protocols computing f on input distribution μ: $\mathsf{IC}_\mu(f) = \inf_{\pi \text{ computing } f} \mathsf{IC}_\mu(\pi)$.

▶ **Definition 6.** The *internal information complexity* of a function f is the infimum, over all protocols π computing f, of the information cost of π when run on the worst input distribution for that protocol: $\mathsf{IC}(f) = \inf_{\pi \text{ computing } f} \sup_{\mu} \mathsf{IC}_\mu(\pi)$.

▶ **Proposition 7** ([11])**.** For any protocol π and input distribution μ, $\mathsf{CC}(\pi) \geq \mathsf{IC}_\mu(\pi)$. Thus, for any function f, $\mathsf{CC}(f) \geq \mathsf{IC}(f)$.

The information revealed to a given player by a protocol can be written in several ways:

▶ **Proposition 8.** For any protocol π, for any player i:

$$I(X_{-i}; \overleftrightarrow{\Pi_i} \mid X_i R^p) = I(X_{-i}; \overleftrightarrow{\Pi_i} \mid X_i R_i R^p) = I(X_{-i}; \Pi_i \mid X_i R_i R^p) \quad \text{and}$$

$$I(X_{-i}; \overleftrightarrow{\Pi_i} \mid X_i R^p f(X)) = I(X_{-i}; \overleftrightarrow{\Pi_i} \mid X_i R_i R^p f(X)) = I(X_{-i}; \Pi_i \mid X_i R_i R^p f(X)).$$

2.4 Information complexity and privacy

The definition of a *private protocol* as defined in [5, 18] is the following.

▶ **Definition 9.** A k-player protocol π for computing a family of functions (f_i) is *private*[3] if for every player $i \in [\![1, k]\!]$, for all pair of inputs $x = (x_1, \ldots, x_k)$ and $x' = (x'_1, \ldots, x'_k)$, such that $f_i(x) = f_i(x')$ and $x_i = x'_i$, for all possible private random tapes r_i of player i, and all possible public random tapes r^p, it holds that for any transcript T
$\Pr[\Pi_i = T \mid R_i = r_i \ ; \ X = x \ ; \ R^p = r^p] = Pr[\Pi_i = T \mid R_i = r_i \ ; \ X = x' \ ; \ R^p = r^p]$,
where the probability is over the randomness R_{-i}.

The notion of privacy has an equivalent formulation in terms of information.

▶ **Proposition 10.** A protocol π is private iff for all μ, $\sum_{i=1}^{k} I(X_{-i}; \Pi_i \mid X_i R_i R^p f_i(X)) = 0$.

It is well known that in the multi-party number-in-the-hand peer-to-peer setting (for $k \geq 3$), unlike in the two-party case, any function can be privately computed.

▶ **Theorem 11** ([5],[18])**.** *Any function of more than two variables can be privately computed.*

Using the above theorem, we can give the following lemma.

▶ **Lemma 12.** *For any family of functions (f_i) of more than two variables and any μ, $\mathsf{IC}_\mu(f) \leq \sum_{i=1}^{k} H(f_i(X))$, where X is distributed according to μ.*

This lemma shows that IC cannot be used in the multi-party setting for any meaningful lower bounds on the communication complexity, since its value is always upper bounded by the entropies of the functions. Our goal is to get lower bounds tight in both k and n. For this reason, we introduce a new information theoretic quantity for the multi-party setting.

[3] In this paper we consider only the setting of 1-privacy, which we call here for simplicity, privacy.

3 The new measure: Public Information Cost

We now introduce a new information theoretic quantity which can be used instead of IC in the multi-party setting. The notion we define will be suitable for studying multi-party communication in a model which is only a slight restriction on the general asynchronous model, and which allows for private protocols. This means that while IC will be at most the entropies of the functions, our new notion remains a strong lower bound for communication.

▶ **Definition 13.** For any protocol π and any μ, the *public information cost* of π is:

$$\mathsf{PIC}_\mu(\pi) = \sum_{i=1}^{k} I(X_{-i}; \Pi_i R_{-i} \mid X_i R_i R^p).$$

The difference between PIC and IC is the presence of the other parties private coins, R_{-i}, in the formula. If π is a protocol using only public randomness, then for any input distribution μ, $\mathsf{PIC}_\mu(\pi) = \mathsf{IC}_\mu(\pi)$, and hence the name public information cost.

The public information cost measures both the information about the inputs learned by the players and the information that is hidden by the use of private coins. It can be decomposed, using the chain rule, into two terms, making explicit the contribution of the internal information cost and of the private randomness of the players.

▶ **Proposition 14.** For any π and any μ, $\mathsf{PIC}_\mu(\pi) = \mathsf{IC}_\mu(\pi) + \sum_{i=1}^{k} I(R_{-i}; X_{-i} | X_i \Pi_i R_i R^p)$.

The meaning of the second term is the following. At the end of the protocol, player i knows its input X_i, its private coins R_i, the public coins R^p and its transcript Π_i. Suppose that the private randomness R_{-i} of the other players is now revealed to player i. This brings to it some new information $I(R_{-i}; X_{-i} | X_i \Pi_i R_i R^p)$ about the inputs X_{-i} of the other players.

We also define the public information complexity of a function.

▶ **Definition 15.** For any f and any μ, $\mathsf{PIC}_\mu(f) = \inf_{\pi \text{ computing } f} \mathsf{PIC}_\mu(\pi)$.

▶ **Definition 16.** For any f, we define the quantity $\mathsf{PIC}(f) = \inf_{\pi \text{ computing } f} \sup_{\mu} \mathsf{PIC}_\mu(\pi)$.

The public information cost is a lower bound for the communication complexity.

▶ **Proposition 17.** For any π and μ, $\mathsf{CC}(\pi) \geq \mathsf{PIC}_\mu(\pi)$. Thus, for any f, $\mathsf{CC}(f) \geq \mathsf{PIC}(f)$.

In fact, as we show below, the public information cost of a function is equal to its internal information cost in a setting where only public randomness is allowed. The role of private coins in communication protocol has been studied for example in [8, 12, 34]. In the next section we will see that the difference between the public information cost and the information cost is related to the private coins used during the protocol.

▶ **Theorem 18.** *For any function f and input distribution μ,*

$$\mathsf{PIC}_\mu(f) = \inf_{\pi \textit{ computing } f,\textit{ using only public coins}} \mathsf{IC}_\mu(\pi)\ ,\ \textit{and}$$

$$\mathsf{PIC}(f) = \inf_{\pi \textit{ computing } f,\textit{ using only public coins}} \sup_{\mu} \mathsf{IC}_\mu(\pi).$$

The following property of the public information cost will be useful for zero-error protocols.

▶ **Proposition 19.** For any function f, for any input distribution μ, $\mathsf{PIC}^0_\mu(f) = \mathsf{IC}^{\det}_\mu(f)$ where $\mathsf{IC}^{\det}_\mu(f) = \inf_{\pi \text{ deterministic protocol computing } f} \mathsf{IC}_\mu(\pi)$.

PIC and IC are strictly different even in the two party case. We prove below that for the AND function, the public information cost is $\log 3 \simeq 1.58$, while, as shown in [9], $\mathsf{IC}^0(\mathsf{AND}) \simeq 1.49$. This implies that the protocol that achieves the optimal information cost for AND must use private coins. We remark also that in [9] it is shown that the external information cost of AND, that we do not consider here, is $\log(3)$.

▶ **Proposition 20.** For two players, $\mathsf{PIC}^0(\mathsf{AND}) = \log(3) \simeq 1.58$.

4 Private computation, randomness, and PIC

We have seen that the public information cost of a function is equal to the information cost of the function when we only consider public coin protocols, and that in order to decrease the information cost even further, the players must use private randomness. We will see now that the difference between the public information cost of a protocol and its information cost can provide a lower bound on the amount of private randomness the players use during the protocol. The entropy of the transcript of the protocol, conditioned on the inputs and the public coins, is defined as $H(\Pi \mid XR^p)$. Once the input and the public coins are fixed, the entropy of the transcript of the protocol comes solely from the private randomness. Thus it provides a lower bound on the entropy of the private randomness used by the players.

▶ **Theorem 21.** *Let $f = (f_i)$ be a family of functions of k variables. Let π be a protocol for f. For any input distribution μ, it holds: $H_\mu(\Pi \mid XR^p) \geq \dfrac{\mathsf{PIC}_\mu(\pi) - \mathsf{IC}_\mu(\pi)}{k-1}$. Thus running a protocol for f with information cost I_μ requires entropy $H_\mu(\Pi \mid XR^p) \geq \dfrac{\mathsf{PIC}_\mu(f) - I_\mu}{k-1}$.*

5 A direct sum for PIC ?

The *direct sum* property is a fundamental question in complexity theory, and has been studied for many computation models. A direct sum theorem affirms that the amount of resources needed to perform t independent tasks is at least the sum of the resources needed to perform each of the t tasks. In this section we show that a direct sum property for PIC implies a direct sum property for CC. For this, we prove a compression result by extending [12, 37] to the multi-party case. Note that information complexity has a direct sum property in the multi-party case. For PIC, it is easy to prove the following inequality.

▶ **Theorem 22.** *For any k-variable functions f and g, for any μ on inputs of f, for any η on inputs of g, $\mathsf{PIC}_{\mu\times\eta}(f \times g) \leq \mathsf{PIC}_\mu(f) + \mathsf{PIC}_\eta(g)$.*

In order to understand whether the opposite inequality holds, i.e., whether a direct sum property holds for PIC, we first need to study the problem of compressing communication.

5.1 Relation between PIC and CC: A compression result

An important open question is how well we can compress the communication cost of an interactive protocol. Compression results have appeared in [3, 11, 12, 37, 4], while, on the other hand, [25, 27, 39, 22, 26] focus on the hardness of compressing communication protocols. Here, we present a compression result with regards to the average-case communication complexity and the public information cost.

▶ **Definition 23.** The *average-case communication complexity* of a protocol π with respect to the input distribution μ, denoted $\mathsf{ACC}_\mu(\pi)$, is the expected number of bits that are transmitted in an execution of π for inputs distributed according to μ and uniform randomness.

▶ **Theorem 24.** *Suppose there exists a protocol π to compute a k-variable function f over the distribution μ with error probability ϵ. Then there exists a public-coin protocol ρ that computes f over μ with error $\epsilon + \delta$, and with average communication complexity*

$$\mathsf{ACC}_\mu(\rho) = \mathcal{O}\left(k^2\mathsf{PIC}_\mu(\pi)\log(\mathsf{CC}(\pi))\left(\log(k\mathsf{CC}(\pi)) + \log\frac{k^2\mathsf{PIC}_\mu(\pi)\log(\mathsf{CC}(\pi))}{\delta}\right)\right).$$

The proof of the above theorem will follow from extending the compression result presented in [12, 37] to the case of $k > 2$ players.

▶ **Theorem 25.** *Suppose there exists a public coin protocol π to compute a k-variable function f over the distribution μ with error probability ϵ. Then there exists a public-coin protocol ρ that computes f over μ with error $\epsilon + \delta$, and with average communication complexity*

$$\mathsf{ACC}_\mu(\rho) = \mathcal{O}\left(k^2\mathsf{IC}_\mu(\pi)\log(\mathsf{CC}(\pi))\left(\log(k\mathsf{CC}(\pi)) + \log\frac{k^2\mathsf{IC}_\mu(\pi)\log(\mathsf{CC}(\pi))}{\delta}\right)\right).$$

Theorems 25 and 18, which make the link between the public information cost of general protocols and the information cost of public coins protocol, imply theorem 24.

In the two-party compression scheme of [12, 37], the two players, given their own input, try to guess the transcript $\pi(x_1, x_2)$ of the protocol π. For this, player 1 picks a candidate t_1 from the set $\mathrm{Im}(\pi(x_1, \cdot))$ of possible transcripts knowing that it has input x_1, while player 2 picks a candidate t_2 from the set $\mathrm{Im}(\pi(\cdot, x_2))$. The two players then communicate in order to find the first bit on which t_1 and t_2 disagree. The general structure of protocols ensures that the common prefix of t_1 and t_2 (until the first bit of disagreement) is identical to the beginning of the correct transcript on inputs x_1 and x_2, i.e., identical to $\pi(x_1, x_2)$. Starting from this correct prefix, the players then pick new candidates for the transcript of the protocol $\pi(x_1, x_2)$, and so on, until they agree on the full transcript $\pi(x_1, x_2)$. Clever choices of the candidates, along with an efficient technique to find the first bit which differs between the candidates, lead to a protocol with little communication.

In extending the proof in [12, 37] to the multi-party case, new difficulties occur. The players can no longer try to guess the full transcript, as they have little information about the conversation between the other players, but can only try to guess their partial transcripts, according to their own input. Then, in order to find the first disagreement in the global transcript, every pair of players needs to find and communicate the place of the first disagreement in their partial transcripts. This induces the k^2 factor in our compression scheme. It is unclear if this is necessary. Moreover, the players lack a common reference time. To solve this, we will introduce, as a technical tool in the proof, a *coordinator*, whose role is to introduce a round structure in the protocol π. Note that this is only in the proof, and that the stated results hold in the model we define in Section 2.

5.2 A direct sum for PIC implies a direct sum for CC

▶ **Theorem 26.** *Given a k-variable function f and a distribution μ on inputs of f, if the existence of a protocol π computing $f^{\otimes t}$ with error $\epsilon \geq 0$ implies that there exists a protocol π' computing f with error ϵ and verifying $\mathsf{PIC}_\mu(\pi') \leq \frac{1}{t}\mathsf{PIC}_{\mu^{\otimes t}}(\pi)$, $\mathsf{CC}(\pi') \leq \mathsf{CC}(\pi)$, then*

$$\mathsf{CC}^{2(\epsilon+\delta)}(f) = \mathcal{O}\left(\frac{k^2}{t(\epsilon+\delta)}\mathsf{CC}^\epsilon(f^{\otimes t})\log(\mathsf{CC}^\epsilon(f^{\otimes t}))\log(k)\log\frac{k^2\mathsf{CC}^\epsilon(f^{\otimes t})\log(\mathsf{CC}^\epsilon(f^{\otimes t}))}{\delta}\right).$$

Note that the result of this theorem is meaningful when t is large with respect to k.

6 Tight lower bounds for the parity function Par

We now show how one can indeed use PIC to study multi-party communication protocols and to prove tight bounds. We study one of the canonical problems for zero-error multi-party computation, the parity function. The k-party parity problem with n-bit inputs Par_k^n is defined as follows. Each player i receives n bits $(x_i^p)_{p\in[\![1,n]\!]}$ and Player 1 has to output the bitwise XOR of the inputs $\left(\bigoplus_{i=1}^{k} x_i^1, \bigoplus_{i=1}^{k} x_i^2, \ldots, \bigoplus_{i=1}^{k} x_i^n\right)$.

There is a simple private protocol for Par_k^n that uses n bits of private randomness. Player 1 uses a private random n-bit string r and sends to Player 2 the string $x_1 \oplus r$. Then, Player 2 computes the bit-wise parity of its input with the message and sends $x_2 \oplus x_1 \oplus r$ to Player 3. The players continue until Player 1 receives back the message $x_k \oplus \ldots \oplus x_1 \oplus r$. Player 1 then takes the bit-wise parity of this message with the private string r to compute the value of the parity function. It is easy to see that this protocol has information cost equal to n, since Player 1 just learns the value of the function and all other players learn nothing. We thus see that information cost cannot provide here lower bounds that scale with k.

We now prove tight lower bounds for this problem using the measure of PIC. For this, we study a restricted class of protocols: we only consider protocols such that for any player i, the sets $(S_i^l)_l$ and $(\overline{S}_i^l)_l$ do not depend on the input x or on the randomness. In other words, the structure of the protocol is fixed and independent of the input and randomness. Note that the private protocol we described above fits in this model.

Our bound for Par_k^n is in fact proved for a wider class of protocols, where we allow the player outputting $\oplus_{i=1}^k x_i^p$ to be different for each coordinate p and to depend on the input.

▶ **Theorem 27.** $\mathsf{PIC}_\mu^0(\mathsf{Par}_k^n) \geq n(k-1)$ *where μ is the uniform input distribution.*

▶ **Theorem 28.** *The entropy in the private randomness of a private protocol for Par_k^n is at least $\dfrac{k-2}{k-1}n$.*

Where the last theorem follows from Theorems 27 and 21. Using Theorem 21 we can also give a lower bound on the randomness one needs for protocols that are allowed to leak some limited amount of information about the inputs of the players.

7 Conclusions

In this paper we introduce a new information-theoretic measure, that we call PIC, for the study of multi-party computation protocols in the number-in-hand, peer-to-peer model. This is probably the most natural (distributed) computation model, and also closely related to the models used in the distributed algorithms community. Previous information-theoretic measures that were used successfully for the study of two-party computation protocols do not extend immediately to the multi-party case due to the fact that private protocols exist for any function in the multi-party setting [5, 18]. Our notion of PIC provides an alternative way of studying multi-party protocols, especially when one is interested in notions of privacy. Furthermore, PIC may yield tight results for certain functions, for which using other models, such as the coordinator model, would imply a loss of a logarithmic factor.

We define this measure in a slightly restricted computation model which however still allows private protocols, and applies to almost any protocol in the literature. We prove a number of properties of our new measure, PIC, and a number of connections to other complexity notions, e.g., the amount of randomness needed for private computation or the central question of direct sum (in communication complexity).

Our work opens the way to interesting directions for further work. A challenging direction would be to prove a tight lower bound for Disjointness in the message passing peer-to-peer model (without the loss of a logarithmic factor). An even more ambitious goal would be to use our result from Section 5 to try and prove the direct sum property for communication complexity, in either the two-party or multi-party setting, via our measure of PIC.

References

1 Reuven Bar-Yehuda, Benny Chor, Eyal Kushilevitz, and Alon Orlitsky. Privacy, additional information and communication. *IEEE Transactions on Information Theory*, 39(6):1930–1943, 1993. doi:10.1109/18.265501.

2 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS'02, pages 209–218, Washington, DC, USA, 2002. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=645413.652164.

3 Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. How to compress interactive communication. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC'10, pages 67–76, New York, NY, USA, 2010. ACM. doi:10.1145/1806689.1806701.

4 Balthazar Bauer, Shay Moran, and Amir Yehudayoff. Internal compression of protocols to entropy. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 481–496. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. URL: http://www.dagstuhl.de/dagpub/978-3-939897-89-7, doi:10.4230/LIPIcs.APPROX-RANDOM.2015.481.

5 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC'88, pages 1–10, New York, NY, USA, 1988. ACM. doi:10.1145/62212.62213.

6 Mark Braverman. Interactive information complexity. In *Proceedings of the 44th symposium on Theory of Computing*, STOC'12, pages 505–524, New York, NY, USA, 2012. ACM. doi:10.1145/2213977.2214025.

7 Mark Braverman, Faith Ellen, Rotem Oshman, Toniann Pitassi, and Vinod Vaikuntanathan. A tight bound for set disjointness in the message-passing model. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 668–677. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.77.

8 Mark Braverman and Ankit Garg. Public vs private coin in bounded-round information. 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 502–513. Springer, 2014. doi:10.1007/978-3-662-43948-7_42.

9 Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein. From information to exact communication. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC'13, pages 151–160, New York, NY, USA, 2013. ACM. doi:10.1145/2488608.2488628.

10 Mark Braverman and Rotem Oshman. On information complexity in the broadcast model. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Sym-*

posium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015, pages 355–364. ACM, 2015. doi:10.1145/2767386.2767425.

11 Mark Braverman and Anup Rao. Information equals amortized communication. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 748–757, Washington, DC, USA, 2011. IEEE Computer Society. doi:10.1109/FOCS.2011.86.

12 Joshua Brody, Harry Buhrman, Michal Koucký, Bruno Loff, Florian Speelman, and Nikolay K. Vereshchagin. Towards a reverse newman's theorem in interactive information complexity. In *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, pages 24–33. IEEE, 2013. doi:10.1109/CCC.2013.12.

13 Amit Chakrabarti, Subhash Khot, and Xiaodong Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *In IEEE Conference on Computational Complexity*, pages 107–117, 2003.

14 Amit Chakrabarti, Yaoyun Shi, Anthony Wirth, and Andrew Chi-Chih Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *FOCS*, pages 270–278, 2001. doi:10.1109/SFCS.2001.959901.

15 Arkadev Chattopadhyay and Sagnik Mukhopadhyay. Tribes is hard in the message passing model. 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 224–237. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. URL: http://www.dagstuhl.de/dagpub/978-3-939897-78-1, doi:10.4230/LIPIcs.STACS.2015.224.

16 Arkadev Chattopadhyay, Jaikumar Radhakrishnan, and Atri Rudra. Topology matters in communication. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 631–640, 2014. doi:10.1109/FOCS.2014.73.

17 Arkadev Chattopadhyay and Atri Rudra. The range of topological effects on communication. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 540–551. Springer, 2015. doi:10.1007/978-3-662-47666-6_43.

18 David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC'88, pages 11–19, New York, NY, USA, 1988. ACM. doi:10.1145/62212.62214.

19 Danny Dolev and Tomás Feder. Multiparty communication complexity. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 428–433. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63514.

20 Tomás Feder, Eyal Kushilevitz, Moni Naor, and Noam Nisan. Amortized communication complexity. *SIAM J. Comput.*, 24(4):736–750, 1995. doi:10.1137/S0097539792235864.

21 Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC'94, pages 554–563, New York, NY, USA, 1994. ACM. doi:10.1145/195058.195408.

22 Lila Fontes, Rahul Jain, Iordanis Kerenidis, Sophie Laplante, Mathieu Laurière, and Jérémie Roland. Relative discrepancy does not separate information and communication complexity. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*,

volume 9134 of *Lecture Notes in Computer Science*, pages 506–516. Springer, 2015. doi:10.1007/978-3-662-47672-7_41.

23 Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. Networks cannot compute their diameter in sublinear time. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1150–1162. SIAM, 2012. doi:10.1137/1.9781611973099.

24 Anna Gál and Parikshit Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. *SIAM J. Comput.*, 39(8):3463–3479, 2010. doi:10.1137/090770801.

25 Anat Ganor, Gillat Kol, and Ran Raz. Exponential separation of information and communication. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 176–185, 2014. doi:10.1109/FOCS.2014.27.

26 Anat Ganor, Gillat Kol, and Ran Raz. Exponential separation of communication and external information. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:88, 2015. URL: http://eccc.hpi-web.de/report/2015/088.

27 Anat Ganor, Gillat Kol, and Ran Raz. Exponential separation of information and communication for boolean functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 557–566. ACM, 2015. doi:10.1145/2746539.2746572.

28 Andre Gronemeier. Asymptotically optimal lower bounds on the nih-multi-party information complexity of the and-function and disjointness. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 505–516. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. doi:10.4230/LIPIcs.STACS.2009.1846.

29 Prahladh Harsha, Rahul Jain, David A. McAllester, and Jaikumar Radhakrishnan. The communication complexity of correlation. *IEEE Transactions on Information Theory*, 56(1):438–449, 2010. doi:10.1109/TIT.2009.2034824.

30 Rahul Jain. New strong direct product results in communication complexity. *J. ACM*, 62(3):20, 2015. doi:10.1145/2699432.

31 Rahul Jain, Jaikumar Radhakrishnan, and Pranab Sen. A direct sum theorem in communication complexity via message compression. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 300–315. Springer, 2003. doi:10.1007/3-540-45061-0_26.

32 T. S. Jayram. Hellinger strikes back: A note on the multi-party information complexity of and. In *Proceedings of the 12th International Workshop and 13th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX'09 / RANDOM'09, pages 562–573, Berlin, Heidelberg, 2009. Springer-Verlag. doi:10.1007/978-3-642-03685-9_42.

33 Hartmut Klauck. A strong direct product theorem for disjointness. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 77–86. ACM, 2010. doi:10.1145/1806689.1806702.

34 Alexander Kozachinskiy. *Computer Science – Theory and Applications: 10th International Computer Science Symposium in Russia, CSR 2015, Listvyanka, Russia, July*

13-17, 2015, Proceedings, chapter Making Randomness Public in Unbounded-Round Information Complexity, pages 296–309. Springer International Publishing, Cham, 2015. `doi:10.1007/978-3-319-20297-6_19`.

35 Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, 1997.

36 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC'95, pages 103–111, New York, NY, USA, 1995. ACM. `doi:10.1145/225058.225093`.

37 Denis Pankratov. *Communication complexity and information complexity.* PhD thesis, The university of Chicago, 2015.

38 Jeff M. Phillips, Elad Verbin, and Qin Zhang. Lower bounds for number-in-hand multiparty communication complexity, made easy. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'12, pages 486–501. SIAM, 2012. URL: `http://dl.acm.org/citation.cfm?id=2095116.2095158`.

39 Anup Rao and Makrand Sinha. Simplified separation of information and communication. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:57, 2015. URL: `http://eccc.hpi-web.de/report/2015/057`.

40 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. *CoRR*, abs/1011.3049, 2010. URL: `http://arxiv.org/abs/1011.3049`.

41 Ronen Shaltiel. Towards proving strong direct product theorems. *Computational Complexity*, 12(1-2):1–22, 2003. `doi:10.1007/s00037-003-0175-x`.

42 C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

43 David P. Woodruff and Qin Zhang. An optimal lower bound for distinct elements in the message passing model. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 718–733. SIAM, 2014. `doi:10.1137/1.9781611973402.54`.

44 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, STOC'79, pages 209–213, New York, NY, USA, 1979. ACM. `doi:10.1145/800135.804414`.

Dividing by Zero – How Bad Is It, Really?*

Takayuki Kihara[1] **and Arno Pauly**[2]

1 **Department of Mathematics, University of California, Berkeley, USA**
`kihara@math.berkeley.edu`

2 **Département d'Informatique, Université Libre de Bruxelles, Belgium**
`Arno.M.Pauly@gmail.com`

Abstract

In computable analysis testing a real number for being zero is a fundamental example of a non-computable task. This causes problems for division: We cannot ensure that the number we want to divide by is not zero. In many cases, any real number would be an acceptable outcome if the divisor is zero - but even this cannot be done in a computable way.

In this note we investigate the strength of the computational problem *Robust division*: Given a pair of real numbers, the first not greater than the other, output their quotient if well-defined and any real number else. The formal framework is provided by Weihrauch reducibility. One particular result is that having later calls to the problem depending on the outcomes of earlier ones is strictly more powerful than performing all calls concurrently. However, having a nesting depths of two already provides the full power. This solves an open problem raised at a recent Dagstuhl meeting on Weihrauch reducibility.

As application for *Robust division*, we show that it suffices to execute Gaussian elimination.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases computable analysis, Weihrauch reducibility, recursion theory, linear algebra

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.58

1 Introduction

We cannot divide by zero! is probably the first mathematical impossibility statement everyone encounters. In the setting we see it first, arithmetic of concrete integers, this does not cause any problems: Since it is obvious whether some number is zero or not, we simply refrain from attempting it – and the multiplicative absorption of 0 ensures that we have no reason for an attempt anyway. As our mathematical world expands to include more kinds of numbers and variables, we may have to introduce case distinctions at times in order to avoid this problem[1].

In most practical situations, this may seem unproblematic. However, a fundamental observation by Brouwer in the early development of constructive mathematics was that *we cannot in general decide whether a real number is zero or not.* Thus, a case distinction based on whether our intended denominator is zero or not is not constructive. In a constructive setting, we can only divide by a number we know to be different from zero.

* The first author was partially supported by a Grant-in-Aid for JSPS fellows. The second author was partially supported by the ERC inVEST (279499) project.

[1] Forgetting about these cases has probably caused a lot of anguish to pupils learning the outcome of their exams.

To consider a concrete example where we might want to divide by a number that could be zero, consider $a, b \in \mathbb{R}$ with $0 \leq a \leq b$, and the linear equation $a = bx$. We know that there is a solution $x_0 \in [0,1]$: If $b \neq 0$, then $x_0 := \frac{a}{b}$, otherwise $b = a = 0$, and any x works. We see that we do not actually care about whether $b = 0$ or not, and we do not even need any particular outcome of a misguided attempt to calculate $\frac{0}{0}$ – any number would do.

Unfortunately, the algorithm to divide a real number a by a real number b starts with searching for a rational number bounding b away from 0. If no such number exists, there will be no output at all, rather than some arbitrary number. The *robust division* we would like to employ to solve linear equations as above is not actually computable.

In this note, we study the extent of non-computability of robust division in the formal setting of Weihrauch reducibility. Some results had already been obtained in [17]. We will recall that robust division lies strictly in between the traditional non-constructive principles LLPO and LPO and some other basic properties. Our concern then is with the question how multiple uses of robust division interact. We show that sequential uses of robust division cannot be reduced to parallel uses – however, it suffices to have a nesting depths of 2.

In [17], finding the solution to systems of linear inequalities via a modified Fourier-Motzkin elimination, and finding Nash equilibria in bimatrix games were explored as applications of robust division. Here, we shall consider Gaussian elimination as additional example.

An extended version of this article is available as [15].

2 Background

Computability on the reals and other represented spaces

The long history of studying computability on the real numbers presumably goes back to Borel [2] (see [1] for a detailed historical picture). Here, we follow the school of Weihrauch [28]. Computability is initially introduced over $\{0,1\}^{\mathbb{N}}$ by means of Type-2 machines. These are obtained from the usual Turing machine model via a simple modification: The head on output tape can move to the right only (and in particular does so whenever a symbol is written), and the machines never halt. The restriction on the output tape ensures that as the computation proceeds, longer and longer finite prefixes of the ultimate infinite output are available.

The transfer of computability from $\{0,1\}^{\mathbb{N}}$ to the spaces of actual interest is achieved via the notion of a represented space. For a more detailed introduction to the theory of represented spaces, we refer to [21]. A represented space is a pair $\mathbf{X} = (X, \delta_X)$ of a set X and a partial surjection $\delta_X :\subseteq \{0,1\}^{\mathbb{N}} \to X$ (the representation).

A multi-valued function[2] between represented spaces is a multi-valued function between the underlying sets. For $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ and $F :\subseteq \{0,1\}^{\mathbb{N}} \to \{0,1\}^{\mathbb{N}}$, we call F a realizer of f (notation $F \vdash f$), iff $\delta_Y(F(p)) \in f(\delta_X(p))$ for all $p \in \operatorname{dom}(f\delta_X)$.

$$\begin{tikzcd}
\{0,1\}^{\mathbb{N}} \arrow[r, "F"] \arrow[d, "\delta_{\mathbf{X}}"] & \{0,1\}^{\mathbb{N}} \arrow[d, "\delta_{\mathbf{Y}}"] \\
\mathbf{X} \arrow[r, "f"] & \mathbf{Y}
\end{tikzcd}$$

A map between represented spaces is called computable (continuous), iff it has a computable (continuous) realizer. Note that a priori, the notion of continuity for maps between represented

[2] For a discussion of the notion of a multi-valued function, and in particular the difference to the notion of a relation, we refer to [23], [20].

spaces differs from topological continuity. For the admissible represented spaces (in the sense of [24]), the two notions do coincide, if a represented space is equipped with the final topology inherited from Cantor space along the representation. All representations we are concerned with in this note are admissible.

Before we introduce the standard representation of the real numbers, we fix some standard enumeration $\nu_\mathbb{Q} : \mathbb{N} \to \mathbb{Q}$ of the rationals. Now we define $\rho :\subseteq \{0,1\}^\mathbb{N} \to \mathbb{R}$ via $\rho(0^{n_0}10^{n_1}1\ldots) = x$ iff $\forall i \in \mathbb{N}\ |\nu_\mathbb{Q}(n_i) - x| < 2^{-n}$. Note that using e.g. the binary or decimal expansion would not have worked satisfactorily[3]. The choice of ρ ensures that, informally spoken, every naturally encountered continuous function on the reals will be computable.

The naturals are represented in the obvious way by $\delta_\mathbb{N}(0^n1^\mathbb{N}) = n$. The finite spaces $\{0,\ldots,n\}$ are just the corresponding subspaces of $\mathbb{N}$. Likewise, we introduce the represented space $[0,1]$ as a subspace of $\mathbb{R}$. The space $\mathbb{S}$ has the elements $\{\bot,\top\}$ represented via $\delta_\mathbb{S}(0^\mathbb{N}) = \bot$ and $\delta_\mathbb{S}(p) = \top$ for $p \neq 0^\mathbb{N}$.

For any represented space $\mathbf{X}$, there is a canonical definition of the represented space $\mathcal{A}(\mathbf{X})$ of closed subsets of $\mathbf{X}$. We only require this for the specific choices of $\mathbf{X} = [0,1], \{0,\ldots,n\}$: In the former case, a closed subset is a closed subset in the usual sense, and it is represented by a list of rational open balls exhausting its complement in $[0,1]$. In the latter, any subset of $\{0,\ldots,n\}$ is an element of $\mathcal{A}(\{0,\ldots,n\})$, and a set A is represented by $p \in \{0,1\}^\mathbb{N}$ iff 01^k0 occurs somewhere in p iff $k \notin A$ for any $k \in \{0,\ldots,n\}$.

As there are canonical tupling functions $\langle\ldots\rangle : (\{0,1\}^\mathbb{N})^n \to \{0,1\}^\mathbb{N}$ available, we can define products of represented spaces in a straight-forward way. We obtain binary and countable disjoint unions by $(\delta_0 + \delta_1)(0p) = \delta_0(p)$ and $(\delta_0 + \delta_1)(1p) = \delta_1(p)$, and $(\coprod_{i\in\mathbb{N}} \delta_i)(0^n1p) = \delta_n(p)$. We will iterate the binary product, starting with the convention $\mathbf{X}^0 = \{0\}$ and setting $\mathbf{X}^{n+1} = \mathbf{X}^n \times \mathbf{X}$. Finally, $\mathbf{X}^*$ is shorthand for $\coprod_{i\in\mathbb{N}} \mathbf{X}^i$.

Weihrauch reducibility

Weihrauch reducibility is a computable many-one reduction comparing multi-valued functions between represented spaces. So $f \leq_\mathrm{W} g$ informally means that f could be computed with the help of a single oracle-call to g.

▶ **Definition 1.** Let $f :\subseteq \mathbf{X} \rightrightarrows \mathbf{Y}$ and $g :\subseteq \mathbf{U} \rightrightarrows \mathbf{V}$ be partial multivalued functions between represented spaces. Say that f is Weihrauch reducible to g, in symbols $f \leq_\mathrm{W} g$, if there are computable functions $K :\subseteq \{0,1\}^\mathbb{N} \times \{0,1\}^\mathbb{N} \to \{0,1\}^\mathbb{N}$ and $H :\subseteq \{0,1\}^\mathbb{N} \to \{0,1\}^\mathbb{N}$ such that whenever G is a realizer of g, the function $F := (p \mapsto K(p, G(H(p))))$ is a realizer for f.

Based on earlier work by Weihrauch [26, 27], Weihrauch reducibility was suggested as a framework for computable metamathematics in [5, 4] (see also [12, 17]). We point to the introduction of [6] for a recent overview on the development of the field so far.

We shall denote the set of Weihrauch degrees by $\mathfrak{W}$, and point out some operations on them. As shown in [18], the binary product $\times$, the binary disjoint union $\sqcup$, the countable disjoint union $\coprod$ and the operation * all can be lifted from represented spaces via multivalued functions between represented spaces to Weihrauch degrees. $\mathfrak{W}$ is a distributive lattice, and $\sqcup$ is the join. However, no non-trivial countable suprema exist in $\mathfrak{W}$ as shown in [13]. In particular, $\coprod$ is not the countable join.

Informally, $f \sqcup g$ means that both f and g are available for use, but the user has to decide for each instance on one of the two to call. A call to $f \times g$ means making two independent

[3] As already noted by Turing [25].

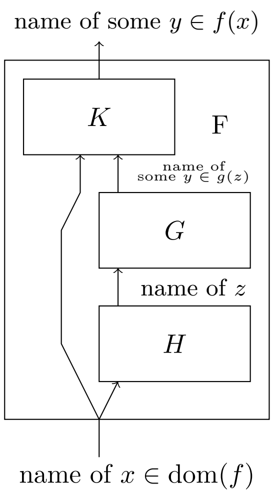

name of some $y \in f(x)$
K
F
name of some $y \in g(z)$
G
name of z
H
name of $x \in \text{dom}(f)$

Figure 1 Illustrating Definition 1.[4]

calls, one to f and one to g. Using f^* means that we first decide on some number $n \in \mathbb{N}$, and then make n independent calls to f.

We want to use a further operation; corresponding to first making a call to some g and then a call to f depending on the outcome of the call to g. In [9, 7] the operation $\star$ was defined as $f \star g := \max_{\leq_W}\{f' \circ g' \mid f' \leq_W f \wedge g' \leq_W g\}$. Here the maximum is understood to range over all f', g' with types such that $f' \circ g'$ is well-defined. While it is not obvious that this maximum exists, an explicit construction is provided in [11]. Informally, an input to $f \star g$ consists of an input to g together with a multivalued function computing some input to f from an output to g. The output is the output of g together with the output of f.

We iterate both $\times$ and $\star$: $f^0 = f^{(0)} = \text{id}_{\{0,1\}^\mathbb{N}}$, and $f^{n+1} = f^n \times f$ and $f^{(n+1)} = f^{(n)} \star f$.

Special Weihrauch degrees

We will refer to a number of well-studied specific Weihrauch degrees in this paper. We shall first recall the degrees LPO and LLPO from [27], the Weihrauch degree counterparts to the Brouwerian counterexamples in intuitionistic mathematics. $\text{LPO} : \{0,1\}^\mathbb{N} \to \{0,1\}$ maps $0^\mathbb{N}$ to 1 and each $p \neq 0^\mathbb{N}$ to 0. This map is equivalent to the characteristic function of 0 in $\mathbb{R}$ or $[0,1]$, and we will not distinguish these maps. The map $\text{LLPO} : \{0,1\}^\mathbb{N} \rightrightarrows \{0,1\}$ outputs 0 if the first 1 in the input occurs at an even position, and 1 if it occurs at an odd position. If the input was $0^\mathbb{N}$, both 0 and 1 are valid outputs. An alternative characterization would be $\text{LLPO} : \mathbb{R} \times \mathbb{R} \to \{0,1\}$ with $0 \in \text{LLPO}(0,y)$, $1 \in \text{LLPO}(x,0)$ and $0,1 \in \text{LLPO}(x,y)$ if $x \neq 0 \neq y$. It holds that $\text{LLPO} <_W \text{LPO}$.

A bountiful source of calibrating principles is found in the closed choice principles and their restrictions:

▶ **Definition 2.** $\text{C}_\mathbf{X} :\subseteq \mathcal{A}(\mathbf{X}) \rightrightarrows \mathbf{X}$, $\text{dom}(\text{C}_\mathbf{X}) = \{A \in \mathcal{A}(\mathbf{X}) \mid A \neq \emptyset\}$, $x \in \text{C}_\mathbf{X}(A) \Leftrightarrow x \in A$.

Thus, closed choice is the task to find a point in a given non-empty closed set. As being non-empty is merely promised rather than constructively witnessed, this task is generally not computable: As long as there is more than one remaining choice, whenever we start

[4] This figure was taken from [22].

outputting some point we might learn next that this point is not in the closed set after all (e.g. by reading some rational ball containing it in the name of the closed set).

These principles have been extensively studied [4, 3, 10, 9, 16, 6]. Depending on the topological properties of the space $\mathbf{X}$ and potentially restrictions to certain subsets, these principles have been revealed to be useful in characterizing many other principles.

Most relevant for us are the principles $\mathrm{C}_{\{0,\ldots,n\}}$. It was shown in [17] that $\mathrm{C}_{\{0,\ldots,n\}} \leq_{\mathrm{W}} \mathrm{C}_{\{0,1\}}^n$, and it follows from the independent choice theorem in [3] that $\mathrm{C}_{\{0,1\}}^n \leq_{\mathrm{W}} \mathrm{C}_{\{0,1\}}^{(n)} \leq_{\mathrm{W}} \mathrm{C}_{\{0,\ldots,2^n-1\}}$. It is quite easy to see that $\mathrm{C}_{\{0,1\}} \equiv_{\mathrm{W}} \mathrm{LLPO}$.

We make passing references to $\mathrm{C}_{\mathbb{N}}$ (and use that $\mathrm{C}_{\mathbb{N}} \equiv_{\mathrm{W}} \mathrm{C}_{\mathbb{N}} \star \mathrm{C}_{\mathbb{N}}$ by the independent choice theorem from [3]), to $\mathrm{CC}_{[0,1]}$, the restriction of $\mathrm{C}_{[0,1]}$ to connected subsets and to $\mathrm{PCC}_{[0,1]}$, the restriction of $\mathrm{C}_{[0,1]}$ to connected sets with positive Lebesgue measure. We also mention $\star$-WWKL from [6], which is $\coprod_{n\in\mathbb{N}}(2^{-n})$-WWKL, where ε-WWKL is the restriction of $\mathrm{C}_{\{0,1\}^{\mathbb{N}}}$ to sets with measure at least ε. By $\mathrm{UC}_{\mathbf{X}}$ we denote the restriction of $\mathrm{C}_{\mathbf{X}}$ to singletons [3]. Finally, $\mathrm{C}_{\sharp=2}$ and $\mathrm{C}_{\sharp\leq 2}$ from [16] are the restrictions of $\mathrm{C}_{\{0,1\}^{\mathbb{N}}}$ to sets with cardinality 2 and at most 2 respectively.

3 Robust division

We consider two variants of robust division: In one case, we know an upper bound on the result, in the second, we do not. Modulo the rescaling, the first case corresponds to knowing that the denominator is at least as big as the numerator.

▶ **Definition 3** ([17]). Define $\mathrm{rDiv} : \mathbb{R} \times \mathbb{R} \rightrightarrows [0,1]$ via $\frac{\min\{|x|,|y|\}}{|y|} \in \mathrm{rDiv}(x,y)$ iff $y \neq 0$ and $z \in \mathrm{rDiv}(x,0)$ for all $x \in \mathbb{R}, z \in [0,1]$.

To simplify notation, we will usually assume that inputs (x,y) for rDiv already satisfy $0 \leq x \leq y$, so that $\frac{\min\{|x|,|y|\}}{|y|} = \frac{x}{y}$ holds.

▶ **Definition 4.** Define $\mathrm{ubrDiv} : \mathbb{R}\times\mathbb{R} \rightrightarrows \mathbb{R}$ by $\frac{x}{y} \in \mathrm{ubrDiv}(x,y)$ iff $y \neq 0$ and $z \in \mathrm{ubrDiv}(x,0)$ for all $x, z \in \mathbb{R}$.

It turns out that the case distinction on $y \neq 0$ or $y = 0$ is equivalent to the unbounded case ubrDiv. Thus, we do not need to investigate ubrDiv as an independent basic operation. Note that the following proof also establishes that it makes no difference for the degree of ubrDiv if the result is presumed to be non-negative.

▶ **Proposition 5.** $\mathrm{ubrDiv} \equiv_W$ *LPO.*

Proof. The direction $\mathrm{ubrDiv} \leq_{\mathrm{W}} \mathrm{LPO}$ follows from computability of division where well-defined and the definition.

For the other direction, note that given some $p \in \{0,1\}^{\mathbb{N}}$ we can compute $x, y \in \mathbb{R}$ such that if $p \neq 0^{\mathbb{N}}$, then $\frac{x}{y} = \min\{n \in \mathbb{N} \mid p(n) = 1\}$ and $x = y = 0$ if $p = 0^{\mathbb{N}}$. Furthermore, there is a computable multivalued retract $\tau : \mathbb{R} \rightrightarrows \mathbb{N}$, so we may pretend that the output of $\mathrm{ubrDiv}(x,y)$ is a natural number n indicating the position of the first 1 in p, if it exists. Given this number, we can then check whether $p(n)$ is 1 or not, which in turn determines the answer to $\mathrm{LPO}(p)$. ◀

The bounded variant of robust division was already established as a *new* degree in [17]. We recall some results on this degree from the literature before continuing its investigation.

▶ **Proposition 6** ([17]).

1. $C_{\{0,1\}} <_W \mathrm{rDiv} <_W LPO$.
2. $\mathrm{rDiv} <_W CC_{[0,1]}$.
3. $\mathrm{rDiv} \not\leq_W C^*_{\{0,1\}}$.

▶ **Proposition 7** ([6, Theorem 16.3, Corollary 16.5 & Theorem 16.6]).

1. $\mathrm{rDiv} <_W PCC_{[0,1]}$.
2. rDiv *is join-irreducible.*
3. $\mathrm{rDiv} \not\leq_W *\text{-}WWKL$.

The preceding results from [6] intuitively state that there is a mechanism to solve rDiv in a probabilistic way with positive probability and error detection. However, there is no way to obtain a positive lower bound on the probability of solving a given instance correctly.

It is a well-known phenomenon in the study of Weihrauch reducibility that closed choice principles make very convenient representatives of Weihrauch degrees (cf. [4, 3, 10, 9, 16]). The case of robust division is no different: For a represented space $\mathbf{X}$ we denote by $\mathrm{AoUC}_\mathbf{X}$ the restriction of $\mathrm{C}_\mathbf{X}$ to $\{A \in \mathcal{A}(\mathbf{X}) \mid |A| = 1\} \cup \{X\}$ following an idea of Brattka. Just by its definition, it is clear that $\mathrm{UC}_\mathbf{X} \leq_W \mathrm{AoUC}_\mathbf{X} \leq_W \mathrm{C}_\mathbf{X}$ holds for any space $\mathbf{X}$. In the following we shall focus on $\mathrm{AoUC}_{[0,1]}$.

▶ **Proposition 8.** [5] $\mathrm{rDiv} \equiv_W AoUC_{[0,1]}$.

Proof. The reduction $\mathrm{rDiv} \leq_\mathrm{W} \mathrm{AoUC}_{[0,1]}$ is straight-forward: On input $(x, y) \in \mathbb{R}^2$ for rDiv, while the search for a $k \in \mathbb{N}$ with $y > 2^{-k}$ continues, the input to $\mathrm{AoUC}_{[0,1]}$ is kept at $[0, 1]$. If such a k is ever found, one can compute $\frac{x}{y}$, and hence also $\{\frac{x}{y}\}$ as $[0, 1]$ is computably Hausdorff and collapse the unit interval to it.

For the other direction, as long as the input to $\mathrm{AoUC}_{[0,1]}$ has not collapsed, one starts to input $(0, 0)$ to rDiv. If the input of $\mathrm{AoUC}_{[0,1]}$ ever collapsed to $\{z\}$, one can compute z. The input to rDiv can still be chosen from some interval $[0, 2^{-k}] \times [0, 2^{-k}]$. In particular, $x = 2^{-k}z$ and $y = 2^{-k}$ works and forces the correct output. ◀

4 Sequential versus concurrent uses of rDiv

If multiple uses of some noncomputable principle are needed to solve a particular task, an important distinction is whether these have to be sequential, or can be applied in a concurrent fashion. In the former case, some instances to the principle may depend on outputs obtained from prior invocations. In the latter, each instantiation is independent of the others. For various principles, however, we find that sequential uses can be reduced to concurrent uses.

▶ **Definition 9.** We call f *finitely concurrent*, iff $f^* \equiv_\mathrm{W} \coprod_{n\in\mathbb{N}} f^{(n)}$.

▶ **Proposition 10** ([16]). *The following are finitely concurrent:*

1. LPO
2. $LLPO \equiv_W C_{\{0,1\}}$
3. $C_{\sharp=2}$
4. $C_{\sharp\leq 2}$

[5] This result was suggested to the author by Brattka, and has been shown in [19].

Whether rDiv is finitely concurrent in this sense was posed as an open question during the Dagstuhl workshop *Measuring the Complexity of Computational Content* in September 2015 [8]. We can now provide a negative answer[6]:

▶ **Theorem 11.** *LLPO* $\star$ *AoUC*$_{[0,1]} \not\leq_W$ *AoUC*$^k_{[0,1]}$ *for all* $k \in \mathbb{N}$.

Proof. We say that a binary tree $T \subseteq \{0,1\}^*$ is an *a.o.u. tree* if for any height $n \in \mathbb{N}$ either $|T \cap \{0,1\}^n| = 2^n$ or $|T \cap \{0,1\}^n| = 1$. Clearly, one can identify $\mathrm{AoUC}_{[0,1]}$ with the partial multi-valued function sending an a.o.u. tree T to all infinite paths through T. We often identify a set $S \subseteq \{0,1\}^*$ with its characteristic function $\chi_S : \{0,1\}^* \to \{0,1\}$. Under this identification, a partial function $t :\subseteq \{0,1\}^* \to \{0,1\}$ is called a *partial tree* if $\mathrm{Tr}(t) := \{\sigma \in \mathrm{dom}(t) : t(\sigma) = 1\}$ forms a subtree of $\{0,1\}^*$. If such t is computable, we call t a *partial computable tree*. Note that a tree is computable if and only if it is of the form $\mathrm{Tr}(t)$ for a partial computable tree t which is *total*, that is, $\mathrm{dom}(t) = \{0,1\}^*$. We say that a partial computable tree t *looks like an a.o.u. tree at* (l,s) if

1. $t(\sigma)[s]$ converges for any binary string σ of length l,
2. for any $n < l$, the cardinality of $\mathrm{Tr}(t) \cap \{0,1\}^n$ is either 2^n or 1,

where $t(\sigma)[s]$ is the result of the computation of $t(\sigma)$ by stage s. Note that given $(l,s) \in \mathbb{N}^2$, we can effectively decide whether t looks like an a.o.u. tree at (l,s) or not. By $\mathrm{aou}(t,s)$ we denote the greatest $l \leq s$ such that t looks like an a.o.u. tree at (l,s).

Consider two partial multi-valued functions $Z_0 = \mathrm{AoUC}_{[0,1]} \times \mathrm{id}_{\mathbf{X}}$ and $Z_1 = (\mathrm{id}_{\{0,1\}^{\mathbb{N}}} \circ \pi_0, \mathrm{C}_{\{0,1\}} \circ \mathrm{eval})$, where $\mathbf{X}$ is the represented space $C(\{0,1\}^{\mathbb{N}}, \mathrm{dom}(\mathrm{C}_{\{0,1\}}))$ of continuous functions from Cantor space $\{0,1\}^{\mathbb{N}}$ into the hyperspace $\mathrm{dom}(\mathrm{C}_{\{0,1\}}) = \mathcal{A}(\{0,1\}) \setminus \{\emptyset\}$ of nonempty closed subsets of $\{0,1\}$. More explicitly, we consider the following two partial multi-valued functions:

$$Z_0 : \mathrm{dom}(\mathrm{AoUC}_{[0,1]}) \times \mathbf{X} \rightrightarrows \{0,1\}^{\mathbb{N}} \times \mathbf{X},$$
$$Z_0(T,S) = \mathrm{AoUC}_{[0,1]}(T) \times \{S\},$$
$$Z_1 : \{0,1\}^{\mathbb{N}} \times \mathbf{X} \rightrightarrows \{0,1\}^{\mathbb{N}} \times 2,$$
$$Z_1(x,S) = \{x\} \times \mathrm{C}_{\{0,1\}}(S(x)).$$

Clearly, $Z_0 \leq_W \mathrm{AoUC}_{[0,1]}$ and $Z_1 \leq_W \mathrm{C}_{\{0,1\}}$. We will show that $Z_1 \circ Z_0 \not\leq_W \mathrm{AoUC}^k_{[0,1]}$. Let $\{(\mathbf{t}^e, \varphi_e, \psi_e)\}_{e \in \mathbb{N}}$ be an effective enumeration of all triples of k-tuples $\mathbf{t}^e = (t^e_i)_{i<k}$ of partial computable trees, partial computable functions $\varphi_e :\subseteq (\{0,1\}^{\mathbb{N}})^k \to \{0,1\}^{\mathbb{N}}$ and $\psi_e :\subseteq (\{0,1\}^{\mathbb{N}})^k \to \{0,1\}$. Intuitively, $(\mathbf{t}^e, \varphi_e, \psi_e)$ is a triple constructed by the opponent Opp, who tries to show $Z_1 \circ Z_0 \leq_W \mathrm{AoUC}^k_{[0,1]}$ for some k. The game proceeds as follows: The proponent Pro of our claim gives an instance (T_r, S_r) of $Z_1 \circ Z_0$, so that $(T_r, S_r) \in \mathrm{dom}(\mathrm{AoUC}_{[0,1]}) \times \mathbf{X}$. In particular, T_r is an a.o.u. tree, and S_r is a continuous function from $[T_r]$ into $\mathcal{A}(\{0,1\})$. Then, Opp reacts with an instance $\mathbf{t}^r$ of $\mathrm{AoUC}^k_{[0,1]}$, that is, a k-tuple $\mathbf{t}^r = (t^r_i)_{i<k}$ of *total* a.o.u. trees. If Opp wins, Opp has to ensure that if $(p_i)_{i<k}$ is a k-tuple of infinite paths through Opp's a.o.u. trees, that is, $p_i \in [\mathrm{Tr}(t^r_i)]$, then $\varphi_r((p_i)_{i<k}) = x$ is a path through Pro's a.o.u. tree T_r and $\psi_r((p_i)_{i<k})$ chooses an element of Pro's set $S_r(x)$, where Opp can use information on (names of) T_r and S_r to construct φ_r and ψ_r. Our purpose is to prevent Opp's strategy.

Given e, we will introduce the e-th strategy, which works as a proponent Pro of our claim. The e-th strategy Pro will construct a computable a.o.u. tree $T_e \subseteq \{0,1\}^*$ and a computable

[6] Which also disproves a claim made in the PhD thesis of the second author [19, Theorem 5.2.1.6].

function $S_e : \{0,1\}^{\mathbb{N}} \to \mathcal{A}(\{0,1\})$ in a computable way uniformly in e. These will prevent Opp's strategy, that is, there is a k-tuple $(p_i)_{i<k}$ of infinite paths through Opp's a.o.u. trees such that if $\varphi_e((p_i)_{i<k}) = x$ chooses a path through Pro's a.o.u. tree T_e then $\psi_e((p_i)_{i<k})$ cannot be an element of Pro's set $S_e(x)$.

We will also define $\mathtt{state}(e,s) \in \{0,\dots,k\} \cup \{\mathtt{end}\}$. The value $\mathtt{state}(e,s) = q$ for $q \neq \mathtt{end}$ indicates that the e-th strategy Pro believes that by stage s, *at least q many trees $(t^e_{u(j)})_{j<q}$ in Opp's k-tuple $(t^e_i)_{i<k}$ have been forced not to have more than one infinite path*, that is, $\mathrm{Tr}(t^e_{u(j)})$ for each $j < q$ has a unique infinite path whenever the opponent Opp has a chance of winning this game with the triple $(\mathbf{t}^e, \varphi_e, \psi_e)$. Under this assumption, if $\mathtt{state}(e,s) = q$, the fact that these q many trees has no more than one infinite path will be witnessed at some stage. The value $\mathtt{state}(e,s) = \mathtt{end}$ indicates that the winning of Pro is already witnessed by Pro's action of shrinking Pro's a.o.u. tree T_e to a tree having a unique path which avoids all φ_e-values made by Opp.

By induction on s, we determine the set $T_e \cap \{0,1\}^s$ of strings in T_e of length s and $\mathtt{state}(e,s)$. In the beginning of our construction, we define $\mathtt{state}(e,0) = 0$. At stage s, we inductively assume that $T_e \cap \{0,1\}^{s-1}$ and $\mathtt{state}(e,s-1)$ have already been defined, say $\mathtt{state}(e,s-1) = q$, and that if $\mathtt{state}(e,s-1) \neq \mathtt{end}$ then $T_e \cap \{0,1\}^{s-1} = \{0,1\}^{s-1}$.

At stage s, the e-th strategy Pro acts as follows:

1. Ask whether there exists $l \leq s$ such that t^e_i for each $i < k$ looks like an a.o.u. tree at (l,s), and at least q-many trees among Opp's k-tuple $(t^e_i)_{i<k}$ have no more than one node above height l. In other words, for $\mathrm{aou}(e,s) := \min_{i<k} \mathrm{aou}(t^e_i, s)$, ask whether there are at least q many $i < k$ such that $|\mathrm{Tr}(t^e_i) \cap \{0,1\}^{\mathrm{aou}(e,s)}| = 1$.
   - **a.** If no, we go to the next stage $s+1$ after setting $\mathtt{state}(e,s) = \mathtt{state}(e,s-1)$ and $T_e \cap \{0,1\}^s = \{0,1\}^s$.
   - **b.** If yes, go to item (2).
2. Ask whether $\varphi_e((p_i)_{i<k})$ already computes a node (of Pro's a.o.u. tree T_e) of length at least $q+1$ for any k-tuple of paths p_i through Opp's a.o.u. trees t^e_i, that is,

$$(\forall i < k)(\forall (\sigma_i)_{i<k})\ [((\forall i < k)\ \sigma_i \in \mathrm{Tr}(t^e_i) \cap \{0,1\}^{\mathrm{aou}(e,s)}) \\ \to\ (\forall m \leq q)\ \varphi_e((\sigma_i)_{i<k})(m)[s] \downarrow].$$

   Here, by effective continuity, the Type-2 computation $\varphi_e : (\{0,1\}^{\mathbb{N}})^k \to \{0,1\}^{\mathbb{N}}$ is approximated by a Type-1 computation $\tilde{\varphi}_e : (\{0,1\}^*)^k \to \{0,1\}^*$ (e.g., consider the Type-2 Turing machine model). We always identify φ_e with $\tilde{\varphi}_e$, and therefore, the notation $\varphi_e((\sigma_i)_{i<k})(m)$ makes sense, that is, by $\varphi_e((\sigma_i)_{i<k})(m)[s] \downarrow$, we mean that the computation of $\tilde{\varphi}_e((\sigma_i)_{i<k})(m)$ halts by stage s.
   - **a.** If no, we go to the next stage $s+1$ after setting $\mathtt{state}(e,s) = \mathtt{state}(e,s-1)$ and $T_e \cap \{0,1\}^s = \{0,1\}^s$.
   - **b.** If yes, go to item (3).
3. Ask whether the image of the product of k many closed sets generated by Opp's k-tuple $\mathbf{t}^e$ under the map φ_e covers the whole space $\{0,1\}^{\mathbb{N}}$. Formally speaking, let us consider the following set:

$$\varphi_e[\mathrm{Tr}(\mathbf{t}^e) \cap \{0,1\}^l] \upharpoonright p := \{\varphi_e((\sigma_i)_{i<k}) \upharpoonright p : (\forall i < k)\ \sigma_i \in \mathrm{Tr}(t^e_i) \cap \{0,1\}^l\},$$

   and then ask whether $\tau \in \varphi_e[\mathrm{Tr}(\mathbf{t}^e) \cap \{0,1\}^{\mathrm{aou}(e,s)}] \upharpoonright q+1$ for all $\tau \in \{0,1\}^{q+1}$.
   - **a.** If no, choose a witness τ, and we finish the construction by setting $\mathtt{state}(e,s) = \mathtt{end}$ after defining T_e as a tree having a unique infinite path $\tau^\frown 0^\infty := \tau^\frown 000\dots$.
   - **b.** If yes, go to item (4).

4. Ask whether $\psi_e((p_i)_{i<k})$ already computes some value $j \in \{0,1\}$ for any k-tuple of paths p_i through Opp's a.o.u. trees t_i^e, that is,

$$(\forall i<k)(\forall(\sigma_i)_{i<k})\ [((\forall i<k)\ \sigma_i \in \mathrm{Tr}(t_i^e)\cap\{0,1\}^{\mathrm{aou}(e,s)}) \ \rightarrow\ \psi_e(\sigma)[s]\downarrow].$$

   a. If no, go to the next stage $s+1$ after setting $\mathtt{state}(e,s) = \mathtt{state}(e,s-1)$ and $T_e \cap \{0,1\}^s = \{0,1\}^s$.
   b. If yes, let $D_{e,s} = \{\sigma \in \mathrm{Tr}(\mathbf{t}^e)\cap\{0,1\}^{\mathrm{aou}(e,s)} : \varphi_e(\sigma) \succeq 0^q1\}$. Note that $D_{e,s} \neq \emptyset$ since we answered yes in item (3); therefore $\varphi_e[\mathrm{Tr}(\mathbf{t}^e)\cap\{0,1\}^{\mathrm{aou}(e,s)}] \upharpoonright q+1 = \{0,1\}^{q+1}$. If $i \notin \psi_e[D_{e,s}]$ for some $i \in \{0,1\}$, then remove $1-i$ from $S_e(0^q1)$ (hence, we have $S_e(0^q1) = \{i\}$). If $\psi_e[D_{e,s}] = \{0,1\}$ then remove 0 from $S_e(0^q1)$. In these cases, set $\mathtt{state}(e,s) = q+1$ and $T_e \cap \{0,1\}^s = \{0,1\}^s$.

Eventually, T_e is constructed as an a.o.u. tree, and $S_e(x) \in \mathrm{dom}(\mathsf{C}_2)$.

▶ **Claim.** *Assume that $(t_i^e)_{i<k}$ determines a k-tuple of a.o.u. trees. Then, there is a realizer G of $\mathrm{AoUC}_{[0,1]}^k$ such that $(\varphi_e \circ G((t_i^e)_{i<k}), \psi_e \circ G((t_i^e)_{i<k}))$ is not a solution to $Z_1 \circ Z_0(T_e, S_e)$, that is, $\varphi_e \circ G((t_i^e)_{i<k}) \notin [T_e]$ or otherwise $\psi_e \circ G((t_i^e)_{i<k}) \notin S_e \circ \varphi_e \circ G((t_i^e)_{i<k})$, where $[T_e]$ denotes the set of all infinite paths through T_e.*

Proof. Assume that $\mathbf{t}^e = (t_i^e)_{i<k}$ determines a k-tuple of a.o.u. trees. In this case, t_i^e is a total tree for each $i<k$. Suppose for the sake of contradiction that the conclusion fails (that is, Opp wins). Then φ_e and ψ_e are defined on all tuples of infinite paths through $\mathrm{Tr}(t_i^e)$, $i<k$. Since $q=0$ at first, the condition in item (1) is automatically fulfilled. Note that since t_i^e is a total a.o.u. tree, the value $\mathrm{aou}(e,s)$ tends to infinity as $s\to\infty$. Therefore, since φ_e is defined on all paths of Opp's trees, by compactness, the condition in item (2) is also satisfied at some stage s. If $\tau \notin \varphi_e[\mathrm{Tr}(\mathbf{t}^e)\cap\{0,1\}^{\mathrm{aou}(e,s)}] \upharpoonright q+1$, then T_e has a unique infinite path $\tau^\frown 0^\infty$; therefore $\varphi_e \circ G(\mathbf{t}^e) \notin [T_e]$ for any realizer G, which contradicts our assumption. Therefore, $\varphi_e[\mathrm{Tr}(\mathbf{t}^e)\cap\{0,1\}^{\mathrm{aou}(e,s)}] \upharpoonright q+1 = \{0,1\}^{q+1}$. By compactness, the condition in item (4) is eventually satisfied. In any cases, for some $\sigma = (\sigma_i)_{i<k} \in D_{e,s}$, $\psi_e(\sigma) \notin S_e(\varphi_e(\sigma))$ by our construction. In order for Opp to win this game, Opp has to declare that σ_i for some $i<k$ is not extendible to an infinite path through t_i^e. Consequently, under our assumption that Opp wins, Pro's strategy forces such t_i^e not to be the full binary tree; therefore t_i^e has only one path since t_i^e is an a.o.u. tree. Then we continue the same argument with $q=1$. We can still satisfy the condition in item (1) at some stage since we know at most one tree t_i^e has only one path. Eventually, this construction forces that any of t_i^e has only one path. Then, however, it is impossible to satisfy $\varphi_e[\mathrm{Tr}(\mathbf{t}^e)\cap\{0,1\}^{\mathrm{aou}(e,s)}] \upharpoonright q+1 = \{0,1\}^{q+1}$. ◀

Suppose for the sake of contradiction that $Z_1 \circ Z_0 \leq_{\mathrm{W}} \mathrm{AoUC}_{[0,1]}^k$ holds via computable H and $K = \langle K_0, K_1\rangle$, i.e., given (T,S), for any k-tuple $\mathbf{p}$ of infinite paths through trees $H(T,S) = \{H_i(T,S)\}_{i<k}$, $(K_0(\mathbf{p},T,S), K_1(\mathbf{p},T,S)) \in Z_1 \circ Z_0(T,S)$, that is, $K_0(\mathbf{p},T,S) \in [T]$ and $K_1(\mathbf{p},T,S) \in S(K_0(\mathbf{p},T,S))$. Choose a computable function f such that $\mathbf{t}^{f(e)} = H(T_e,S_e)$, $\varphi_{f(e)} = \lambda\mathbf{p}.K_0(\mathbf{p},T_e,S_e)$, and $\psi_{f(e)} = \lambda\mathbf{p}.K_1(\mathbf{p},T_e,S_e)$. By Kleene's recursion theorem, there is r such that $(\mathbf{t}^{f(r)}, \varphi_{f(r)}, \psi_{f(r)}) = (\mathbf{t}^r, \varphi_r, \psi_r)$. This triple clearly satisfies the premise of the above claim. The realizer G in the claim witnesses the failure of $Z_1 \circ Z_0 \leq_{\mathrm{W}} \mathrm{AoUC}_{[0,1]}^k$ via H and K. Consequently, $\mathsf{LLPO} \star \mathrm{AoUC}_{[0,1]} \not\leq_{\mathrm{W}} \mathrm{AoUC}_{[0,1]}^k$ for all $k \in \mathbb{N}$. ◀

We point out that the preceding theorem relativizes, i.e. even provides a separation w.r.t. continuous Weihrauch reductions. The same holds for all other separation results in this article.

▶ **Corollary 12.** *$LLPO \star AoUC_{[0,1]} \not\leq_W AoUC_{[0,1]}^*$.*

Proof. Assume to the contrary that $\mathrm{C}_{\{0,1\}} \star \mathrm{AoUC}_{[0,1]} \leq_{\mathrm{W}} \mathrm{AoUC}^*_{[0,1]}$. Consider as input to $\mathrm{LLPO} \star \mathrm{AoUC}_{[0,1]}$ the set $[0,1]$ together with the constant function $h : [0,1] \to \mathcal{A}(\{0,1\})$, $x \mapsto \{0,1\}$. The latter can be represented in such a way that it shares arbitrarily long prefixes with names for any other continuous function of that type. The reduction has to chose some $k \in \mathbb{N}$ eventually that serves as the first component of the derived input to $\mathrm{AoUC}^*_{[0,1]}$. But since the original input can still be altered to any other suitable input, this would imply $\mathrm{LLPO} \star \mathrm{AoUC}_{[0,1]} \leq_{\mathrm{W}} \mathrm{AoUC}^k_{[0,1]}$, thus contradicting Theorem 11. ◀

▶ **Corollary 13.** rDiv *is not finitely concurrent.*

Proof. By Proposition 8, $\mathrm{rDiv} \equiv_{\mathrm{W}} \mathrm{AoUC}_{[0,1]}$ and by Proposition 6 $\mathrm{LLPO} \equiv_{\mathrm{W}} C_{\{0,1\}} \leq_{\mathrm{W}} \mathrm{rDiv}$. Thus, Corollary 12 implies $\mathrm{rDiv} \star \mathrm{rDiv} \nleq_{\mathrm{W}} \mathrm{rDiv}^*$. ◀

We will find next that rDiv only barely fails being finitely concurrent: While some amount of nesting is required to obtain the full power of finitely many uses of rDiv, nesting depths 2 already suffices. This result will be proven via a number of individual technical contributions.

Let $\mathcal{O}(\mathbb{N})$ denote subsets of $\mathbb{N}$ represented via an enumeration of their elements. Call a set $A \subseteq \mathcal{O}(\mathbb{N})$ *nice*, if $\emptyset \in A$ and A contains a computable dense sequence $(a_n)_{n\in\mathbb{N}}$.

▶ **Proposition 14.** *Let $f :\subseteq \mathcal{O}(\mathbb{N}) \rightrightarrows \mathbf{X}$ have a nice domain and a computable closed graph. Then $f \star AoUC^k_{[0,1]} \leq_W C_{\{1,\ldots,2^k\}} \star \left(f^{2^k} \times AoUC^k_{[0,1]}\right)$.*

Proof. For any subset $I \subseteq \{1,\ldots,k\}$ we compute an input to f under the assumption that the components $i \in I$ for $\mathrm{AoUC}^k_{[0,1]}$ are singletons, and the components $i \notin I$ are the whole interval. We start with providing a name for $\emptyset \in \mathrm{dom}(f)$ and wait until all components $i \in I$ have started to collapse. Then we can compute the actual values in those singletons, and can attempt to compute the input to f associated with those values, together with $0 \in [0,1]$ for those components $i \notin I$. Before actually fixing any values, we make sure that there is some element a_n of the dense sequence extending the current finite prefix. If we ever find that some component $i \notin I$ is starting to collapse, we abandon the attempt to find the correct input to f, and just extend the current prefix to some suitable a_n. By the assumption that $\mathrm{dom}(f)$ is nice, this is guaranteed to produce a valid input to f, and if I was indeed the correct choice, will be the correct input.

Now we consider the output of f on each of these values, together with the output$(x_1,\ldots,x_k)$ of $\mathrm{AoUC}^k_{[0,1]}$ on the original input. We replace those x_i with $i \notin I$ with 0, and ask whether this is still a correct output. As the graph of f is a computable closed set, we can ask whether this output matches the input to f obtained from the so modified output of $\mathrm{AoUC}^k_{[0,1]}$. We can compute a truth value $t_I \in \mathbb{S}$ which is false iff both questions answer to true. If I was indeed correct, the corresponding t_I will be false. If t_I is false, then the combined outputs of f and $\mathrm{AoUC}^k_{[0,1]}$ allow us to solve the original question to $f \star \mathrm{AoUC}^k_{[0,1]}$. We can use $\mathrm{C}_{\{1,\ldots,2^k\}}$ to pick some false t_I. ◀

▶ **Corollary 15.** $AoUC^l_{[0,1]} \star AoUC^m_{[0,1]} \leq_W C_{\{1,\ldots,2^m\}} \star AoUC^{l2^m+m}_{[0,1]}$.

Proof. We just need to argue that $\mathrm{AoUC}_{[0,1]}$ is equivalent to some $f :\subseteq \mathcal{O}(\mathbb{N}) \rightrightarrows \mathbf{X}$ satisfying the criteria of Proposition 14. Recall that $A \in \mathcal{A}([0,1]) \supseteq \mathrm{dom}(\mathrm{AoUC}_{[0,1]})$ is represented by enumerating rational open balls exhausting the complement of A. By letting f be equal to $\mathrm{AoUC}_{[0,1]}$, but acting on the enumerations rather than the sets themselves, we have found the required candidate. ◀

▶ **Proposition 16.** *Let $f :\subseteq \mathcal{O}(\mathbb{N}) \rightrightarrows \mathbf{X}$ have a nice domain. Then $f \star C_{\{1,\ldots,n\}} \leq_W f^n \times C_{\{1,\ldots,n\}}$.*

Proof. For each $i \in \{1, \dots, n\}$ we attempt to compute the suitable input to f if i were the output provided by $\mathrm{C}_{\{1,\dots,n\}}$. We only actually write a finite prefix of the output once we have found an element a_n extending it. If we ever learn that i is not a correct output of $\mathrm{C}_{\{1,\dots,n\}}$, we abandon the attempt and simply extend the current input to f to some a_n. The nice domain of f ensures that this procedure results in a valid input for f. If we do this for all choices of i in parallel, and also compute a suitable i, we can then read of a correct output to $f \star \mathrm{C}_{\{1,\dots,n\}}$. ◀

▶ **Corollary 17.** $AoUC^l_{[0,1]} \star C_{\{1,\dots,m\}} \leq_W AoUC^{lm+m-1}_{[0,1]}$.

Proof. To argue that we may use $\mathrm{AoUC}_{[0,1]}$ in place of f in Proposition 16, we argue as we did to obtain Corollary 15 from Proposition 14. Now $\mathrm{C}_{\{1,\dots,m\}} \leq_{\mathrm{W}} C^{m-1}_{\{0,1\}}$ and $C_{\{0,1\}} \leq_{\mathrm{W}} \mathrm{AoUC}_{[0,1]}$ from [17] complete the argument. ◀

So we do find that 3 (or more) consecutive applications of powers of $\mathrm{AoUC}_{[0,1]}$ do reduce to 2:

▶ **Corollary 18.** $AoUC^l_{[0,1]} \star AoUC^m_{[0,1]} \star AoUC^k_{[0,1]} \leq_W AoUC^{(l+1)2^k-1}_{[0,1]} \star AoUC^{m2^k+k}_{[0,1]}$

▶ **Proposition 19.** *Let $f : \mathbf{X} \rightrightarrows \mathbb{N}$ be such that $n \in f(x) \wedge m > n \Rightarrow m \in f(x)$. Then if $f \leq_W C_{\{0,1\}^\mathbb{N}}$, f is already computable.*

▶ **Corollary 20.**

$$\begin{aligned} AoUC^*_{[0,1]} \star AoUC^*_{[0,1]} \quad & \equiv_W \coprod_{n\in\mathbb{N}} (AoUC^n_{[0,1]} \star AoUC^n_{[0,1]}) \\ \equiv_W \coprod_{n\in\mathbb{N}} AoUC^{(n)}_{[0,1]} \equiv_W C^*_{\{0,1\}} \star AoUC^*_{[0,1]} \quad & \equiv_W \left(AoUC^*_{[0,1]}\right)^{(n+1)} \end{aligned}$$

Proof. By Proposition 19 it follows that in e.g. $\mathrm{AoUC}^*_{[0,1]} \star \mathrm{AoUC}^*_{[0,1]}$ the number of oracle calls made in the second round can be bounded in advance. The equivalences now follow from the uniform versions of Corollaries 15, 18. ◀

5 Gaussian Elimination

Most work on algorithms in linear algebra assumes equality to be decidable, and is thus applicable to computability over the rational or algebraic numbers, but not to computability over the real numbers. In the latter setting, computability of some basic questions (rank, eigenvectors,...) was studied in [30], with some additional results in [29, 10]. Here, we shall consider LU-decomposition and Gaussian elimination.

Gaussian elimination is one of the basic algorithms in linear algebra, used in particular to compute the LU-decomposition of matrices. The goal is to transform a given matrix into row echelon form by means of swapping rows (and maybe columns) and adding multiples of one row to another. Sometimes the leading non-zero coefficients in each row are required to be 1, however, as this is easily seen to require equality testing, we shall not include this requirement.

▶ **Definition 21.** $\text{LU-Decomp}_{P,Q}$ takes as input a matrix A, and outputs permutation matrices P, Q, a matrix U in upper echelon form and a matrix L in lower echelon form with all diagonal elements being 1 such that $PAQ = LU$. By LU-Decomp_Q we denote the extension where P is required to be the identity matrix.

▶ **Theorem 22.** *LU-Decomp*$_{P,Q} \equiv_W \mathrm{rDiv}^*$ *and* $\mathrm{rDiv}^* \leq_W$ *LU-Decomp*$_Q \leq_W \mathrm{rDiv}^* \star \mathrm{rDiv}^*$.

The proof of the preceding theorem follows in form of some lemmata. We point out that the upper bounds are proven via variants of Gaussian elimination. In the case of LU-Decomp$_{P,Q}$ and its matching lower bound, this shows that Gaussian elimination exhibits no more incomputability than inherent in the problem it solves. It is consistent with the classifications that the extra freedom in choosing the pivot elements in solving LU-Decomp$_{P,Q}$ compared to solving LU-Decomp$_Q$ makes the problem less incomputable. Resolving the precise degree of LU-Decomp$_Q$ seems to be beyond the reach of our current methods though.

▶ **Lemma 23.** *LU-Decomp*$_{P,Q} \equiv_W$ *LU-Decomp*$^*_{P,Q}$ *and LU-Decomp*$_Q \equiv_W$ *LU-Decomp*$^*_Q$.

Proof. An LU-decomposition of $\left(\begin{array}{cc} A & 0 \\ 0 & B \end{array}\right)$ gives rise to LU-decompositions of both A and B. ◀

▶ **Lemma 24.** *LU-Decomp*$_{P,Q} \leq_W \mathrm{rDiv}^*$.

Proof. Initially, we rearrange all the matrix elements such that at each step the pivot element chosen has the largest absolute value amongst the remaining elements, and obtain their signs. This can be achieved by $\mathrm{C}_{\{0,\ldots,k\}}$ for suitable k. Then we can compute all the relevant divisions simultaneously, using some rDiv^l. Corollary 17 then shows that this reduces to rDiv^*. ◀

▶ **Lemma 25.** *LU-Decomp*$_Q \leq_W \mathrm{rDiv}^* \star \mathrm{rDiv}^*$.

Proof. Given some real matrix $(a_{ij})_{i\leq n, j\leq m}$ we can use $\mathrm{C}_{\{0,\ldots,n-1\}}$ to pick some i_0 such that $|a_{i_0,1}| = \max_{i\leq n} |a_{i,1}|$, and permute the rows to move the i_0-th row to the top. We can use $C^n_{\{0,1\}}$ to figure out for each i whether $|a_{i,1}|$ is non-negative or non-positive. For each $i \neq i_0$ we compute $\mathrm{rDiv}(|a_{i,1}|, |a_{i_0,1}|)$, pick the sign depending on the putative signs on $a_{i,1}$ and $a_{i_0,1}$ and then subtract the corresponding multiple of the i_0-th row from the i-th row. By choice of i_0, either all $a_{i,1}$ are 0 anyway, or $a_{i_0,1} \neq 0$ – in both cases, this ensures that in all rows but the i_0-th the first entry is zero after the operation.

The procedure so far made use of $\mathrm{rDiv}^{n-1} \star \left(\mathrm{C}_{\{0,\ldots,n-1\}} \times C^n_{\{0,1\}}\right)$.

After the first round, the now first row is fixed. Amongst the remaining ones, we pick one with the largest absolute value in the second column (using $\mathrm{C}_{\{0,\ldots,n-2\}}$), determine the signs of entries in the second column (using $C^{n-2}_{\{0,1\}}$) and again use rDiv to compute the coefficients for subtracting the second row from the lower ones.

This is repeated until each row has been dealt with. Overall, we use $n-1$ rounds, so the procedure is reducible to $\left[\mathrm{rDiv}^{n-1} \star \left(\mathrm{C}_{\{0,\ldots,n-1\}} \times C^n_{\{0,1\}}\right)\right]^{(n)}$. By repeated application of Corollaries 17,18 this reduces to $\mathrm{AoUC}^k_{[0,1]} \star \mathrm{AoUC}^k_{[0,1]}$ for sufficiently big k (depending effectively on n). ◀

▶ **Lemma 26.** $\mathrm{rDiv} \leq_W$ *LU-Decomp*$_{P,Q}$.

Proof. We consider the computable function $B : [0,1] \to \mathbb{R}^{2\times 2}$ of matrices defined via:

$$B(\varepsilon) = \exp(-\varepsilon^{-2}) \left(\begin{array}{cc} \cos(\varepsilon^{-1}) & \sin(\varepsilon^{-1}) \\ -\sin(\varepsilon^{-1}) & \cos(\varepsilon^{-1}) \end{array}\right) \text{ for } \varepsilon > 0 \quad B(0) = \left(\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}\right)$$

This is based on a counterexample due to Rellich (cmp. [14, II.5.3], [30, Example 18]). If $\varepsilon \neq 0$, then the lower-left corner of L in an LU-decomposition of $B(\varepsilon)$ will be one of $\tan \varepsilon^{-1}$,

$\cot \varepsilon^{-1}$ or $-\cot \varepsilon^{-1}$. The relevant case can be obtained from P and Q. As arctan and arccot are total, we can apply the relevant inverse even if $\varepsilon = 0$, and thus the lower-left corner of L is an arbitrary real number. Let x'_ε be the result, and $x_\varepsilon = \max\{0, \min\{1, x'_\varepsilon\}\}$.

We want to show that $\mathrm{AoUC}_{[0,1]} \leq_{\mathrm{W}} \text{LU-Decomp}_{P,Q}$ (which is equivalent to the claim by Proposition 8). Given $A \in \mathrm{dom}(\mathrm{AoUC}_{[0,1]})$, we show how to compute some $\varepsilon \in [0,1]$ such that $x_\varepsilon \in A$. As long as $A = [0,1]$ is consistent with our knowledge of the input, we specify that $\varepsilon \in [0, 2^{-t}]$ for smaller and smaller $t \in \mathbb{N}$. If we learn at time t that $A \neq [0,1]$, we compute y such that $A = \{y\}$ and choose $k \in \mathbb{N}$ such that $(2k\pi)^{-k} \leq 2^{-t}$. We can then specify $\varepsilon = (2k\pi + y)^{-1}$. But now $x_\varepsilon = y$. If $A = [0,1]$, then $x_\varepsilon \in A$ anyway by definition. ◀

Based on the preceding lemma and [30, Example 18], we can also see that rDiv is reducible to finding eigenvectors of a matrix.

References

1 Jeremy Avigad and Vasco Brattka. Computability and analysis: the legacy of Alan Turing. In Rod Downey, editor, *Turing's Legacy*, volume 42 of *Lecture Notes in Logic*, pages 1–47. Cambridge University Press, 2014. available at http://arxiv.org/abs/1206.3431.

2 Émile Borel. Le calcul des intégrales définies. *Journal de Mathématiques pures et appliquées*, 6(8):159–210, 1912.

3 Vasco Brattka, Matthew de Brecht, and Arno Pauly. Closed choice and a uniform low basis theorem. *Annals of Pure and Applied Logic*, 163(8):968–1008, 2012. `doi:10.1016/j.apal.2011.12.020`.

4 Vasco Brattka and Guido Gherardi. Effective choice and boundedness principles in computable analysis. *Bulletin of Symbolic Logic*, 1:73–117, 2011. arXiv:0905.4685. `doi:10.2178/bsl/1294186663`.

5 Vasco Brattka and Guido Gherardi. Weihrauch degrees, omniscience principles and weak computability. *Journal of Symbolic Logic*, 76:143–176, 2011. arXiv:0905.4679.

6 Vasco Brattka, Guido Gherardi, and Rupert Hölzl. Probabilistic computability and choice. *Information and Computation*, 242:249–286, 2015. arXiv 1312.7305. `doi:10.1016/j.ic.2015.03.005`.

7 Vasco Brattka, Guido Gherardi, and Alberto Marcone. The Bolzano-Weierstrass Theorem is the jump of Weak König's Lemma. *Annals of Pure and Applied Logic*, 163(6):623–625, 2012. also arXiv:1101.0792. `doi:10.1016/j.apal.2011.10.006`.

8 Vasco Brattka, Akitoshi Kawamura, Alberto Marcone, and Arno Pauly. Measuring the Complexity of Computational Content (Dagstuhl Seminar 15392). *Dagstuhl Reports*, 5(9):77–104, 2016. `doi:10.4230/DagRep.5.9.77`.

9 Vasco Brattka, Stéphane Le Roux, and Arno Pauly. On the computational content of the Brouwer fixed point theorem. In S.Barry Cooper, Anuj Dawar, and Benedikt Löwe, editors, *How the World Computes*, volume 7318 of *Lecture Notes in Computer Science*, pages 56–67. Springer Berlin Heidelberg, 2012. `doi:10.1007/978-3-642-30870-3_7`.

10 Vasco Brattka and Arno Pauly. Computation with advice. *Electronic Proceedings in Theoretical Computer Science*, 24, 2010. CCA 2010. URL: `http://arxiv.org/html/1006.0551`.

11 Vasco Brattka and Arno Pauly. On the algebraic structure of Weihrauch degrees. arXiv 1604.08348, 2016. URL: `http://arxiv.org/abs/1604.08348`.

12 Guido Gherardi and Alberto Marcone. How incomputable is the separable Hahn-Banach theorem? *Notre Dame Journal of Formal Logic*, 50(4):393–425, 2009. `doi:10.1215/00294527-2009-018`.

13 Kojiro Higuchi and Arno Pauly. The degree-structure of Weihrauch-reducibility. *Logical Methods in Computer Science*, 9(2), 2013. `doi:10.2168/LMCS-9(2:2)2013`.

14 Tosio Kato. *Pertubation Theory for Linear Operators*. Springer, 1976.

15 Takayuki Kihara and Arno Pauly. Dividing by zero – how bad is it, really? arXiv:1606.04126, 2016.

16 Stéphane Le Roux and Arno Pauly. Finite choice, convex choice and finding roots. *Logical Methods in Computer Science*, 2015. URL: `http://arxiv.org/abs/1302.0380`, `doi:10.2168/LMCS-11(4:6)2015`.

17 Arno Pauly. How incomputable is finding Nash equilibria? *Journal of Universal Computer Science*, 16(18):2686–2710, 2010. `doi:10.3217/jucs-016-18-2686`.

18 Arno Pauly. On the (semi)lattices induced by continuous reducibilities. *Mathematical Logic Quarterly*, 56(5):488–502, 2010. `doi:10.1002/malq.200910104`.

19 Arno Pauly. *Computable Metamathematics and its Application to Game Theory*. PhD thesis, University of Cambridge, 2012.

20 Arno Pauly. Many-one reductions and the category of multivalued functions. *Mathematical Structures in Computer Science*, 2015. available at: arXiv 1102.3151. `doi:10.1017/S0960129515000262`.

21 Arno Pauly. On the topological aspects of the theory of represented spaces. *Computability*, 2016. accepted for publication, available at http://arxiv.org/abs/1204.3763. `doi:10.3233/COM-150049`.

22 Arno Pauly and Florian Steinberg. Representations of analytic functions and weihrauch degrees. In *Proceedings of* Computer Science Russia *(CSR)*, volume 9691 of *LNCS*, pages 367–381, 2016. `doi:10.1007/978-3-319-34171-2_26`.

23 Arno Pauly and Martin Ziegler. Relative computability and uniform continuity of relations. *Journal of Logic and Analysis*, 5, 2013.

24 Matthias Schröder. Extended admissibility. *Theoretical Computer Science*, 284(2):519–538, 2002. `doi:10.1016/S0304-3975(01)00109-8`.

25 Alan Turing. On computable numbers, with an application to the Entscheidungsproblem: Corrections. *Proceedings of the LMS*, 2(43):544–546, 1937.

26 Klaus Weihrauch. The degrees of discontinuity of some translators between representations of the real numbers. Informatik Berichte 129, FernUniversität Hagen, Hagen, 1992.

27 Klaus Weihrauch. The TTE-interpretation of three hierarchies of omniscience principles. Informatik Berichte 130, FernUniversität Hagen, Hagen, 1992.

28 Klaus Weihrauch. *Computable Analysis*. Springer-Verlag, 2000.

29 Martin Ziegler. Real computation with least discrete advice: A complexity theory of nonuniform computability with applications to effective linear algebra. *Annals of Pure and Applied Logic*, 163(8):1108–1139, 2012. `doi:10.1016/j.apal.2011.12.030`.

30 Martin Ziegler and Vasco Brattka. Computability in linear algebra. *Theoretical Computer Science*, 326:187–211, 2004.

Advice Complexity of the Online Induced Subgraph Problem*

Dennis Komm[1], Rastislav Královič[2], Richard Královič[3], and Christian Kudahl[4]

1 **Dept. of Computer Science, ETH Zurich, Zurich, Switzerland**
`dennis.komm@inf.ethz.ch`
2 **Dept. of Computer Science, Comenius University, Bratislava, Slovakia**
`kralovic@dcs.fmph.uniba.sk`
3 **Google Inc., Switzerland**
`richard.kralovic@dcs.fmph.uniba.sk`
4 **Dept. of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark**
`kudahl@imada.sdu.dk`

Abstract

Several well-studied graph problems aim to select a largest (or smallest) induced subgraph with a given property of the input graph. Examples include maximum independent set, maximum planar graph, maximum clique, minimum feedback vertex set, and many others. In online versions of these problems, the vertices of the graph are presented in an adversarial order, and with each vertex, the online algorithm must irreversibly decide whether to include it into the constructed subgraph, based only on the subgraph induced by the vertices presented so far. We study the properties that are common to all these problems by investigating a generalized problem: for an arbitrary but fixed hereditary property π, find some maximal induced subgraph having π. We investigate this problem from the point of view of advice complexity, i. e., we ask how some additional information about the yet unrevealed parts of the input can influence the solution quality. We evaluate the information in a quantitative way by considering the best possible advice of given size that describes the unknown input. Using a result from Boyar et al. [STACS 2015, LIPIcs 30], we give a tight trade-off relationship stating that, for inputs of length n, roughly n/c bits of advice are both needed and sufficient to obtain a solution with competitive ratio c, regardless of the choice of π, for any c (possibly a function of n). This complements the results from Bartal et al. [SIAM Journal on Computing 36(2), 2006] stating that, without any advice, even a randomized algorithm cannot achieve a competitive ratio better than $\Omega(n^{1-\log_4 3-o(1)})$. Surprisingly, for a given cohereditary property π and the objective to find a minimum subgraph having π, the advice complexity varies significantly with the choice of π. We also consider a preemptive online model, inspired by some applications mainly in networking and scheduling, where the decision of the algorithm is not completely irreversible. In particular, the algorithm may discard some vertices previously assigned to the constructed set, but discarded vertices cannot be reinserted into the set. We show that, for the maximum induced subgraph problem, preemption does not significantly help by giving a lower bound of $\Omega(n/(c^2 \log c))$ on the bits of advice that are needed to obtain competitive ratio c, where c is any increasing function bounded from above by $\sqrt{n/\log n}$. We also give a linear lower bound for c close to 1.

1998 ACM Subject Classification F.1.2 Modes of Computation

Keywords and phrases Online algorithms, advice complexity, induced subgraph problem

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.59

* Supported in part by the Villum Foundation and the Stibo-Foundation and SNF grant 200021-146372.

1 Introduction

Online algorithms get their input gradually, and this way have to produce parts of the output without full knowledge of the instance at hand, which is a large disadvantage compared to classical *offline computation*, yet a realistic model of many real-world scenarios [5]. Most of the offline problems have their online counterpart. Instead of asking about the time and space complexity of algorithms to solve a computational problem, *competitive analysis* is commonly used as a tool to study how well online algorithms perform [5, 18] without any time or space restrictions; the analogous offline measurement is the analysis of the *approximation ratio.* A large class of computational problems for both online and offline computation are formulated on graphs; we call such problems (online) *graph problems.*

In this paper, we deal with problems on unweighted undirected graphs that are given to an online algorithm vertex by vertex in consecutive discrete time steps. Formally, we are given a graph $G = (V, E)$, where $|V| = n$, with an ordering $\prec$ on V. Without loss of generality, assume $V = \{v_1, \ldots, v_n\}$, and $v_1 \prec \ldots \prec v_n$ specifies the order in which the vertices of G are presented to an online algorithm; this way, the vertex v_i is given in the ith time step. Together with v_i, all edges $\{v_j, v_i\} \in E$ are revealed for all $v_j \prec v_i$. If v_i is revealed, an online algorithm must decide whether to accept v_i or discard it. Neither G nor n are known to the online algorithm. We study two versions of online problems; with and without preemption. In the former case, the decision whether v_i is accepted or not is definite. In the latter case, in every time step, the online algorithm may preempt (discard) some of the vertices it previously accepted; however, a vertex that was once discarded cannot be part of the solution anymore.

For an instance $I = (v_1, \ldots, v_n)$ of some graph problem, we denote by $\textsc{Alg}(I)$ the solution computed by some online algorithm $\textsc{Alg}$; $\textsc{Opt}(I)$ denotes an optimal solution for I, which can generally only be computed with the full knowledge of I. We assume that I is constructed in an *adversarial manner* to give worst-case bounds on the solution quality of any online algorithm. This means that we explicitly think of I as being given by an adversary that knows $\textsc{Alg}$ and wants to make it perform as poorly as possible; for more details, we refer to the standard literature [5].

For maximization problems with an associated *profit function* called profit, an online algorithm $\textsc{Alg}$ is called *c-competitive* if, for every instance I of the given problem, it holds that

$$\text{profit}(\textsc{Alg}(I)) \geq 1/c \cdot \text{profit}(\textsc{Opt}(I)) \; ; \tag{1}$$

likewise, for minimization problems with a *cost function* called cost, we require

$$\text{cost}(\textsc{Alg}(I)) \leq c \cdot \text{cost}(\textsc{Opt}(I)) \tag{2}$$

for every instance I. In this context, $c > 1$ may be a constant or a function that increases with the input length n. We will use c and $c(n)$ interchangeably to refer to the competitive ratio; the latter is simply used to emphasize that c may depend on n.

Throughout this paper, log denotes the binary logarithm $\log_2$.

Instead of studying specific graph problems, in this paper, we investigate a large class of such problems, which are defined by *hereditary properties.* This class includes many well-known problems such as *maximum independent set*, *maximum planar graph*, *maximum induced clique*, and *maximum acyclic subgraph.* The *cohereditary problems* we consider are online versions of the offline problem of searching for a specific structure within a graph. An example is to find the shortest cycle; this defines the girth of the graph. *Online cycle finding* was considered by Boyar et al. [7].

We call any collection of graphs a *graph property* π. A graph has (or *satisfies*) property π if it is in the collection. Examples include the property of being planar (the collection contains all planar graphs), or being an independent set (the collection contains all graphs with no edges). We only consider properties that are *non-trivial*, i. e., they are both true for infinitely many graphs and false for infinitely many graphs. A property is called *hereditary* if it holds that, if a graph G satisfies π, then also any induced subgraph G' of G satisfies π; conversely, it is called *cohereditary* if it holds that, if a graph G satisfies π, and G is an induced subgraph of G', then also G' satisfies π. For a graph $G = (V, E)$ and a subset of vertices $S = \{v_1, \ldots, v_i\} \subseteq V$, let $G[S]$ (or $G[v_1, \ldots, v_i]$) denote the subgraph of G induced by the vertices from S. For a graph $G = (V, E)$, let $\overline{G} = (V, \overline{E})$ be the complement of G, i. e., $\{u, v\} \in \overline{E}$ if and only if $\{u, v\} \notin E$. Let K_n denote the complete graph on n vertices, and let $\overline{K}_n$ denote the independent set on n vertices. We consider the online version of the problem of finding maximal (minimal, respectively) induced subgraphs satisfying a hereditary (cohereditary, respectively) property π, denoted by $\textsc{Max}$-π ($\textsc{Min}$-π, respectively). For the ease of presentation, we will call such problems *hereditary (cohereditary, respectively) problems*. Let $S_{\textsc{Alg}} := \textsc{Alg}(I)$ denote the set of vertices accepted by some online algorithm $\textsc{Alg}$ for some instance I of a hereditary problem. Then, for $\textsc{Max}$-π, the profit of $\textsc{Alg}$ is $|S_{\textsc{Alg}}| := \text{profit}(\textsc{Alg}(I))$ if $G[S_{\textsc{Alg}}]$ has the property π and $-\infty$ otherwise; the goal is to maximize the profit. Conversely, for $\textsc{Min}$-π, the cost of $\textsc{Alg}$ is $|S_{\textsc{Alg}}| := \text{cost}(\textsc{Alg}(I))$ if $G[S_{\textsc{Alg}}]$ has the property π and ∞ otherwise; the goal is to minimize the cost. As an example, consider the *online maximum independent set* problem; the set of all independent sets is clearly a hereditary property (every independent set is a feasible solution, and every induced subset of an independent set is again an independent set). When a vertex is revealed, an online algorithm needs to decide whether it becomes part of the solution or not. The goal is to compute an independent set that is as large as possible; the profit of the solution is thus equal to $|S_{\textsc{Alg}}|$. It is straightforward to define the problem without or with preemption.

In this paper, we study *online algorithms with advice* for hereditary and cohereditary problems. In this setup, an online algorithm is equipped with an additional resource that contains information about the instance it is dealing with. A related model was originally introduced by Dobrev et al. [9]. Revised versions were defined by Emek et al. [11], Böckenhauer et al. [4], and Hromkovič et al. [12]. Here, we use the model of the latter two papers. Consider an input $I = (v_1, \ldots, v_n)$ of a hereditary problem. An *online algorithm* $\textsc{Alg}$ *with advice* computes the output sequence $\textsc{Alg}^{\phi}(I) = (y_1, \ldots, y_n)$ such that y_i is computed from $\phi, v_1, \ldots, v_i$, where ϕ is the content of the advice tape, i. e., an infinite binary sequence. We denote the cost (profit, respectively) of the computed output by $\text{cost}(\textsc{Alg}^{\phi}(I))$ ($\text{profit}(\textsc{Alg}^{\phi}(I))$, respectively). The algorithm $\textsc{Alg}$ is *c-competitive with advice complexity* $b(n)$ if, for every n and for each I of length at most n, there exists some ϕ such that $\text{cost}(\textsc{Alg}^{\phi}(I)) \leq c \cdot \text{cost}(\textsc{Opt}(I))$ ($\text{profit}(\textsc{Alg}^{\phi}(I)) \geq 1/c \cdot \text{profit}(\textsc{Opt}(I))$, respectively) and at most the first $b(n)$ bits of ϕ have been accessed by $\textsc{Alg}$.[1] We sometimes simply write b instead of $b(n)$ to increase readability.

The motivation for online algorithms with advice is mostly of a theoretical nature, as we may think of the information necessary and sufficient to compute an optimal solution as the *information content* of the given problem [12]. Moreover, there is a non-trivial connection to randomized online algorithms [3, 13]. Lower bounds on the advice complexity often translate

[1] Note that usually an additive constant is included in the definition of c-competitiveness, i. e., in (1) and (2). However, for the problems we consider, this changes the advice complexity by at most $O(\log n)$; see Remark 9 in Boyar et al. [7].

to lower bounds for semi-online algorithms. Essentially, here one studies whether knowing some small parameter of an online problem (such as the length of the input or the number of requests of a certain type) results in a much better competitive ratio. Lower bound results using advice can often help to answer this question. Similarly, lookahead can be seen as a special kind of advice that is supplied to an algorithm. This way, online algorithms with advice generalize a number of concepts introduced to give online algorithms more power. However, the main question posed is how much any kind of (computable) information could help; and maybe even more importantly, which amount of information will never help to overcome some certain threshold, no matter what this information actually is.

Organization, Related Work, and Results

We are mainly concerned with proving lower bounds of the form that a particular number of advice bits is necessary in order to obtain some certain output quality for a given hereditary property. We make heavy use of online reductions between generic problems and the studied ones that allow us to bound the number of advice bits necessary from below. Emek et al. [11] used this technique in order to prove lower bounds for metrical task systems. The foundations of the reductions as we perform them here are due to Böckenhauer et al. [2], who introduced the *string guessing problem*, and Boyar et al. [7], who studied a problem called *asymmetric string guessing*. Mikkelsen [20] introduced a problem, which we call the *anti-string guessing problem*, and which is a variant of string guessing with a more "friendly" cost function. Our reductions rely on some results from Bartal et al. [1] that characterize hereditary properties by forbidden subgraphs together with some insights from Ramsey theory (see, e. g., Diestel [8]).

In Section 2, we recall some basic results from Ramsey theory and define the generic online problems that we use as a basis of our reductions. In Section 3, we study both Max-π and Min-π in the case that no preemption is allowed; using a reduction from the asymmetric string guessing problem, we show that any c-competitive online algorithm for Max-π needs roughly n/c advice bits, and this is essentially tight. This complements results from Bartal et al. [1], which state that, without any advice, even a randomized algorithm cannot achieve a competitive ratio better than $\Omega(n^{1-\log_4 3-o(1)})$. The advice complexity of the maximum independent set problem on bipartite and sparse graphs was studied by Dobrev et al. [10]. In the subsequent sections, we allow the online algorithm to use preemption. In Section 4, we use a reduction from the string guessing problem to show a lower bound of $\Omega(n/(c^2 \log c))$ on the number of advice bits that are needed to obtain competitive ratio c, where c is any increasing function bounded from above by $\sqrt{n/\log n}$. In Section 5, using a reduction from the anti-string guessing problem, we also give a linear lower bound for c being close to 1.

Due to space constraints, some of the proofs are omitted.

2 Preliminaries

Hereditary properties can be characterized by forbidden induced subgraphs as follows: if a graph G does not satisfy a hereditary property π, then any graph H such that G is an induced subgraph of H does not satisfy π neither. Hence, there is a (potentially infinite) set of minimal forbidden graphs (w.r.t. being induced subgraph) S_π such that G satisfies π if and only if no graphs from S_π are induced subgraphs of G. Conversely, any set of graphs S defines a hereditary property π_S of not having a graph from S as induced subgraph.

Furthermore, there is the following bijection between hereditary and cohereditary properties: for a hereditary property π we can define a property $\overline{\pi}$ such that a graph G satisfies $\overline{\pi}$

if and only if it does not satisfy π (it is easy to see that $\overline{\pi}$ is cohereditary), and vice versa. Hence, a cohereditary property $\overline{\pi}$ can be characterized by a set of minimal (w.r.t. being induced subgraph) *obligatory* subgraphs $S_{\overline{\pi}}$ such that a graph G has the property $\overline{\pi}$ if and only if at least one graph from $S_{\overline{\pi}}$ is an induced subgraph of G.

To each property π we can define the complementary property π^c such that a graph G satisfies π^c if and only if the complement of G satisfies π. Clearly, if π is (co)hereditary, so is π^c. Moreover, if H is forbidden (obligatory, respectively) for π, $\overline{H}$ is forbidden (obligatory, respectively) for π^c. The following statement is due to Lewis and Yannakakis.

▶ **Lemma 1** (Lewis and Yannakakis [15], proof of Theorem 4)**.** *Every non-trivial hereditary property π is satisfied either by all cliques or by all independent sets.*

Proof. Assume, for the sake of contradiction, that there is a hereditary property π, and two numbers m, n, such that K_m and $\overline{K}_n$ do not satisfy π. Let $r(m,n)$ be the Ramsey number[17], such that every graph with at least $r(m,n)$ vertices contains K_m or $\overline{K}_n$ as induced subgraph. Since π is non-trivial, there is a graph G with more than $r(m,n)$ vertices that satisfies π. G contains either K_m or $\overline{K}_n$ as induced subgraph, and since π is hereditary, either K_m or $\overline{K}_n$ satisfies π. ◀

Bartal et al. proved the following theorem. It is formulated in the known supergraph model, where a graph $G=(V,E)$ with n vertices is a-priori known to the algorithm, and the input is a sequence of vertices $v_1,\ldots,v_k$. The task is to select in an online manner the subgraph of the induced graph $G[v_1,\ldots,v_k]$ having property π.

▶ **Theorem 2** (Bartal et al. [1] and references therein)**.** *In the known supergraph model, any randomized algorithm for the* Max-π *problem has competitive ratio*

$$\Omega\left(n^{1-\log_4 3-o(1)}\right),$$

even if preemption is allowed.

Note that n in the previous theorem thus refers to the size of the known supergraph, and not to the length of the input sequence. However, in the proof a graph with $n=4^i$ vertices is considered, from which subgraphs of size 3^i are presented. Each of these instances has an optimal solution of size at least 2^i, and it is shown that any deterministic algorithm can have a profit of at most $\alpha(3/2)^i\log n$ on average, for some constant α. From that, using Yao's principle [19] as stated in [6], the result follows. The same set of instances thus yields the following result.

▶ **Theorem 3** (Bartal et al. [1])**.** *Any randomized algorithm for the* Max-π *problem has competitive ratio*

$$\Omega\left(n^{2/\log 3-1-o(1)}\right),$$

even if preemption is allowed.

Next, we describe some specific online problems that allow us to give lower bounds on the advice complexity using a special kind of reduction. Böckenhauer et al. [2] introduced a very generic online problem called *string guessing with known history over alphabets of size σ* (σ-SGKH). The input is a sequence of requests $(x_0,\ldots,x_n)$ where $x_0=n$ and for $i\geq 1$, $x_i\in\{1,\ldots,\sigma\}$. The algorithm has to produce a sequence of answers $(y_1,\ldots,y_n,y_{n+1})$, where $y_i\in\{1,\ldots,\sigma\}$ and $y_{n+1}=\perp$ and where y_i is allowed to depend on $x_0,\ldots,x_{i-1}$ (and of course any advice bits the algorithm reads). The cost is the number of positions i for which $y_i\neq x_i$.

▶ **Theorem 4** (Böckenhauer et al. [2]). *Let $\sigma \geq 2$. Any online algorithm with advice for σ-SGKH that guesses γn bits of the input correctly must read at least*

$$\left(1 + (1-\gamma)\log_\sigma\left(\frac{1-\gamma}{\sigma-1}\right) + \gamma \log_\sigma \gamma\right) n \log \sigma$$

bits of advice.

Mikkelsen [20] introduced the problem *anti-string guessing with known history over alphabets of size σ* (Anti-σ-SGKH). It is defined exactly as σ-SGKH except that the cost is the number of positions i for which $y_i = x_i$.

▶ **Theorem 5** (Mikkelsen [20, Theorem 11]). *Let $\sigma \geq 2$ and let $1 \leq c < \sigma/(\sigma-1)$. Any c-competitive Anti-σ-SGKH algorithm must read at least*

$$\left(1 - h_\sigma\left(\frac{1}{c}\right)\right) n \log \sigma$$

bits of advice, where n is the input length. This holds even if n is known in advance. Here, h_σ is the σ-ary entropy function given by $h_\sigma(x) = x\log_\sigma(\sigma-1) - x\log_\sigma x - (1-x)\log_\sigma(1-x)$.

Boyar et al. [7] investigated a problem called *maximum asymmetric string guessing* (maxASGk). The input is a sequence of requests $(x_0, \ldots, x_n)$ where $x_0 = \perp$ and for $i \geq 1$, $x_i \in \{0,1\}$. The algorithm has to produce a sequence of answers $(y_1, \ldots, y_n, y_{n+1})$. The output is feasible if $x_i \leq y_i$ for all $1 \leq i \leq n$. The profit of the algorithm is the number of zeros in $y_1, \ldots, y_n$ for feasible outputs, and $-\infty$ otherwise. The "blind" version of the problem, where the algorithm has to produce the output without actually seeing the requests (i. e., in each step, the algorithm receives some dummy request $\perp$), is denoted maxASGu. In what follows, let

$$B_c := \log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right) \approx \frac{1}{c} \cdot \frac{1}{\mathrm{e}\ln 2}\ .$$

▶ **Theorem 6** (Boyar et al. [7]). *For any function $c(n)$ such that $1 \leq c(n) \leq n$, there is a c-competitive algorithm for maxASGk (maxASGu, respectively) with advice of size $B_c \cdot n + O(\log n)$. Moreover, any c-competitive algorithm for maxASGk (maxASGu, respectively) must read at least*

$$B_c \cdot n - O(\log n)$$

bits of advice.

Note that, in general, it does not make much difference if the length of the input is initially known to the algorithm or not. More specifically, it changes the advice complexity by at most $O(\log n)$.

3 Max-π and Min-π without Preemption

First, we show that for any non-trivial hereditary property π, the Max-π problem is equivalent to asymmetric string guessing in the following sense.

▶ **Theorem 7.** *If there is a c-competitive algorithm for maxASGu, then there is a c-competitive algorithm for Max-π using the same advice.*

▶ **Theorem 8.** *If there is a c-competitive algorithm for* Max-Π *that reads $b(n)$ bits of advice, then there is a c-competitive algorithm for* maxASGk *using*

$$b(n) + O(\log^2 n)$$

bits of advice.

The proof of Theorem 7 is omitted due to space constraints. Before proving Theorem 8, let us recall Lemma 3 from Bartal et al. [1].

▶ **Lemma 9** (Bartal et al. [1]). *Given any graph H, there exist constants n_0 and α such that for all $n > n_0$ there exists a graph G on n vertices such that any induced subgraph of G on at least $\alpha \log n$ vertices contains H as an induced subgraph.*

This is a variant of Lemma 9 from Lund and Yannakakis[2] [16].

▶ **Lemma 10** (Lund and Yannakakis [16]). *Let H be a graph on k vertices. For sufficiently large N, for any graph G on N vertices and for all $\ell = \Omega(\log N)$, a random subgraph G' of G does not, with probability $1/2$, contain a subset S of ℓ vertices that is a clique in G but H is not an induced subgraph of $G'[S]$.*

Proof of Theorem 8. According to Lemma 1, π is satisfied either by all cliques or by all independent sets. Without loss of generality, suppose the latter (otherwise, swap the edges and non-edges in the following arguments).

Consider a binary string $\nu = x_1, \ldots, x_n$ (for large enough n). Let us consider the graph $G_\nu = (V, E)$ defined as follows. Let H be an arbitrary but fixed forbidden subgraph of π. Let G' be the n-vertex graph from Lemma 9 with vertices $V = \{v_1, \ldots, v_n\}$. If $x_i = 0$ for some i, delete from G' all edges $\{v_i, v_j\}$ for $j > i$. In the graph G_ν defined this way, the vertices v_i for which the corresponding x_i satisfies $x_i = 0$ (denoted by $I_\nu \subseteq V$ in the sequel) form an independent set, and hence $G_\nu[I_\nu]$ has property π. On the other hand, any induced subgraph $G_\nu[S]$ with property π can contain at most $\alpha \log n$ vertices from $V \setminus I_\nu$ (otherwise it would contain the forbidden graph H as induced subgraph). Note that, with $O(\log n)$ bits of advice to encode n, the graph G_ν can be constructed from the string ν in an online manner: the base graph G' is fixed for a fixed n, and the subgraph $G_\nu[v_1, \ldots, v_i]$ depends only on the values of $x_1, \ldots, x_{i-1}$.

Now consider a c-competitive algorithm Alg$_\pi$ for Max-Π that uses b bits of advice. Let us describe how to derive an algorithm Alg for maxASGk from Alg$_\pi$. For a given string $\nu = x_1, \ldots, x_n$, where $\perp, x_1, \ldots, x_n$ is the input for maxASGk, the advice for Alg consists of three parts: first, there is a self-delimiting encoding of n using $O(\log n)$ bits, followed by a (self-delimiting) correction string e_ν of length $O(\log^2 n)$ bits described later, and the rest is the advice for Alg$_\pi$ on the input G_ν. Let S be the solution (set of vertices) returned by Alg$_\pi$ on G_ν (with the proper advice). As argued before, S can contain at most $\alpha \log n$ vertices from $V \setminus I_\nu$. The indices of these vertices from $S_{\text{out}} := S \cap (V \setminus I_\nu)$ are part of the string e_ν. Apart from that, e_ν contains the indices of at most $\alpha \log n$ vertices $S_{\text{in}} \subseteq I_\nu$ such that $|(S \setminus S_{\text{out}}) \cup S_{\text{in}}| = \min\{|S|, |I_\nu|\}$.

The algorithm Alg works as follows: at the beginning, it constructs the graph G'. When a request x_i arrives, Alg sends the new vertex v_i of G_ν to Alg$_\pi$, and finds out whether

[2] Note that the original lemma speaks about pseudo-random subgraphs, which is a stronger assumption that we do not need here.

$v_i \in S$. If $v_i \in S_{\text{in}}$, ALG answers 0 regardless of the answer of ALG_π. Similarly, if $v_i \in S_{\text{out}}$, ALG answers 1. Otherwise, ALG answers 0 if and only if $v_i \in S$.

First, note that ALG always produces a feasible solution: if the input $x_i = 1$, then either $v_i \notin S$ and ALG returns $y_i = 1$, or else v_i is included in S_{out}. Moreover, the number of zeros (the profit) in the output of ALG is $\min\{|S|, |I_\nu|\}$, where $|I_\nu|$ is the profit of the optimal solution. Since ALG_π is c-competitive, $|S| \geq 1/c \cdot \text{profit}(\text{OPT}(G_\nu)) \geq 1/c \cdot |I_\nu|$. ◀

▶ **Corollary 11.** *Let π be any non-trivial hereditary property. Let $A_{c,n}$ be the minimum advice needed for a c-competitive* MAX-Π *algorithm. Then*

$$B_c \cdot n - O(\log^2 n) \leq A_{c,n} \leq B_c \cdot n + O(\log n) \ .$$

We have shown that the advice complexity of MAX-Π essentially does not depend on the choice of the property π. Interestingly, this is not the case for cohereditary properties and MIN-Π. On the one hand, there are cohereditary properties where little advice is sufficient for optimality as the following theorem shows.

▶ **Theorem 12.** *If a cohereditary property π can be characterized by finitely many obligatory subgraphs, there is an optimal algorithm for* MIN-Π *with advice $O(\log n)$.*

Proof. Since each obligatory subgraph has constant size, $O(\log n)$ bits can be used to encode the indices of the vertices (forming the smallest obligatory subgraph) that are included in an optimal solution. ◀

On the other hand, there are properties for which MIN-Π requires large advice as stated by the following theorem, which was proven by Boyar et al. [7]. The problem *minimum cycle finding* requires to identify a smallest possible set of vertices S such that $G[S]$ contains a cycle. Hence, it is the MIN-Π problem for the non-trivial cohereditary property "contains cycle."

▶ **Theorem 13** (Boyar et al. [7]). *Any c-competitive algorithm for the* minimum cycle finding problem *must read at least*

$$B_c \cdot n - O(\log n)$$

bits of advice.

An upper bound analogous to Theorem 7 also follows from the results of Boyar et al. [7]. Note that, for the minimum cycle finding problem, this bound is tight up to an additive constant of $O(\log n)$.

▶ **Theorem 14.** *Let π be any non-trivial cohereditary property. There is a c-competitive algorithm for* MIN-Π *which reads*

$$B_c \cdot n + O(\log n)$$

bits of advice.

4 Max-Π with Preemption – Large Competitive Ratios

In this and the subsequent section, we consider the problem MAX-Π with preemption where π is a non-trivial hereditary property. In every time step, an online algorithm can either accept or reject the currently given vertex and preempt any number of vertices that it accepted in previous time steps. However, vertices that were once rejected or preempted cannot be accepted in later time steps. The goal is to accept as many vertices as possible. After each request, the current solution is required to have the property π.[3] Using a string guessing reduction, we can prove the following theorem; due to space constraints, we only give the idea.

▶ **Theorem 15.** *Consider the* Max-Π *problem with preemption for a hereditary property π with a forbidden subgraph H, such that π holds for all independent sets. Let $c(n)$ be an increasing function such that $c(n)\log c(n) = o(\sqrt{n/\log n})$. Any $c(n)$-competitive* Max-Π *algorithm must read at least*

$$\Omega\left(\frac{n}{c(n)^2 \log c(n)}\right)$$

bits of advice.

Proof Sketch. First, for some given n and σ, let us define the graph $G_{n,\sigma}$ that will be used in the reduction. To ease the presentation, assume that $n' = n/\sigma$ is integer. Let G_1 be a graph with σ vertices, the existence of which is asserted by Lemma 9, such that any subgraph of G_1 with at least $\kappa_1 \log \sigma$ vertices contains H as induced subgraph. Let G_B be the complement of a union of n' cliques of size σ each (i. e., G_B consists of n' independent sets $V_1, \ldots, V_{n'}$ of size σ each, and all remaining pairs of vertices are connected by edges). Applying Lemma 10 to G_B proves the existence of a graph $G_2 \subseteq G_B$ such that any subset of G_2 with at least $\kappa_2 \log n$ vertices contains H as an induced subgraph. The graph $G_{n,\sigma}$ is obtained from G_2 by replacing each independent set V_i with a copy of G_1 (each such copy is called a "layer" in what follows).

Let us suppose that a $c(n)$-competitive Max-Π algorithm Alg is given that uses $b(n)$ advice bits on instances of size n. Now fix an arbitrary n, and choose $\sigma := 4c\kappa_1 \log(4c\kappa_1)$. We show how to solve instances of σ-SGKH of length $n'-1$ using Alg. Let $q_1, \ldots, q_{n'-1}$ be the instance of σ-SGKH, where $q_i \in \{1, \ldots, \sigma\}$. The corresponding instance G for the Max-Π problem is as follows: take the graph $G_{n,\sigma}$, and denote by $v_{i,1}, \ldots, v_{i,\sigma}$ the vertices of the set V_i. Let v_{i,q_i} be the *distinguished* vertex in set V_i. Delete from $G_{n,\sigma}$ all edges of the form $\{v_{i,q_i}, v_{i',q_{i'}}\}$ where $i' > i$. The resulting graph G is presented to Alg in the order $v_{1,1}, \ldots, v_{1,\sigma}, v_{2,1}, \ldots, v_{2,\sigma}, \ldots$.

Note that G can be constructed online based on the instance $q_1, \ldots, q_{n'-1}$. The distinguished vertices form an independent set of size n', and thus a feasible solution. On the other hand, apart from the distinguished vertices, any solution can have at most $\kappa_1 \log \sigma$ vertices in one layer (otherwise, there would be a forbidden subgraph in that layer), and at most $\kappa_2 \log n$ layers with vertices other than the distinguished ones (if there are more than $\kappa_2 \log n$ nonempty layers, choose one vertex from each nonempty layer; these form a clique in G_B, and due to Lemma 10 induce H in G_2, and thus also in G). Hence, $n' \le \text{profit}(\text{Opt}(G)) \le n' + K$, where $K := \kappa_1\kappa_2 \log \sigma \log n$.

Since Alg is c-competitive, it produces a solution of size at least $\text{profit}(\text{Opt}(G))/c$. Since any solution can have at most K non-distinguished vertices, the solution of Alg contains at least $g := \text{profit}(\text{Opt}(G))/c - K$ distinguished vertices.

[3] Note that without preemption, the condition to maintain π in every time step is implicit. Indeed, if π is violated in some step, the algorithm has accepted a forbidden subgraph, which means that no matter how the sequence continues, the solution will ultimately be invalid. Let us emphasize that any algorithm that works for the case without preemption also works with preemption.

Consider an algorithm ALG$'$ for σ-SGKH on an instance of length $n'-1$, which simulates ALG. For the ith request, it presents ALG the layer of vertices V_i. Let $\mathrm{Cand}(i) \subseteq V_i$ (the *candidate* set) be the set of vertices selected by ALG from V_i. As stated before, $|\mathrm{Cand}(i)| \leq \kappa_1 \log \sigma$. A set $\mathrm{Cand}(i)$ is *good* if it contains the distinguished vertex v_{i,q_i}. It follows from the definition of the problem that there are at least g good candidate sets.

ALG$'$ uses an additional $O(\log\log\sigma)$ bits of advice to describe a number j with $1 \leq j \leq \kappa_1 \log \sigma$, and selects the jth vertex from any set $\mathrm{Cand}(i)$ as an answer (if $|\mathrm{Cand}(i)|$ is smaller than j, it is extended in an arbitrary fixed way). The number j is selected in such a way that ALG$'$ gives the correct answer for a fraction of $1/(\kappa_1 \log \sigma)$ of the good sets. As a result, the fraction of correctly guessed numbers by ALG$'$ is at least

$$\alpha := \frac{n' - cK}{c\kappa_1 \log \sigma (n'-1)} \ .$$

Note that $1/(c\kappa_1 \log\sigma) \geq \alpha \geq 1/(2c\kappa_1 \log \sigma)$ holds for large enough n, provided that $n' \geq 2cK - 1$. To see that this inequality holds, note that

$$n' \geq 2cK - 1 \iff \frac{n}{4c\kappa_1 \log(4c\kappa_1)} \geq 2cK - 1 \iff (2cK-1)4c\kappa_1 \log(4c\kappa_1) \leq n \ .$$

The last inequality holds for large enough n by the choice of $c(\cdot)$ due to the fact that

$$(2cK-1)4c\kappa_1 \log(4c\kappa_1) \in O(c(n)^2 K \log c(n)) = O((c(n)\log c(n))^2 \log n) = o(n) \ .$$

Due to Theorem 4, any algorithm for σ-SGKH that correctly guesses a fraction of α numbers (for $1/\sigma \leq \alpha \leq 1$) on an input of length $n'-1$ requires at least $b := F(\sigma,\alpha)\cdot(n'-1)\cdot\log\sigma$ bits of advice where

$$F(\sigma,\alpha) := 1 + (1-\alpha)\log_\sigma\left(\frac{1-\alpha}{\sigma-1}\right) + \alpha \log_\sigma \alpha \ .$$

It can be shown that $F(\sigma,\alpha)\log\sigma \in \Omega(1/c)$. Finally, the theorem follows by noting that $n'-1 \in \Omega(n/(c\log c))$. ◀

Using a similar approach, we can get a stronger bound for the independent set problem; the proof is omitted due to space constraints.

▶ **Theorem 16.** *Let $c(n)$ be any function such that*

$$8 \leq c(n) \leq \frac{1+\sqrt{1+4n}}{4} \ .$$

Any $c(n)$-competitive independent set algorithm that can use preemption must read at least

$$\frac{0.01 \cdot \log(2c)}{2c^2}(n-2c)$$

bits of advice.

5 Max-π with Preemption – Small Competitive Ratios

In this section, we use Theorem 5 to give bounds for small constant values of the competitive ratio for algorithms for MAX-π complementing the bounds from Theorem 15. In what follows, π is a non-trivial hereditary property and k is the size of a smallest forbidden subgraph with respect to π.

▶ **Theorem 17.** *If there is a c-competitive algorithm for* Max-Π *with preemption that reads $b(kn)$ bits of advice for inputs of length kn, then there exists a c-competitive algorithm for Anti-k-SGKH, which, for inputs of length n, reads*

$$b(kn) + O(\log^2 n)$$

bits of advice.

Proof. According to Lemma 1, π is satisfied either by all cliques or by all independent sets. As in the proof of Theorem 8, we assume in the following that π is satisfied by all independent sets (if it is not, we can use the same argument by swapping edges and non-edges between layers). We describe how to transform an instance of Anti-k-SGKH into an instance of Max-Π with preemption. The length of the instance for Max-Π with preemption will be k times as long as the length n of the Anti-k-SGKH instance. We proceed to show that a c-competitive algorithm for the latter implies a c-competitive algorithm for the former which reads at most $O((\log n)^2)$ additional advice bits.

Let $\nu = x_1, \ldots, x_n$ with $x_i \in \{1, \ldots, k\}$ be an instance of Anti-k-SGKH. Consider the n-vertex graph $\tilde{G} = (V(\tilde{G}), E(\tilde{G}))$ given by Lemma 9 for a size-k smallest minimal forbidden subgraph $H = (V(H), E(H))$ for π. Recall that any induced subgraph of $\tilde{G}$ with at least $\alpha \log n$ vertices contains H as an induced subgraph. Let us denote $V(\tilde{G}) = \{\tilde{v}_1, \ldots, \tilde{v}_n\}$ and $V(H) = \{h_1, \ldots, h_k\}$. We now describe the construction of a graph $G_\nu = (V(G_\nu), E(G_\nu))$, which will be the input for the given algorithm for Max-Π. To this end, let

$$V(G_\nu) := \bigcup_{i=1}^{n} \bigcup_{j=1}^{k} v_j^i,$$

$$E(G_\nu) := \{\{v_j^i, v_{j'}^i\} \mid \{h_j, h_{j'}\} \in E(H)\} \cup \{\{v_j^i, v_{j'}^{i'}\} \mid i < i', \{\tilde{v_j}, \tilde{v_{j'}}\} \in E(\tilde{G}), j \neq x_i\},$$

where we assume an ordering $v_1^1, \ldots, v_k^1, v_1^2, \ldots, v_k^2, \ldots, v_1^n, \ldots, v_k^n$ on the vertices. Moreover, we denote the requests $v_1^i, \ldots, v_k^i$ as layer i. Let X denote the set of vertices $v_{x_i}^i$ for $i \in \{1, \ldots, n\}$.

We start with a few observations about G_ν that are straightforward.

▶ **Observation 18.** *$G_\nu[X]$ is an independent set of size n. In particular, it has property π.*

▶ **Observation 19.** *$G_\nu[v_1^i, \ldots, v_k^i] = H$ for an arbitrary but fixed i. Thus, any induced subgraph of G_ν that contains $G_\nu[v_1^i, \ldots, v_k^i]$ does not have property π.*

▶ **Observation 20.** *Consider a set of vertices, V, in G_ν which is disjoint from X. If $|V| \geq k\alpha \log n$, then $G_\nu[V]$ does not have property π. Note that V must in this case contain vertices from at least $\alpha \log n$ different layers. These have H as an induced subgraph since none of them are in X.*

Now consider a c-competitive algorithm Alg$_\pi$ for Max-Π with preemption reading $b(kn)$ bits of advice (recall that kn is the length of its input G_ν). We start by describing an algorithm Alg$'$ for Anti-k-SGKH, which uses $b(kn)$ bits of advice (n is the length of its input). Afterwards, we use Alg$'$ to define another algorithm Alg for Anti-k-SGKH, which uses $O(\log^2 n)$ additional advice bits and is c-competitive.

For a given string $\nu = x_1, \ldots, x_n$, let $\perp, x_1, \ldots, x_n$ be the input for Anti-k-SGKH. Let S be the solution (set of vertices) returned by Alg$_\pi$ on G_ν (with the proper advice). Note that this is the resulting set of vertices after the unwanted vertices have been preempted. Alg$'$ works as follows: It constructs the graph G_ν online and simulates Alg$_\pi$ on it. When a request i arrives, the goal of Alg$'$ is to guess a number in $\{1, \ldots, k\}$ different from x_i.

It does this by presenting all vertices in layer i to $\textsc{Alg}_\pi$. It is important to note that the vertices in layer i can be presented without knowledge of $x_i, \ldots, x_n$. Let S_i denote the set of these vertices, which are accepted by $\textsc{Alg}_\pi$ and have not been preempted after request v_k^i. In layer i, $\textsc{Alg}'$ outputs $y_i = w$ where w is the smallest number in $\{1, \ldots, k\}$ such that $v_w^i \notin S_i$. Note that such a number always exists due to Observation 19.

We now describe $\textsc{Alg}$, which uses $O(\log^2 n)$ additional advice bits. The advice for $\textsc{Alg}$ consists of three parts (similar to the proof of Theorem 8). First, it contains a self-delimiting encoding of n (this requires $O(\log n)$ bits). This is followed by a list of up to $k\alpha \log n$ indices i, where $\textsc{Alg}'$ outputs $y_i = x_i$. Let S_{error} denote the set of these indices. A self-delimiting encoding of this requires $O(\log^2 n)$ bits (recall that α and k are constant). Finally, the advice which $\textsc{Alg}'$ received is included. This is $b(kn)$ bits.

$\textsc{Alg}$ works as follows for each request. If the request is not in S_{error}, it outputs the same as $\textsc{Alg}'$. Conversely, if the request is in S_{error}, it outputs another number in $\{1, \ldots, k\}$.

We now argue that $\textsc{Alg}$ is c-competitive. Note that the optimal offline solution for G_ν contains at most $k\alpha \log n$ vertices not in X. The same of course holds for the solution produced by $\textsc{Alg}_\pi$. Moreover, it holds that if in layer i the algorithm $\textsc{Alg}_\pi$ accepts a vertex in X, then $\textsc{Alg}'$ outputs $y_i \neq x_i$. This means that the score of $\textsc{Alg}_\pi$ is at most $k\alpha \log n$ more than the score of $\textsc{Alg}'$. Since the score of $\textsc{Alg}$ is $k\alpha \log n$ more than the score of $\textsc{Alg}'$, we have that $\textsc{Alg}$ is c-competitive. ◀

Combining Theorems 5 and 17, we get the following corollary.

▶ **Corollary 21.** *Let $1 < c < k/(k-1)$. Let π be any non-trivial hereditary property with a minimal forbidden subgraph of size k. Any c-competitive algorithm for* $\textsc{Max}$-Π *with preemption must read at least*

$$\left(1 - h_k\left(\frac{1}{c}\right)\right) n \frac{\log k}{k} - O(\log^2 n)$$

bits of advice, where n is the input length. Here, h_k is the k-ary entropy function given by $h_k(x) = x \log_k(k-1) - x \log_k x - (1-x)\log_k(1-x)$.

6 Closing Remarks

In Corollary 11, we describe lower and upper bounds for the advice complexity of all online hereditary graph problems, which are essentially tight (there is just a gap of $O(\log^2 n)$). It turns out that, for all of them, roughly the same amount of information about the future is required to achieve a certain competitive ratio.

Intriguingly, we see quite a different picture for cohereditary properties. Theorem 14 gives the same upper bound as we had for hereditary properties, and Theorem 13 shows that this upper bound is essentially tight. However, Theorem 12 shows that there exist cohereditary problems that have an advice complexity as low as $O(\log n)$ bits to be optimal. It remains open if it is only those problems with a finite set of obligatory graphs that have this very low advice complexity, or if this can also happen for cohereditary problems with an infinite set of obligatory graphs.

For hereditary problems with preemption, we show that to achieve a competitive ratio strictly smaller than $k/(k-1)$, a linear number of advice bits is needed. This is asymptotically tight, since optimality (even without preemption) can be achieved with n bits. Furthermore, we show a lower bound for non-constant competitive ratios (that are roughly smaller than $\sqrt{n}$). It remains open if there is an algorithm for the preemptive case which uses fewer advice bits than the algorithms solving the same problem in the non-preemptive case.

References

1 Y. Bartal, A. Fiat, and S. Leonardi. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. *SIAM Journal on Computing*, 36(2):354–393, 2006.

2 H.-J. Böckenhauer, J. Hromkovič, D. Komm, S. Krug, J. Smula, and A. Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theoretical Computer Science* 554:95–108, 2014.

3 H.-J. Böckenhauer, D. Komm, R. Královič, and R. Královič. On the advice complexity of the k-server problem. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Proc. of ICALP 2011*, volume 6755 of *LNCS*, pp. 207–218. Springer-Verlag, Berlin, 2011.

4 H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič, and T. Mömke. On the advice complexity of online problems. In Y. Dong, D.-Z. Du, and O. H. Ibarra, editors, *Proc. of ISAAC 2009*, volume 5878 of *LNCS*, pp. 331–340. Springer-Verlag, Berlin, 2009.

5 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

6 A. Borodin and R. El-Yaniv. On randomization in on-line computation. *Information and Computation*, 150(2):244–267, 1999.

7 J. Boyar, L. M. Favrholdt, C. Kudahl, and J. W. Mikkelsen. Advice Complexity for a Class of Online Problems. In E. W. Mayr and N. Ollinger, editors, *Proc. of STACS 2015*, volume 30 of *LIPIcs*, pp. 116–129, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

8 R. Diestel. *Graph Theory.* Springer-Verlag, Berlin, 4th edition, 2010.

9 S. Dobrev, R. Královič, and D. Pardubská. Measuring the problem-relevant information in input. *RAIRO Theoretical Informatics and Applications*, 43(3):585–613, 2009.

10 S. Dobrev, R. Královič, and R. Královič: Advice complexity of maximum independent set in sparse and bipartite graphs. *Theory of Computing Systems* 56(1):197–219, 2015.

11 Y. Emek, P. Fraigniaud, A. Korman, and A. Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.

12 J. Hromkovič, R. Královič, and R. Královič. Information complexity of online problems. In F. Murlak and P. Sankowski, editors, *Proc. of MFCS 2010*, volume 6281 of *LNCS*, pp. 24–36. Springer-Verlag, Berlin, 2010.

13 D. Komm and R. Královič. Advice complexity and barely random algorithms. *RAIRO Theoretical Informatics and Applications*, 45(2):249–267, 2011.

14 I. Csiszár. The method of types. *Information Theory, IEEE Transactions on* 44(6):2505–2523, 1998.

15 J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

16 C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In A. Lingas, R. Karlsson, and S. Carlsson, editors, *Proc. of ICALP 1993*, volume 700 of *LNCS*, pp. 40–51. Springer-Verlag, Berlin, 1993.

17 F. P. Ramsey. On a problem in formal logic. *Proc. London Mathematical Society (3)*, 30:264–286, 1930.

18 D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

19 A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proc. of FOCS 1977*, pp. 222–227. IEEE Computer Society, Los Alamitos, 1977.

20 J. W. Mikkelsen. Randomization can be as helpful as a glimpse of the future in online computation. *CoRR*, abs/1511.05886, 2015.

Decidability of Predicate Logics with Team Semantics

Juha Kontinen[1], Antti Kuusisto[2], and Jonni Virtema[3]

1 **University of Helsinki, Helsinki, Finland**
juha.kontinen@helsinki.fi
2 **University of Bremen, Bremen, Germany**
kuusisto@uni-bremen.de
3 **University of Helsinki, Helsinki, Finland, and**
Leibniz Universität Hannover, Hannover, Germany
jonni.virtema@gmail.com

Abstract

We study the complexity of predicate logics based on team semantics. We show that the satisfiability problems of two-variable independence logic and inclusion logic are both *NEXPTIME*-complete. Furthermore, we show that the validity problem of two-variable dependence logic is undecidable, thereby solving an open problem from the team semantics literature. We also briefly analyse the complexity of the Bernays-Schönfinkel-Ramsey prefix classes of dependence logic.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Team semantics, dependence logic, complexity, two-variable logic

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.60

1 Introduction

The satisfiability problem of *two-variable logic* FO^2 was shown to be *NEXPTIME*-complete in [9]. The extension of two-variable logic with counting quantifiers, FOC^2, was proved decidable in [10, 22], and it was subsequently shown to be *NEXPTIME*-complete in [23]. Research on extensions and variants of two-variable logic is currently very active. Recent research efforts have mainly concerned decidability and complexity issues in restriction to particular classes of structures and also questions related to different built-in features and operators that increase the expressivity of the base language. Recent articles in the field include for example [1], [4], [13], [16], [24], and several others.

In this article we study two-variable fragments of logics based on *team semantics*. Team semantics was originally conceived in [15] in the context of *independence friendly* (IF) *logic* [14]. In [25], Väänänen introduced *dependence logic*, which is a novel approach to IF logic based on new atomic formulas $=(x_1, ...x_k, y)$ stating that the interpretation of the variable y is functionally determined by the interpretations of the variables $x_1, ..., x_k$.

After the introduction of dependence logic, research on logics based on team semantics has been active. Several different logics with different applications have been suggested. In particular, team semantics has proved to be a powerful framework for studying different kinds of *dependency notions*. *Independence logic* [11] is a variant of dependence logic that extends first-order logic by new atomic formulas $x_1, ..., x_k \perp y_1, ..., y_l$ with the intuitive meaning that the interpretations of the variables $x_1, ..., x_k$ are informationally independent of the interpretations of the variables $y_1, ..., y_l$. *Inclusion logic* [6] extends first-order logic by atomic formulas $x_1, ..., x_k \subseteq y_1, ..., y_k$, whose intuitive meaning is that tuples interpreting the variables $x_1, ..., x_k$ are also tuples interpreting $y_1, ..., y_k$. Currently dependence, independence and inclusion logics are the three most important and most widely studied systems based on team semantics.

Both dependence logic and independence logic are equiexpressive with existential second-order logic (see [25], [11]), and thereby capture NP. Curiously, inclusion logic is equiexpressive with *greatest fixed point logic* (see [7]), and thereby characterizes P on finite ordered models. While the descriptive complexity of most known logics based on team semantics is understood reasonably well, the complexity of related satisfiability problems has received somewhat less attention. The satisfiability problem of the two-variable fragment of dependence logic and IF-logic have been studied in [18]. It is shown that while the two-variable IF-logic is undecidable, the corresponding fragment of dependence logic is *NEXPTIME*-complete.

In this article we establish that the satisfiablity problems of the two-variable fragments of independence and inclusion logics are likewise *NEXPTIME*-complete. This result is established via proving a more general theorem that implies also a range of other decidability results for a variety of team-semantics-based logics with generalized dependency notions. Furthermore, we prove that the *validity* problem of two-variable dependence logic is undecidable; this result is the main result of the paper. The problem has been open for some time in the team semantics literature and has been explicitly posed in, e.g., [5], [18], [26], and elsewhere.

In addition to studying two-variable logics, we study the Bernays-Schönfinkel-Ramsey prefix class, i.e., sentences with the quantifier prefix $\exists^*\forall^*$. We show that—as in the case of ordinary first-order logic—the prefix class $\exists^*\forall^*$ of $\mathsf{FO}(\mathcal{A})$ is decidable for any uniformly polynomial time computable class $\mathcal{A}$ of generalized dependencies closed under substructures. We prove inclusion in 2*NEXPTIME*, and furthermore, for vocabularies of fixed arity, we show *NEXPTIME*-completeness. We also prove a partial converse of the result concerning logics $\mathsf{FO}(\mathcal{A})$ with a decidable prefix class $\exists^*\forall^*$, see Theorem 22.

2 Preliminaries

The domain of a structure $\mathfrak{A}$ is denoted by A. We assume that the reader is familiar with first-order logic FO. The extension of FO with counting quantifiers $\exists^{\geq i}$ is denoted by FOC. The two-variable fragments FO^2 and FOC^2 are the fragments of FO and FOC with formulas in which only the variables x and y appear. We let Σ_1^1 denote the fragment of formulas of second-order logic of the form $\exists X_1 ... \exists X_k\, \varphi$, where $X_1, ..., X_k$ are relation symbols and φ a first-order formula. $\Sigma_1^1(\mathsf{FOC}^2)$ is the extension of FOC^2 consisting of formulas of the form $\exists X_1 ... \exists X_k\, \chi$, where $X_1, ..., X_k$ are relation symbols and χ a formula of FOC^2.

2.1 Logics based on team semantics

Let $\mathbb{Z}_+$ denote the set of positive integers, and let $\mathrm{VAR} = \{ v_i \mid i \in \mathbb{Z}_+ \}$ be the set of exactly all first-order *variable symbols*. We mainly use metavariables x, y, z, x_1, x_2, etc., in order to refer to variable symbols in VAR. We let $\overline{x}, \overline{y}, \overline{z}, \overline{x}_1, \overline{x}_2$, etc., denote finite nonempty tuples of variable symbols, i.e., tuples in VAR^n for some $n \in \mathbb{Z}_+$. When we study two-variable logics, we use the metavariables x and y, and assume they denote distinct variables in VAR.

Let $D \subseteq \mathrm{VAR}$ be a *finite*, possibly empty set. Let $\mathfrak{A}$ be a model. We do not allow for models to have an empty domain, so $A \neq \emptyset$. A function $s : D \to A$ is called an *assignment* with codomain A. If $\overline{x} = (x_1, \ldots, x_n)$, we denote $(s(x_1), \ldots, s(x_n))$ by $s(\overline{x})$. We let $s[a/x]$ denote the variable assignment with the domain $D \cup \{ x \}$ and codomain A defined such that $s[a/x](y) = a$ if $y = x$, and $s[a/x](y) = s(y)$ if $y \neq x$. Let $T \in \mathcal{P}(A)$, where $\mathcal{P}$ denotes the power set operator. We define $s[\,T/x\,] \;=\; \{\; s[a/x] \mid a \in T \;\}$.

Let $D \subseteq \mathrm{VAR}$ be a finite, possibly empty set of first-order variable symbols. Let X be a set of assignments $s : D \to A$. Such a set X is a *team* with the *domain* D and *codomain* A. Note that the empty set is a team, as is the set $\{\emptyset\}$ containing only the empty assignment. The team $\emptyset$ does not have a unique domain; any finite subset of VAR is a domain of $\emptyset$. The domain of the team $\{\emptyset\}$ is $\emptyset$.

Let X be a team with the domain D and codomain A. Let $T \subseteq A$. We define $X[T/x] = \{\; s[a/x] \mid a \in T,\;\; s \in X \;\}$. Let $F : X \to \mathcal{P}(A)$ be a function. We define $X[\,F/x\,] = \bigcup_{s\in X} s[\,F(s)/x\,]$. Let $C \subseteq A$. We define $X \upharpoonright C = \{\, s \in X \mid s(x) \in C$ for all $x \in D \,\}$.

Let X be a team with domain D. Let $k \in \mathbb{Z}_+$, and let $y_1, ..., y_k$ be variable symbols. Assume that $\{y_1, ..., y_k\} \subseteq D$. We define $\mathrm{rel}\big(X, (y_1, ..., y_k)\big) \;=\; \{\; \big(s(y_1), ..., s(y_k)\big) \mid s \in X \;\}$.

Let τ be a relational vocabulary, i.e., a vocabulary containing relation symbols only. (In this article we consider only relational vocabularies.) The syntax of a logic based on team semantics is usually given in negation normal form. We shall also follow this convention in the current article. For this reason, we define the syntax of first-order logic as follows.

$$\varphi \quad ::= \quad R(\overline{x}) \mid \neg R(\overline{x}) \mid x_1 = x_2 \mid \neg x_1 = x_2 \mid (\varphi_1 \vee \varphi_2) \mid (\varphi_1 \wedge \varphi_2) \mid \;\exists x\varphi \mid \forall x\varphi \;,$$

where $R \in \tau$. The first four formula formation rules above introduce *first-order literals* to the language. Below we shall consider logics $\mathsf{FO}(\mathcal{A})$, where the above syntax is extended by clauses of the type $A_Q\,(\overline{y}_1, ..., \overline{y}_k)$. Here A_Q is (a symbol corresponding to) a *generalized atom* in $\mathcal{A}$ and each $\overline{y}_i$ is a tuple of variables. Before considering such novel atoms, let us define *lax team semantics* for first-order logic.

▶ **Definition 1** ([15, 25])**.** Let $\mathfrak{A}$ be a model and X a team with codomain A. The satisfaction relation $\mathfrak{A} \models_X \varphi$ is defined as follows.

1. If φ is a first-order literal, then $\mathfrak{A} \models_X \varphi$ iff for all $s \in X$: $\mathfrak{A}, s \models_{\mathsf{FO}} \varphi$. Here $\models_{\mathsf{FO}}$ refers to the ordinary Tarskian satisfaction relation of first-order logic.
2. $\mathfrak{A} \models_X \psi \wedge \varphi$ iff $\mathfrak{A} \models_X \psi$ and $\mathfrak{A} \models_X \varphi$.
3. $\mathfrak{A} \models_X \psi \vee \varphi$ iff there exist teams Y and Z such that $X = Y \cup Z$, $\mathfrak{A} \models_Y \psi$, and $\mathfrak{A} \models_Z \varphi$.
4. $\mathfrak{A} \models_X \exists x\, \psi$ iff $\mathfrak{A} \models_{X[F/x]} \psi$ for some $F\colon X \to (\mathcal{P}(A) \setminus \{\emptyset\})$.
5. $\mathfrak{A} \models_X \forall x\, \psi$ iff $\mathfrak{A} \models_{X[A/x]} \psi$.

Finally, a sentence φ is true in a model $\mathfrak{A}$ ($\mathfrak{A} \models \varphi$) if $\mathfrak{A} \models_{\{\emptyset\}} \varphi$.

▶ **Proposition 2** ([15, 25])**.** *Let ψ be a formula of first-order logic. We have $\mathfrak{A} \models_X \psi$ iff $\mathfrak{A}, s \models_{\mathsf{FO}} \psi$ for all $s \in X$.*

In this paper we consider first-order logic extended with generalized dependency atoms. Before formally introducing the notion of a generalized dependency atom, we recall some particular atoms familiar from the literature related to team semantics.

Dependence atoms $=\!(\overline{x}, y)$, inspired by the slashed quantifiers of Hintikka and Sandu [14], were introduced by Väänänen [25]. The intuitive meaning of the atom $=\!(\overline{x}, y)$ is that the value of the variable y depends solely on the values of the variables in $\overline{x}$. The semantics for dependence atoms is defined as follows: $\mathfrak{A} \models_X =\!(\overline{x}, y)$ iff $\forall s, s' \in X :$ if $s(\overline{x}) = s'(\overline{x})$ then $s(y) = s'(y)$. *Dependence logic* (D) is the extension of first-order logic with dependence atoms.

While dependence atoms of dependence logic declare dependences between variables, *independence atoms*, introduced by Grädel and Väänänen [11], do just the opposite; independence atoms are used to declare independencies between variables. Independence atom is an atomic formula of the form $(x_1, ..., x_k) \perp_{(z_1,...,z_t)} (y_1, ..., y_l)$ with the intuitive meaning that for any fixed interpretation of the variables $z_1, \ldots, z_t$, the interpretations of the variables $x_1, ..., x_k$ are independent of the interpretations of the variables $y_1, ..., y_l$. The semantics for independence atoms is defined as follows:

$$\mathfrak{A} \models_X (x_1, ..., x_k) \perp_{(z_1,...,z_t)} (y_1, ..., y_l) \text{ iff } \forall s, s' \in X\, \exists s'' \in X : \bigwedge_{i \le t} s(z_i) = s'(z_i)$$

$$\text{implies that } \bigwedge_{i \le k} s''(x_i) = s(x_i) \wedge \bigwedge_{i \le t} s''(z_i) = s(z_i) \wedge \bigwedge_{i \le l} s''(y_i) = s'(y_i).$$

Independence logic (Ind) is the extension of first-order logic with independence atoms.

Galliani [6] introduced *inclusion* and *exclusion atoms*. The intuitive meaning of the inclusion atom $(x_1, \ldots, x_n) \subseteq (y_1, \ldots, y_n)$ is that tuples interpreting the variables $x_1 \ldots, x_n$ are also tuples interpreting $y_1, \ldots, y_n$. The intuitive meaning of the exclusion atom $(x_1, \ldots, x_n) \mid (y_1, \ldots, y_n)$ on the other hand is that tuples interpreting the variables $x_1 \ldots, x_n$ and the tuples interpreting $y_1, \ldots, y_n$ are distinct. The semantics for inclusion atoms and exclusion atoms is defined as follows:

$$\mathfrak{A} \models_X \overline{x} \subseteq \overline{y} \text{ iff } \forall s \in X\, \exists s' \in X : s(\overline{x}) = s'(\overline{y}), \quad \mathfrak{A} \models_X \overline{x} \mid \overline{y} \text{ iff } \forall s, s' \in X : s(\overline{x}) \neq s'(\overline{y}).$$

The extension of first-order logic with inclusion atoms (exclusion atoms) is called *inclusion logic* (*exclusion logic*) and denoted by Inc (Exc). The extension of first-order logic with both inclusion atoms and exclusion atoms is called *inclusion/exclusion logic* and denoted by Inc/Exc.

2.2 Generalized atoms

In this section we first give the well known definition of generalized quantifiers (Lindström quantifiers [21]). We then show how each generalized quantifier naturally gives rise to a generalized atom. Finally, we discuss on some fundamental properties of first-order logic extended with generalized atoms. Generalized atoms were first defined in [20].

Let $(i_1, ..., i_n)$ be a nonempty sequence of positive integers. A generalized quantifier of the type $(i_1, ..., i_n)$ is a class $\mathcal{C}$ of structures $(A, B_1, ..., B_n)$ such that the following conditions hold.

1. $A \neq \emptyset$, and for each $j \in \{1, ..., n\}$, we have $B_j \subseteq A^{i_j}$.
2. If $(A', B'_1, ..., B'_n) \in \mathcal{C}$ and if there is an isomorphism $f : A' \to A''$ from $(A', B'_1, ..., B'_n)$ to another structure $(A'', B''_1, ..., B''_n)$, then $(A'', B''_1, ..., B''_n) \in \mathcal{C}$.

Let Q be a generalized quantifier of the type $(i_1, ..., i_n)$. Let $\mathfrak{A}$ be a model with the domain A. We define $Q^{\mathfrak{A}}$ to be the set $\{\ (B_1, ..., B_n) \mid (A, B_1, ..., B_n) \in Q\ \}$.

Let n be a positive integer. Let Q be a generalized quantifier of the type $(i_1, ..., i_n)$. Extend the syntax of first-order logic with atomic expressions of the type $A_Q(\overline{y}_1, ..., \overline{y}_n)$, where each $\overline{y}_j$ is a tuple of variables of length i_j. Let X be a team whose domain contains all variables occurring in the tuples $\overline{y}_1, ..., \overline{y}_n$. Extend team semantics such that $\mathfrak{A} \models_X A_Q(\overline{y}_1, ..., \overline{y}_n)$ if and only if $\big(\mathrm{rel}(X, \overline{y}_1), ..., \mathrm{rel}(X, \overline{y}_n)\big) \in Q^{\mathfrak{A}}$. The generalized quantifier Q defines a *generalized atom* A_Q of the type $(i_1, ..., i_n)$.

A generalized atom A_Q is *downwards closed* if for all $\mathfrak{A}$, X and $\overline{y}_1, ..., \overline{y}_k$, it holds that if $\mathfrak{A} \models_X A_Q(\overline{y}_1, ..., \overline{y}_k)$ and $Y \subseteq X$, then $\mathfrak{A} \models_Y A_Q(\overline{y}_1, ..., \overline{y}_k)$. Similarly, a generalized atom A_Q is *closed under substructures* if for all $\mathfrak{A}$, X and $\overline{y}_1, ..., \overline{y}_k$, it holds that if $\mathfrak{A} \models_X A_Q(\overline{y}_1, ..., \overline{y}_k)$, $\mathfrak{A}' := \mathfrak{A} \upharpoonright B$ and $X' := X \upharpoonright B$ for some $B \subseteq A$, then we have $\mathfrak{A}' \models_{X'} A_Q(\overline{y}_1, ..., \overline{y}_k)$. Finally, a generalized atom A_Q is *universe independent* if for all $\mathfrak{A}$, $\mathfrak{B}$, X and $\overline{y}_1, ..., \overline{y}_k$, where both A and B are codomains for X, it holds that $\mathfrak{A} \models_X A_Q(\overline{y}_1, ..., \overline{y}_k)$ if and only if $\mathfrak{B} \models_X A_Q(\overline{y}_1, ..., \overline{y}_k)$.

Let φ be a formula of first-order logic, possibly extended with generalized atoms. The set $\mathrm{Fr}(\varphi)$ of *free variables* of φ is defined in the same way as in first-order logic. The set $\mathrm{Fr}(A_Q(\overline{y}_1, ..., \overline{y}_k))$ of course contains exactly all variable that occur in the tuples $\overline{y}_i$. The satisfiability problem of a (possibly team-semantics-based) logic L takes as an input a sentence of L and asks whether $\mathfrak{A} \models \varphi$ for some model $\mathfrak{A}$. The validity problem asks, given a sentence φ, whether $\mathfrak{A} \models \varphi$ for all models $\mathfrak{A}$.

Let $k \in \mathbb{Z}_+$ and let A_Q be a generalized atom of the type $(i_1, ..., i_n)$, where $i_j \leq k$ for each j. Let $\varphi(R_1, ..., R_n)$ be a sentence of $\Sigma_1^1(\mathsf{FOC}^k)$ with unquantified relation symbols $R_1, ..., R_n$ of arities $i_1, ..., i_n$, respectively. Assume that for all models $\mathfrak{A}$ and teams X with codomain A and domain containing the variables in $A_Q(\overline{x}_1, ..., \overline{x}_n)$, we have $\mathfrak{A} \models_X A_Q(\overline{x}_1, ..., \overline{x}_n)$ iff

$$\big(\mathfrak{A}, R_1 := \mathrm{rel}(X, \overline{x}_1), ..., R_n := \mathrm{rel}(X, \overline{x}_n)\big) \models_{\mathsf{FO}} \varphi(R_1, ..., R_n).$$

Then we say that the atom A_Q is definable in $\Sigma_1^1(\mathsf{FOC}^k)$.

We now show that, *for any generalized atom* A_Q, the logic $\mathsf{FO}(A_Q)$ has the so-called locality property. We also show that, for a downwards closed atom A_Q, all formulas of $\mathsf{FO}(A_Q)$ satisfy the downwards closure property. These two properties have previously turned out to be very useful in the study of dependence logic.

Let X be a team with domain $\{x_1, \ldots, x_k\}$, and let $V \subseteq \{x_1, \ldots, x_k\}$. We denote by $X(V)$ the team $\{s \upharpoonright V \mid s \in X\}$ with the domain V. The following proposition shows that the truth of an $\mathsf{FO}(A_Q)$-formula depends only on the interpretations of the variables occurring free in the formula. The proof uses the fact that generalized atoms satisfy the claim by definition. Otherwise the proof is identical to the corresponding proof given in [6].

▶ **Proposition 3** (Locality). *Let A_Q be a generalized atom and $\varphi \in \mathsf{FO}(A_Q)$ a formula. If $V \supseteq \mathrm{Fr}(\varphi)$, then $\mathfrak{A} \models_X \varphi$ if and only if $\mathfrak{A} \models_{X(V)} \varphi$.*

The next proposition is also very useful. The proof is almost identical to the corresponding proof for dependence logic, see [25]. The additional case for generalized atoms follows by the assumption of downwards closure.

▶ **Proposition 4** (Downward closure). *Let A_Q be a downwards closed generalized atom. Suppose φ is an $\mathsf{FO}(A_Q)$-formula, $\mathfrak{A}$ a model, and $Y \subseteq X$ teams. Then $\mathfrak{A} \models_X \varphi$ implies $\mathfrak{A} \models_Y \varphi$.*

3 Satisfiability problems of logics $\mathsf{FO}^2(\mathcal{A})$

In this section we show that for any finite collection $\mathcal{A}$ of $\Sigma_1^1(\mathsf{FOC}^2)$-definable atoms A_Q, both $\textsc{Sat}(\mathsf{FO}^2(\mathcal{A}))$ and $\textsc{FinSat}(\mathsf{FO}^2(\mathcal{A}))$ are *NEXPTIME*-complete. Our proof relies on a translation from $\mathsf{FO}^2(\mathcal{A})$ into $\Sigma_1^1(\mathsf{FOC}^2)$ and the fact that $\textsc{Sat}(\mathsf{FOC}^2)$ and $\textsc{FinSat}(\mathsf{FOC}^2)$ are *NEXPTIME*-complete [23].

We start by establishing a more general translation. We show that for every $k \geq 1$ and every $\Sigma_1^1(\mathsf{FOC}^k)$ definable atom A_Q, we have $\mathsf{FO}^k(A_Q) \leq \Sigma_1^1(\mathsf{FOC}^k)$. Note that strictly speaking $\mathsf{FO}^k(A_Q)$ uses only one atom A_Q instead of a finite collection $\mathcal{A}$ of atoms, but our proof below generalizes *directly* to the case with a finite collection of atoms. The reason for considering a single atom is simply to keep the notation light.

When considering k-variable logic, we let $\{x_1, ..., x_k\}$ denote the k distinct variables used in the syntax of the logic, and we let $\mathit{rel}(X)$ denote $\mathit{rel}\big(X, (x_1, ..., x_k)\big)$. The proof of the following lemma (see the full version in arXiv [19]) significantly modifies and extends the argument establishing Lemma 3.3.14 of [26]. See also [18] and Theorem 6.2 in [25].

▶ **Lemma 5.** *Assume that $k, t \geq 1$. Let τ be a relational vocabulary, let $R \notin \tau$ be a k-ary relation symbol and let A_Q be a $\Sigma^1_1(\mathsf{FOC}^k)$-definable atom of type $(i_1, \ldots, i_t)$, where $i_j \leq k$ for each j. For every formula $\varphi \in \mathsf{FO}^k(A_Q)$ there exists a sentence $\varphi^* \in \Sigma^1_1(\mathsf{FOC}^k)(\tau \cup \{R\})$ such that for every model $\mathfrak{A}$ and team X with codomain A and $\mathrm{dom}(X) = \{x_1, \ldots, x_k\}$, we have*

$$\mathfrak{A} \models_X \varphi \quad \textit{iff} \quad \big(\mathfrak{A}, \mathrm{rel}(X)\big) \models \varphi^*, \tag{1}$$

where $(\mathfrak{A}, \mathrm{rel}(X))$ is the expansion $\mathfrak{A}'$ of $\mathfrak{A}$ into the vocabulary $\tau \cup \{R\}$ such that $R^{\mathfrak{A}'} := \mathrm{rel}(X)$. Moreover φ^* is computable from φ in polynomial time.

▶ **Theorem 6.** *For every $k \geq 1$ and for every $\Sigma^1_1(\mathsf{FOC}^k)$-definable atom A_Q it holds that $\mathsf{FO}^k(A_Q) \leq \Sigma^1_1(\mathsf{FOC}^k)$, i.e., for every sentence of $\mathsf{FO}^k(A_Q)$, there exists an equivalent sentence of $\Sigma^1_1(\mathsf{FOC}^k)$.*

Proof. Let τ be a relational vocabulary, $k \geq 1$, and A_Q a $\Sigma^1_1(\mathsf{FOC}^k)$-definable atom. Let φ be an $\mathsf{FO}^k(A_Q)(\tau)$-sentence and $\varphi^* = \exists R_1 \ldots \exists R_n \psi$ the related $\Sigma^1_1(\mathsf{FOC}^k)(\tau \cup \{R\})$-sentence given by Lemma 5. The following conditions are equivalent.

1. $\mathfrak{A} \models \varphi$.
2. $\mathfrak{A} \models_X \varphi$ for some nonempty team X such that $\mathrm{dom}(X) = \{x_1, \ldots, x_k\}$.
3. $\big(\mathfrak{A}, \mathrm{rel}(X)\big) \models \varphi^*$ for some nonempty team X such that $\mathrm{dom}(X) = \{x_1, \ldots, x_k\}$.
4. $(\mathfrak{A}, R) \models \exists R_1 \ldots \exists R_n \big(\exists x_1 \ldots \exists x_k R(x_1, \ldots, x_k) \wedge \psi\big)$ for some $R \subseteq A^k$.
5. $\mathfrak{A} \models \exists R \exists R_1 \ldots \exists R_n \big(\exists x_1 \ldots \exists x_k R(x_1, \ldots, x_k) \wedge \psi\big)$.

The equivalence of 1 and 2 follows from Proposition 3 and the fact that $\mathrm{Fr}(\varphi) = \emptyset$. By Lemma 5, conditions 2 and 3 are equivalent. The equivalence of 3 and 4 follows from the fact that $\varphi^* = \exists R_1 \ldots \exists R_n \psi$. The conditions 4 are 5 clearly equivalent. ◀

▶ **Theorem 7.** *Let A_Q be a $\Sigma^1_1(\mathsf{FOC}^2)$-definable generalized atom. Then* $\textsc{Sat}(\mathsf{FO}^2(A_Q))$ *and* $\textsc{FinSat}(\mathsf{FO}^2(A_Q))$ *are NEXPTIME-complete.*

Proof. Since the translation $\varphi \mapsto \varphi^*$ is computable in polynomial time and (finite) satisfiability of $\Sigma^1_1(\mathsf{FOC}^2)$ can be checked in *NEXPTIME* [23], we conclude that both $\textsc{Sat}(\mathsf{FO}^2(A_Q))$ and $\textsc{FinSat}(\mathsf{FO}^2(A_Q))$ are in *NEXPTIME*. On the other hand, since $\mathsf{FO}^2 \leq \mathsf{FO}^2(A_Q)$ by Proposition 2, and since both $\textsc{Sat}(\mathsf{FO}^2)$ and $\textsc{FinSat}(\mathsf{FO}^2)$ are *NEXPTIME*-hard [9], it follows that both $\textsc{Sat}(\mathsf{FO}^2(A_Q))$ and also $\textsc{FinSat}(\mathsf{FO}^2(A_Q))$ are as well. ◀

The result of Theorem 7 can be directly generalized to concern finite collections $\mathcal{A}$ of generalized atoms. The proof of the following theorem is practically the same as that of Theorem 7.

▶ **Theorem 8.** *Let $\mathcal{A}$ be a finite collection of $\Sigma^1_1(\mathsf{FOC}^2)$-definable generalized atoms. The satisfiability and the finite satisfiability problems of $\mathsf{FO}^2(\mathcal{A})$ are NEXPTIME-complete.*

We shall next make use of Theorem 8 in order to show that the satisfiability and the finite satisfiability problems of two-variable fragments of dependence logic, inclusion logic, exclusion logic and independence logic are *NEXPTIME*-complete. The result for two-variable dependence logic was already established in [18]. Note that when regarded as generalized atoms, each of the dependency notions above correspond to a collection of generalized atoms; for example the atomic formulas $=(x, y)$ and $=(x, y, z)$ refer to two different atoms, one of type (2) and the other of type (3). However, in order to capture the two-variable fragments of of these logics, we only need a finite number of generalized atoms for each logic, as we shall see. We define $\varphi_{const} := \exists^{\leq 1} x R(x)$, $\varphi_{dep} := \forall x \exists^{\leq 1} y R(x, y)$, $\varphi_{inc} := \forall x \forall y \big(R(x, y) \rightarrow$

$S(x,y)\big)$, $\varphi_{exc} := \forall x \forall y \big(R(x,y) \to \neg S(x,y)\big), \varphi_{ind} := \forall x \forall y \big((\exists y R(x,y) \wedge \exists x R(x,y)) \to R(x,y)\big)$.

The formulas φ_{const}, φ_{dep}, φ_{inc}, φ_{exc} and φ_{ind} define the generalized atoms A_{const} of type (1), A_{dep} of type (2), A_{inc} of type $(2,2)$, A_{exc} of type $(2,2)$, and A_{ind} of type (2), respectively.

▶ **Theorem 9.** *The satisfiability and finite satisfiability problems of the two-variable fragments of dependence logic, inclusion logic, exclusion logic, inclusion/exclusion logic, and independence logic are all NEXPTIME-complete.*

Proof. The proof proceeds via polynomial time translations $\mathsf{D}^2 \to \mathsf{FO}^2(\{A_{const}, A_{dep}\})$, $\mathsf{Inc}^2 \to \mathsf{FO}^2(A_{inc})$, $\mathsf{Exc}^2 \to \mathsf{FO}^2(A_{exc})$, $\mathsf{Inc/Exc}^2 \to \mathsf{FO}^2(A_{inc}, A_{exc})$, $\mathsf{Ind}^2 \to \mathsf{FO}^2(\{A_{const}, A_{dep}, A_{ind}\})$ that preserve equivalence. The result then follows from Lemma 8 and the fact that the generalised atoms A_{const}, A_{dep}, A_{exc}, A_{inc}, A_{ind} are all $\Sigma^1_1(\mathsf{FOC}^2)$-definable. For the full proof, see [19]. ◀

4 Undecidability via non-tiling

In this section we introduce structures and methods that we will later employ to prove undecidability of the validity problem of two-variable dependence logic. Curiously, all attempts (by us or known to us) to use the standard (Π^0_1-complete) tiling problem for the undecidability proof have failed; we will instead use the (Σ^0_1-complete) non-tiling problem in our arguments below.

The grid is the structure $\mathfrak{G} = (\mathbb{N}^2, V, H)$, where $V = \{\big((i,j),(i,j{+}1)\big) \in \mathbb{N}^2 \times \mathbb{N}^2 \mid i,j \in \mathbb{N}\}$ and $H = \{\big((i,j),(i+1,j)\big) \in \mathbb{N}^2 \times \mathbb{N}^2 \mid i,j \in \mathbb{N}\}$. A function $t : 4 \longrightarrow \mathbb{N}$ is called a *tile type*. Define the set TILES $:= \{P_t \mid t$ is a tile type$\}$ of unary relation symbols. The unary relation symbols in the set TILES are called *tiles*. The number $t(0)$ is the *top colour*, $t(1)$ the *right colour*, $t(2)$ the *bottom colour*, and $t(3)$ the *left colour* of P_t.

Let T be a finite nonempty set of tiles and V and H binary relation symbols. We say that a structure $\mathfrak{A} = (A, V, H)$ is T-*tilable*, if there exists an expansion of $\mathfrak{A}$ to the vocabulary $\{H, V\} \cup \{ P_t \mid P_t \in T \}$ such that the following conditions hold for all $u, v \in A$.

1. The point u belongs to the extension of exactly one symbol P_t in T.
2. If uHv, $P_t(u)$ and $P_s(v)$, then the right colour of P_t is the same as the left colour of P_s.
3. If uVv, $P_t(u)$ and $P_s(v)$, then the top colour of P_t is the same as the bottom colour of P_s.

We will next define the *tiling problem* and the *non-tiling problem*. Let $\mathcal{F}$ denote the set of finite, nonempty subsets of TILES. We define $\mathcal{T} := \{T \in \mathcal{F} \mid \mathfrak{G}$ is T-tilable$\}$ and $\bar{\mathcal{T}}' := \{T \in \mathcal{F} \mid \mathfrak{G}$ is not T-tilable$\}$. The *tiling problem (non-tiling problem,* resp.) is the membership problem of the set $\mathcal{T}$ ($\bar{\mathcal{T}}'$, resp.) with the input set $\mathcal{F}$.

▶ **Theorem 10** ([2]). *The tiling problem is Π^0_1-complete.*

The *non-tiling problem* is the complement of the tiling problem. Thus the following corollary follows.

▶ **Corollary 11.** *The non-tiling problem is Σ^0_1-complete.*

The proof of the following lemma is straightforward.

▶ **Lemma 12.** *There is a computable function associating each input T to the non-tiling problem with an FO^2-sentence φ_T of the vocabulary $\tau := \{H, V\} \cup T$ such that for every structure $\mathfrak{A}$ of the vocabulary $\{H, V\}$, the structure $\mathfrak{A}$ is not T-tilable iff for every expansion $\mathfrak{A}^*$ of $\mathfrak{A}$ to the vocabulary τ, it holds that $\mathfrak{A}^* \models \varphi_T$.*

▶ **Definition 13.** Let $\tau = \{V, H\}$ be a vocabulary where V and H are binary relation symbols. Let $\mathfrak{A} = (A, V, H)$ be a τ-structure. We say that $\mathfrak{A}$ is *gridlike* if the below conditions hold.

1. The extension of V in $\mathfrak{A}$ is serial (i.e., $\forall x \in A\ \exists y \in A$ s.t. $V(x, y)$).
2. The extension of H in $\mathfrak{A}$ is serial (i.e., $\forall x \in A\ \exists y \in A$ s.t. $H(x, y)$).
3. If $a, b, c, b', c' \in A$ are such that $V(a, b)$, $H(b, c)$, $H(a, b')$, and $V(b', c')$, then $c = c'$.

Note that it follows from the above definition that in gridlike structures, for every point a, there exist points b, c and d such that $H(a, b)$, $V(a, c)$, $V(b, d)$, and $H(c, d)$.

Let τ be the vocabulary of gridlike structures and U, P, Q, C unary relation symbols. We say that a $\tau \cup \{U, P, Q, C\}$-structure $\mathfrak{A}$ is *striped and gridlike* if the τ-reduct of $\mathfrak{A}$ is gridlike, the extensions of P and Q in $\mathfrak{A}$ are *distinct* singleton sets, the extension of U in $\mathfrak{A}$ is the union of the extensions of P and Q, and $\mathfrak{A}$ has the following property (intuitively C creates stripes in $\mathfrak{A}$):

$$\big(H(a, b) \Rightarrow (C(a) \Leftrightarrow C(b))\big) \text{ and } \big(V(a, b) \Rightarrow (C(a) \Leftrightarrow \neg C(b))\big). \tag{2}$$

The following lemma can be now proven by a simple inductive argument.

▶ **Lemma 14.** *If $\mathfrak{A}$ is striped and gridlike, then there exists a homomorphism from the grid into $\mathfrak{A}$.*

▶ **Lemma 15.** *Let T be an input to the non-tiling problem. The grid is non-T-tilable iff (the $\{H, V\}$-reduct of) every striped gridlike structure is non-T-tilable.*

Proof. The direction from left to right follows from Lemma 14 in a straightforward way. The converse holds since the grid is an {H,V}-reduct of a striped gridlike structure. ◀

5 The validity problem of D^2 is undecidable

In this section we give a reduction from the non-tiling problem to the validity problem of D^2.

Let $\tau = \{V, H, C, U, P, Q\}$ be the vocabulary of striped gridlike structures. We will first define a formula $\varphi_{non-grid}$ of D^2 such that $\mathfrak{A}$ is not striped and gridlike iff $\mathfrak{A} \models \varphi_{non-grid}$. We first notice that the first two conditions of Definition 13 are easy to deal with. Define $\varphi_{non-serial} := \exists x \forall y \neg V(x, y) \vee \exists x \forall y \neg H(x, y)$. The third condition of Definition 13 is nontrivial. In the below construction, we will use the predicates P, Q, U for counting (only). We will first show how to force the extensions of P and Q to be distinct singletons and the extension of U to be the union of P and Q. The next formulae will be used for dealing with the cases where this *does not* hold.

$$\begin{aligned}
\varphi_{non-singleton}(X) &:= \forall x \neg X(x) \vee \exists x \exists y \big(X(x) \wedge X(y) \wedge \neg x = y\big)\\
\varphi_{non-distinct}(X, Y) &:= \exists x \big(X(x) \wedge Y(x)\big)\\
\varphi_{non-union}(X, Y, Z) &:= \exists x \Big(X(x) \wedge \big(\neg Y(x) \vee \neg Z(x)\big)\Big) \vee \exists x \Big(\neg X(x) \wedge \big(Y(x) \vee Z(x)\big)\Big)\\
\varphi_{|U| \neq 2} &:= \varphi_{non-singleton}(P) \vee \varphi_{non-singleton}(Q) \vee \varphi_{non-distinct}(P, Q)\\
&\quad\ \vee \varphi_{non-union}(U, P, Q).
\end{aligned}$$

It is easy to check that the τ-models $\mathfrak{A}$ such that $\mathfrak{A} \not\models \varphi_{|U| \neq 2}$ are exactly those models where the extensions of P and Q are distinct singletons and the extension of U is the union of the extensions of P and Q (and thus the cardinality of the extension of U is 2).

We will now show how to enforce Equation (2). The formula $\varphi_{non-stripes}$ below takes care of the cases where (2) does *not* hold. Define

$$\varphi_{non-stripes} := \exists x \exists y \Big(\big(H(x,y) \wedge (C(x) \leftrightarrow \neg C(y)) \big) \vee \big(V(x,y) \wedge (C(x) \leftrightarrow C(y)) \big) \Big).$$

We are now ready to show how to deal with models that violate the last condition of Definition 13. To understand the intended meaning of the following formula, assume that the extension of U is of size two and that the condition given by Equation (2) holds. Note also that from (2) it follows that if such points c and c' exist that violate the last condition of Definition 13, then c and c' agree about C, i.e., we have $C(c)$ iff $C(c')$. We first deal with the case where $C(c)$ and $C(c')$ both hold. We denote by $\varphi_{non-C^{+}-join}$ the following formula (whose meaning is fully explained in the proof of Lemma 16):

$$\forall x \Big(\neg U(x) \vee \exists y \Big(C(y) \wedge {=}(y,x) \wedge \exists x \Big({=}(x,y) \wedge \big(({=}(x) \wedge H(x,y)) \vee ({=}(x) \wedge V(x,y)) \big) \wedge \exists y \big({=}(y) \wedge \big(V(y,x) \vee H(y,x) \big) \wedge \neg C(y) \big) \Big) \Big) \Big).$$

To deal with the case where $\neg C(c)$ and $\neg C(c')$, we define the formula $\varphi_{non-C^{-}-join}$ which is obtained from $\varphi_{non-C^{+}-join}$ by simultaneously replacing each $C(x)$ and $C(y)$ by $\neg C(x)$ and $\neg C(y)$, respectively. Finally, we define that $\varphi_{non-join} := \varphi_{non-C^{+}-join} \vee \varphi_{non-C^{-}-join}$ and $\varphi_{non-grid} := \varphi_{non-serial} \vee \varphi_{|U| \neq 2} \vee \varphi_{non-stripes} \vee \varphi_{non-join}$.

▶ **Lemma 16.** *Let $\tau = \{V, H, C, U, P, Q\}$ be the vocabulary of striped gridlike structures. Let $\mathfrak{A}$ be a τ-structure such that the extension of U is of cardinality 2. Assume the condition (2) holds. Then $\mathfrak{A} \models \varphi_{non-join}$ iff the last condition of Definition 13 fails in $\mathfrak{A}$.*

Proof. From (2) it follows that if such c and c' exist in $\mathfrak{A}$ that violate the last condition of Definition 13, then c and c' agree on C. We will show that

$$\mathfrak{A} \models \varphi_{non-C^{+}-join} \text{ iff the last condition of Definition 13 fails in } \mathfrak{A} \text{ for some } c, c' \text{ s.t. } C(c) \,\&\, C(c'). \quad (3)$$

The analogous argument for $\varphi_{non-C^{-}-join}$ and the case where $\neg C(c)$ and $\neg C(c')$ hold is similar.

Below we denote by $\{(x_1, v_1), ..., (x_k, v_k)\}$ the variable assignment that maps x_i to v_i for each i. Let u, u' be the elements that are in the extension of U in $\mathfrak{A}$. We thus have $\mathfrak{A} \models \varphi_{non-C^{+}-join}$ iff

$$\mathfrak{A} \models_{X_1} \exists y \Big(C(y) \wedge {=}(y,x) \wedge \exists x \Big({=}(x,y) \wedge \big(({=}(x) \wedge H(x,y)) \vee ({=}(x) \wedge V(x,y)) \big) \wedge \exists y \big({=}(y) \wedge \big(V(y,x) \vee H(y,x) \big) \wedge \neg C(y) \big) \Big) \Big),$$

where $X_1 = \{\{(x,u)\}, \{(x,u')\}\}$. Now, recalling that dependence logic has the downwards closure property (cf. proposition 4), we observe that the above holds if and only if there exist *distinct* (distinctness being due to the atom ${=}(y,x)$) points c, c' in the extension of C such that

$$\mathfrak{A} \models_{X_2} \exists x \Big({=}(x,y) \wedge \big(({=}(x) \wedge H(x,y)) \vee ({=}(x) \wedge V(x,y)) \big) \wedge \exists y \big({=}(y) \wedge \big(V(y,x) \vee H(y,x) \big) \wedge \neg C(y) \big) \Big),$$

where $X_2 = \{\{(x,u), (y,c)\}, \{(x,u'), (y,c')\}\}$. The above holds if and only if there exist distinct points b, b' of $\mathfrak{A}$ such that $H(b,c)$ and $V(b',c')$ (or $V(b,c)$ and $H(b',c')$ in which case the argument is analogous) and

$$\mathfrak{A} \models_{X_3} \exists y \big({=}(y) \wedge \big(V(y,x) \vee H(y,x) \big) \wedge \neg C(y) \big),$$

where $X_3 = \{\{(x,b),(y,c)\},\{(x,b'),(y,c')\}\}$. The above holds if and only if there exists a point a in $\mathfrak{A}$ such that $\neg C(a)$, ($V(a,b)$ or $H(a,b)$) and ($V(a,b')$ or $H(a,b')$). Since $C(c)$ and $C(c')$ hold, it follows from the assumption that (2) holds that $C(b)$ and $\neg C(b')$. Now since also $\neg C(a)$ holds, it follows again from (2) that $V(a,b)$ and $H(a,b')$. When all of the above is combined, we obtain (3). The analogous condition where $\neg C(c)$ and $\neg C(c')$ is proved similarly. Since (2) holds for $\mathfrak{A}$, any points c and c' of $\mathfrak{A}$ that violate the last condition of Definition 13, must agree on C. Thus the lemma holds. ◀

The next lemma follows from Lemma 16 together with the observations made above.

▶ **Lemma 17.** *Let $\tau = \{V,H,C,U,P,Q\}$ be the vocabulary of striped gridlike structures and let $\mathfrak{A}$ be a τ-model. Then $\mathfrak{A}$ is striped and gridlike iff $\mathfrak{A} \not\models \varphi_{non-grid}$.*

▶ **Theorem 18.** *The validity problem for D^2 is undecidable (more precisely, Σ^0_1-hard).*

Proof. We give a computable reduction from the non-tiling problem to the validity problem of D^2. Since the former is Σ^0_1-complete (Corollary 11), we obtain Σ^0_1-hardness for the latter.

If T is an input to the non-tiling problem, then φ_T denotes the FO^2-sentence given by Lemma 12 and $\varphi_{non-T-tiling} := (\varphi_{non-grid} \vee \varphi_T)$. Let τ be as defined in Lemma 17. Let $C_{\tau,T}$ denote the class of all $\tau \cup T$-structures and let $\mathcal{C}^{\tau,T}_{s-gridlike}$ be the class of exactly all expansions of striped gridlike structures to the vocabulary $\tau \cup T$.

Let T be an input to the non-tiling problem. We will show that the grid is non-T-tilable iff the D^2-sentence $\varphi_{non-T-tiling}$ is valid. By definition, $\varphi_{non-T-tiling}$ is valid iff $\mathfrak{A} \models \varphi_{non-grid} \vee \varphi_T$ holds for every $\mathfrak{A} \in \mathcal{C}_{\tau,T}$. Since $\varphi_{non-grid}$ and φ_T are sentences, the right-hand side of this equivalence is equivalent to the claim that

$$\forall \mathfrak{A} \in \mathcal{C}_{\tau,T} : \mathfrak{A} \models \varphi_{non-grid} \text{ or } \mathfrak{A} \models \varphi_T. \tag{4}$$

By Lemma 17, $\mathfrak{B}^* \models \varphi_{non-grid}$ holds for every τ-reduct $\mathfrak{B}^*$ of $\mathfrak{B} \in \mathcal{C}_{\tau,T}$ that is not striped and gridlike. Hence for every $\mathfrak{B} \in \mathcal{C}_{\tau,T}$ such that the τ-reduct $\mathfrak{B}^*$ of $\mathfrak{B}$ is not striped and gridlike, it holds that $\mathfrak{B} \models \varphi_{non-grid}$. Thus (4) is equivalent to the claim that

$$\forall \mathfrak{A} \in \mathcal{C}^{\tau,T}_{s-gridlike} : \mathfrak{A} \models \varphi_{non-grid} \text{ or } \mathfrak{A} \models \varphi_T. \tag{5}$$

Now let $\mathfrak{B}$ be an arbitrary striped and gridlike τ-structure. By Lemma 17, $\mathfrak{B} \not\models \varphi_{non-grid}$. Thus $\mathfrak{B}^* \not\models \varphi_{non-grid}$ for every expansion $\mathfrak{B}^*$ of $\mathfrak{B}$ to the vocabulary $\tau \cup T$. From this it follows that (5) is equivalent to the claim that

$$\forall \mathfrak{A} \in \mathcal{C}^{\tau,T}_{s-gridlike} : \mathfrak{A} \models \varphi_T. \tag{6}$$

Thus, by Lemma 12, (6) holds if and only if every striped gridlike structure is non-T-tilable. Finally, from Lemma 15 it follows that this is equivalent to the claim that the grid is non-T-tilable. ◀

6 Satisfiability of $\exists^*\forall^*$-formulas

In this section we consider the complexity of satisfiability for sentences of dependence logic and its variants in the prefix class $\exists^*\forall^*$. For first-order logic, the satisfiability and finite satisfiability problems of the prefix class $\exists^*\forall^*$ are known to be *NEXPTIME*-complete. The results hold for both the case with equality and the case without equality, see [3].

Let $\mathcal{A}$ be a collection of generalized atoms. We denote by $\exists^*\forall^*[\mathcal{A}]$ the class of sentences of $\mathsf{FO}(\mathcal{A})$ of the form $\exists x_0 \cdots \exists x_n \forall y_0 \cdots \forall y_m \theta$, where θ is a quantifier-free formula whose generalized atoms are in $\mathcal{A}$. It is worth noting that, depending on the set $\mathcal{A}$, the expressive power and complexity of sentences in $\exists^*\forall^*[\mathcal{A}]$ can vary considerably even when $\mathcal{A}$ is finite and contains only computationally non-complex atoms. For example, there are universal sentences of dependence logic that define NP-complete problems [17]. Furthermore, every sentence of inclusion logic is equivalent to a sentence with a prefix of the form $\exists^*\forall^1$ [12] implying that the satisfiability problem of the $\exists^*\forall^*$-fragment of inclusion logic is undecidable.

Recall that we say that a formula φ is *closed under substructures* if for all $\mathfrak{A}$ and X it holds that if $\mathfrak{A} \models_X \varphi$, $\mathfrak{A}' := \mathfrak{A} \upharpoonright B$ and $X' := X \upharpoonright B$ for some $B \subseteq A$, then we have $\mathfrak{A}' \models_{X'} \varphi$.

▶ **Lemma 19.** *Let $\mathcal{A}$ be a collection of generalized atoms that are closed under substructures. Then the following conditions hold.*

1. *Suppose $\varphi \in \mathsf{FO}[\mathcal{A}]$ is of the form $\forall y_0 \cdots \forall y_m \theta$, where θ is quantifier-free. Then φ is closed under substructures.*
2. *Let $\varphi \in \exists^*\forall^*[\mathcal{A}]$ be a sentence. Then, if φ is satisfiable, φ has a model with at most $max\{1, k\}$ elements, where k refers to the number of existentially quantified variables in φ.*

Proof. We will first prove claim (1). Suppose that $\varphi := \forall y_0 \cdots \forall y_m \theta$. We will first show the claim for quantifier-free formulas θ, i.e., we will show that for all $\mathfrak{A}$, X, $\mathfrak{A}'$, and X' such that $\mathfrak{A}' := \mathfrak{A} \upharpoonright B$ and $X' := X \upharpoonright B$ for some $B \subseteq A$, the following implication holds.

$$\mathfrak{A} \models_X \theta \Rightarrow \mathfrak{A}' \models_{X'} \theta. \tag{7}$$

The claim obviously holds if θ is a first-order literal. If θ is a generalized atom from $\mathcal{A}$, then the claim holds by assumption. The case $\theta := \psi_1 \wedge \psi_2$ follows immediately from the induction hypothesis. Let us then assume that $\theta := \psi_1 \vee \psi_2$. Since $\mathfrak{A} \models_X \theta$, there are sets Y and Z such that $Y \cup Z = X$, $\mathfrak{A} \models_Y \psi_1$ and $\mathfrak{A} \models_Z \psi_2$. By the induction hypothesis, we have $\mathfrak{A}' \models_{Y'} \psi_1$ and $\mathfrak{A}' \models_{Z'} \psi_2$, where $Y' := Y \upharpoonright B$ and $Z' := Z \upharpoonright B$. Since $Y' \cup Z' = X'$, it follows that $\mathfrak{A}' \models_{X'} \theta$.

We will now show that the claim also holds for φ. Suppose that $\mathfrak{A} \models_X \varphi$. Then, by the truth definition, $\mathfrak{A} \models_{X[A/y_0]\cdots[A/y_m]} \theta$. Using (7), we have $\mathfrak{A}' \models_{(X[A/y_0]\cdots[A/y_m])\upharpoonright B} \theta$. It is easy to check that $(X[A/y_0]\cdots[A/y_m]) \upharpoonright B = (X \upharpoonright B)[B/y_0]\cdots[B/y_m]$. Hence we have $\mathfrak{A}' \models_{X'} \varphi$.

Let us then prove 2. Assume φ is a sentence of the form $\exists x_0 \cdots \exists x_n \forall y_0 \cdots \forall y_m \theta$, where θ is quantifier-free, and that there is a structure $\mathfrak{A}$ such that $\mathfrak{A} \models \varphi$. Hence there exists functions F_i such that $\mathfrak{A} \models_X \forall y_0 \cdots \forall y_m \theta$, where $X = \{\emptyset\}[F_0/x_0]\cdots[F_n/x_n]$. Let s be some assignment in X. Let range(s) denote the set of elements b such that $s(x) = b$ for some variable x in the domain of s. If range$(s) \neq \emptyset$ define $B :=$ range(s), and if range$(s) = \emptyset$ (i.e., $s = \emptyset$), define $B = \{b\}$, where b is an arbitrary element in A. By claim (1), the formula $\forall y_0 \cdots \forall y_m \theta$ is closed under substructures. Thus $\mathfrak{A} \upharpoonright B \models_{X \upharpoonright B} \forall y_0 \cdots \forall y_m \theta$. Thus it follows that $\mathfrak{A} \upharpoonright B \models \varphi$. ◀

A generalized atom A_Q is said to be *polynomial time computable* if the question whether $\mathfrak{A} \models_X A_Q(\overline{y}_1, ..., \overline{y}_n)$ holds can be decided in time polynomial in the size of $\mathfrak{A}$ and X. A class of atoms $\mathcal{A}$ is said to be *uniformly polynomial time computable* if there exists a polynomial function $f : \mathbb{N} \to \mathbb{N}$ such that for every atom $A_Q \in \mathcal{A}$ it holds that the question whether $\mathfrak{A} \models_X A_Q(\overline{y}_1, ..., \overline{y}_n)$ holds can be decided in time $f\big(|\mathfrak{A}| + |X| + |A_Q(\overline{y}_1, ..., \overline{y}_n)|\big)$. Note that every finite class of polynomial time computable atoms is also uniformly polynomial time computable.

The following theorem now follows from Lemma 19. We will make use of the recent result of Grädel showing that for a uniformly polynomial time computable collection $\mathcal{A}$ of atoms, the model checking problem for $\mathsf{FO}(\mathcal{A})$-formulas is in *NEXPTIME* [8].

▶ **Theorem 20.** *Let A_Q be a generalized atom that is closed under substructures and polynomial time computable. Then* $\textsc{Sat}(\exists^*\forall^*[A_Q])$ *and* $\textsc{FinSat}(\exists^*\forall^*[A_Q])$ *are in* 2*NEXPTIME. If τ is a vocabulary consisting of relation symbols of arity at most k, $k \in \mathbb{Z}_+$, then* $\textsc{Sat}(\exists^*\forall^*[A_Q](\tau))$ *and* $\textsc{FinSat}(\exists^*\forall^*[A_Q](\tau))$ *are NEXPTIME-complete.*

Proof. Note first that the lower bounds follow from the fact that both $\textsc{Sat}(\exists^*\forall^*)$ and $\textsc{FinSat}(\exists^*\forall^*)$ are already *NEXPTIME*-complete. It hence suffices to show containments in 2*NEXPTIME* and *NEXPTIME*, respectively.

Let $\varphi \in \exists^*\forall^*[A_Q]$. By Lemma 19, φ is satisfiable if and only if it has a model of cardinality at most $|\varphi|$. We can decide satisfiability of φ as follows: non-deterministically guess a structure $\mathfrak{A}$ of cardinality at most $|\varphi|$ and accept iff $\mathfrak{A} \models \varphi$. By the result of Grädel in [8], the question whether $\mathfrak{A} \models \varphi$ can be checked non-deterministically in exponential time with input $\mathfrak{A}$ and φ. Assume first that the maximum arity of relation symbols that may occur in φ is not a fixed constant. Relation symbols of arity at most $|\varphi|$ may occur in φ. Thus the size of the *binary encoding of a model* $\mathfrak{A}$ of φ such that $A \leq |\varphi|$ is worst case exponential with respect to $|\varphi|$. If, on the other hand, the maximum arity of relation symbols that can occur in φ is a fixed constant, then the size of the encoding of $\mathfrak{A}$ is just worst case polynomial with respect to $|\varphi|$. Therefore it follows that our algorithm for checking satisfiability of φ is in *NEXPTIME* in the case of fixed arity vocabularies and in 2*NEXPTIME* in the general case. The corresponding results for the finite satisfiability problem follow by the observation that $\exists^*\forall^*[A_Q]$ has the finite model property, Lemma 19. ◀

▶ **Corollary 21.** *Let $\mathcal{A}$ be a uniformly polynomial time computable class of generalized atoms that are closed under substructures. Then* $\textsc{Sat}(\exists^*\forall^*[\mathcal{A}])$ *and* $\textsc{FinSat}(\exists^*\forall^*[\mathcal{A}])$ *are in* 2*NEXPTIME. If τ is a vocabulary consisting of relation symbols of arity at most k, $k \in \mathbb{Z}_+$, then* $\textsc{Sat}(\exists^*\forall^*[\mathcal{A}](\tau))$ *and* $\textsc{FinSat}(\exists^*\forall^*[\mathcal{A}](\tau))$ *are NEXPTIME-complete.*

In the following sense Theorem 20 is optimal: there exists a polynomial time computable generalized atom A_Q such that $\textsc{Sat}(\exists^3\forall[A_Q])$ and $\textsc{FinSat}(\exists^3\forall[A_Q])$ are undecidable. This already holds for vocabularies with at least one binary relation symbol and a countably infinite set of unary relation symbols. Let $\varphi_{5-inc} := \forall x_1 \ldots \forall x_5 \big(R(x_1, \ldots, x_5) \rightarrow S(x_1, \ldots x_5)\big)$, and let A_{5-inc} be the related generalized atom of the type $(5, 5)$, i.e., A_{5-inc} is the 5-ary inclusion atom interpreted as a generalized atom. Clearly A_{5-inc} is computable in polynomial time.

▶ **Theorem 22.** *Let τ be a vocabulary consisting of one binary relation symbol and a countably infinite set of unary relation symbols. Then both* $\textsc{Sat}(\exists^3\forall[A_{5-inc}](\tau))$ *and* $\textsc{FinSat}(\exists^3\forall[A_{5-inc}](\tau))$ *are undecidable.*

Proof. It well known that for the Kahr class (i.e., the prefix class $\forall\exists\forall$ of FO with vocabulary τ) the satisfiability and the finite satisfiability problems are undecidable (see, e.g., [3]). From the proof of [12, Theorem 5] it follows that there exists a polynomial time translation $\varphi \mapsto \varphi^*$ from the Kahr class into $\exists^3\forall[A_{5-inc}](\tau)$ such that $\mathfrak{A} \models_X \varphi \Leftrightarrow \mathfrak{A} \models_X \varphi^*$ holds for every model $\mathfrak{A}$ and team X with codomain A. Thus $\textsc{Sat}(\exists^3\forall[A_{5-inc}](\tau))$ and $\textsc{FinSat}(\exists^3\forall[A_{5-inc}](\tau))$ are undecidable. ◀

It is easy to see that dependence atoms viewed as generalized atoms are closed under substructures since they are downwards closed and universe independent. Likewise, it is straightforward to check that the class of dependence atoms is uniformly polynomial time computable. Hence we obtain:

▶ **Corollary 23.** *Both the satisfiability and the finite satisfiability problems for the* $\exists^*\forall^*$*-sentences of dependence logic are in* 2*NEXPTIME. If* τ *is a vocabulary consisting of relation symbols of arity at most* k*, then the satisfiability and the finite satisfiability problems for the* $\exists^*\forall^*$*-sentences of dependence logic over the vocabulary* τ *are NEXPTIME-complete.*

7 Conclusion

We have tied some loose ends concerning the complexity of predicate logics based on team semantics. Using a general approach, we have shown that the satisfiability and the finite satisfiability problems of the two-variable fragments of inclusion logic, exclusion logic, inclusion/exclusion logic, and independence logic are all *NEXPTIME*-complete. Additionally, we have shown that the satisfiability and the finite satisfiability problems of the prefix class $\exists^*\forall^*$ of dependence logic are *NEXPTIME*-complete for any vocabulary of bounded arity, and in 2*NEXPTIME* in the general case. The general approach we have employed of course also implies a range of other results on team-semantics-based logics. Finally, we have proved that the validity problem of two-variable dependence logic is undecidable, thereby answering an open problem from the literature on team semantics.

This article clears path to a more comprehensive classification of the decidability and complexity of different fragments of logics with generalized atoms and team semantics. In the future, we aim to identify further interesting related systems with a decidable satisfiability problem.

Acknowledgements. Juha Kontinen was supported by grant 292767 of the Academy of Finland. Antti Kuusisto was supported by the ERC grant 647289 “CODA” and the Jenny and Antti Wihuri Foundation. Jonni Virtema was supported by grants 266260 and 292767 of the Academy of Finland and the Vilho, Yrjö and Kalle Väisälä Foundation.

References

1 Saguy Benaim, Michael Benedikt, Witold Charatonik, Emanuel Kieroński, Rastislav Lenhardt, Filip Mazowiecki, and James Worrell. Complexity of two-variable logic on finite trees. In *ICALP (2)*, pages 74–88, 2013. doi:10.1007/978-3-642-39212-2_10.

2 R. Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 66:369–395, 1966.

3 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

4 Witold Charatonik and Piotr Witkowski. Two-variable logic with counting and trees. In *LICS*, pages 73–82. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.12.

5 A. Durand, J. Kontinen, and H. Vollmer. *Expressivity and Complexity of Dependence Logic, in Dependence Logic: Theory and Applications*. Springer, In Press, 2016.

6 Pietro Galliani. Inclusion and exclusion dependencies in team semantics – on some logics of imperfect information. *Ann. Pure Appl. Logic*, 163(1):68–84, 2012. doi:10.1016/j.apal.2011.08.005.

7 Pietro Galliani and Lauri Hella. Inclusion logic and fixed point logic. In *proceedings of CSL 2013*, pages 281–295, 2013.

8 Erich Grädel. Model-checking games for logics of imperfect information. *Theor. Comput. Sci.*, 493:2–14, 2013. doi:10.1016/j.tcs.2012.10.033.

9 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *The Bulletin of Symbolic Logic*, 3(1):53–69, 1997. URL: `http://www.jstor.org/stable/421196`.

10 Erich Grädel, Martin Otto, and Eric Rosen. Undecidability results on two-variable logics. In *Proceedings of STACS'97*, pages 249–260, London, UK, 1997. Springer-Verlag.

11 Erich Grädel and Jouko Väänänen. Dependence and independence. *Studia Logica*, 101(2):399–410, 2013. `doi:10.1007/s11225-013-9479-2`.

12 Miika Hannula. Hierarchies in inclusion logic with lax semantics. In Mohua Banerjee and Shankara Narayanan Krishna, editors, *Logic and Its Applications: 6th Indian Conference, ICLA 2015, Mumbai, India, January 8-10, 2015. Proceedings*, pages 100–118, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. `doi:10.1007/978-3-662-45824-2_7`.

13 Lauri Hella and Antti Kuusisto. One-dimensional fragment of first-order logic. In *Advances in Modal Logic 10, invited and contributed papers from the tenth conference on "Advances in Modal Logic," held in Groningen, The Netherlands, August 5-8, 2014*, pages 274–293, 2014. URL: `http://www.aiml.net/volumes/volume10/Hella-Kuusisto.pdf`.

14 Jaakko Hintikka and Gabriel Sandu. Informational independence as a semantical phenomenon. In J. E. Fenstad, I. T. Frolov, and R. Hilpinen, editors, *Logic, Methodology and Philosophy of Science VIII*, volume 126, pages 571–589. Elsevier, Amsterdam, 1989.

15 Wilfrid Hodges. Compositional semantics for a language of imperfect information. *Log. J. IGPL*, 5(4):539–563 (electronic), 1997.

16 Emanuel Kieroński, Jakub Michaliszyn, Ian Pratt-Hartmann, and Lidia Tendera. Two-variable first-order logic with equivalence closure. *SIAM Journal of Computing*, 43(3):1012–1063, 2014.

17 Jarmo Kontinen. Coherence and computational complexity of quantifier-free dependence logic formulas. *Studia Logica*, 101(2):267–291, 2013. `doi:10.1007/s11225-013-9481-8`.

18 Juha Kontinen, Antti Kuusisto, Peter Lohmann, and Jonni Virtema. Complexity of two-variable dependence logic and IF-logic. *Inf. Comput.*, 239:237–253, 2014. `doi:10.1016/j.ic.2014.08.004`.

19 Juha Kontinen, Antti Kuusisto, and Jonni Virtema. Decidability of predicate logics with team semantics. *CoRR*, abs/1410.5037, 2016. URL: `http://arxiv.org/abs/1410.5037`.

20 Antti Kuusisto. A double team semantics for generalized quantifiers. *Journal of Logic, Language and Information*, 24(2):149–191, 2015. `doi:10.1007/s10849-015-9217-4`.

21 Per Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

22 Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *Proceedings of LICS'97*, pages 318–327, 1997. `doi:10.1109/LICS.1997.614958`.

23 Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005. `doi:10.1007/s10849-005-5791-1`.

24 Wieslaw Szwast and Lidia Tendera. FO^2 with one transitive relation is decidable. In *STACS*, pages 317–328, 2013.

25 Jouko Väänänen. *Dependence logic: A new approach to independence friendly logic.* Number 70 in London Mathematical Society student texts. Cambridge University Press, 2007.

26 Jonni Virtema. *Approaches to Finite Variable Dependence: Expressiveness and Computational Complexity.* PhD thesis, University of Tampere, 2014.

On the Complexity of Universality for Partially Ordered NFAs*

Markus Krötzsch[1], **Tomáš Masopust**[2], **and Michaël Thomazo**[3]

1 **Institute of Theoretical Computer Science and Center of Advancing Electronics Dresden (cfaed), TU Dresden, Germany**
markus.kroetzsch@tu-dresden.de

2 **Institute of Theoretical Computer Science and Center of Advancing Electronics Dresden (cfaed), TU Dresden, Germany**
tomas.masopust@tu-dresden.de

3 **Inria, Gif-sur-Yvette, France**
michael.thomazo@inria.fr

Abstract

Partially ordered nondeterminsitic finite automata (poNFAs) are NFAs whose transition relation induces a partial order on states, i.e., for which cycles occur only in the form of self-loops on a single state. A poNFA is universal if it accepts all words over its input alphabet. Deciding universality is PSpace-complete for poNFAs, and we show that this remains true even when restricting to a fixed alphabet. This is nontrivial since standard encodings of alphabet symbols in, e.g., binary can turn self-loops into longer cycles. A lower coNP-complete complexity bound can be obtained if we require that all self-loops in the poNFA are deterministic, in the sense that the symbol read in the loop cannot occur in any other transition from that state. We find that such restricted poNFAs (rpoNFAs) characterise the class of $\mathcal{R}$-trivial languages, and we establish the complexity of deciding if the language of an NFA is $\mathcal{R}$-trivial. Nevertheless, the limitation to fixed alphabets turns out to be essential even in the restricted case: deciding universality of rpoNFAs with unbounded alphabets is PSpace-complete. Our results also prove the complexity of the inclusion and equivalence problems, since universality provides the lower bound, while the upper bound is mostly known or proved in the paper.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases Automata, Nondeterminism, Partial order, Universality

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.61

1 Introduction

The universality problem asks if a given automaton (or grammar) accepts (or generates) all possible words over its alphabet. In typical cases, deciding universality is more difficult than deciding the word problem. For example, universality is undecidable for context-free grammars [3] and PSpace-complete for nondeterministic finite automata (NFAs) [25]. The study of universality (and its complement, emptiness) has a long tradition in formal languages, with many applications across computer science, e.g., in the context of formal knowledge representation and database theory [10, 33, 4]. Recent studies investigate the problem for specific types of automata or grammars, e.g., for prefixes or factors of regular languages [28].

* This work was supported by the German Research Foundation (DFG) within the Collaborative Research Center SFB 912 (HAEC) and in Emmy Noether grant KR 4381/1-1 (DIAMOND).

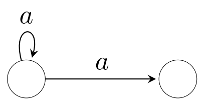


Figure 1 Forbidden pattern of rpoNFAs.

Table 1 Complexity of deciding universality.

	Unary alphabet		Fixed alphabet		Arbitrary alphabet	
DFA	in P		in P		in P	
rpoNFA	in P	(Thm. 4)	coNP-comp.	(Cor. 16)	PSpace-comp.	(Thm. 19)
poNFA	in P	(Thm. 4)	PSpace-comp.	(Thm. 3)	PSpace-comp.	[1]
NFA	coNP-comp.	[34]	PSpace-comp.	[1]	PSpace-comp.	[1]

In this paper, we are interested in the universality problem for *partially ordered NFAs* (poNFAs) and special cases thereof. An NFA is partially ordered if its transition relation induces a partial order on states: the only cycles that are allowed are self-loops on a single state. Partially ordered NFAs define a natural class of languages that has been shown to coincide with level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [31] and with Alphabetical Pattern Constraint (APC) languages, a subclass of regular languages effectively closed under permutation rewriting [6]. Deciding if an automaton recognises an APC language (and hence whether it can be recognised by a poNFA) is PSpace-complete for NFAs and NL-complete for DFAs [6].

Restricting to partially ordered deterministic finite automata (poDFAs), we can capture further classes of interest: two-way poDFAs characterise languages whose syntactic monoid belongs to the variety **DA** [31], introduced by Schützenberger [30]; poDFAs characterise $\mathcal{R}$-trivial languages [9]; and confluent poDFAs characterise level 1 of the Straubing-Thérien hierarchy, also known as $\mathcal{J}$-trivial languages or piecewise testable languages [32]. Other relevant classes of partially ordered automata include partially ordered Büchi automata [20] and two-way poDFAs with look-around [21].

A first result on the complexity of universality for poNFAs is readily obtained. It is well known that universality of regular expressions is PSpace-complete [1, Lemma 10.2], and it is easy to verify that the regular expressions used in the proof can be expressed in poNFAs:

▶ **Corollary 1** (Lemma 10.2 [1]). *The universality problem for poNFAs is* PSpace*-complete.*

A closer look at the proof reveals that the underlying encoding requires an alphabet of size linear in the input: PSpace-hardness is not established for alphabets of bounded size. Usually, one could simply encode alphabet symbols σ by sequences $\sigma_1 \cdots \sigma_n$ of symbols from a smaller alphabet, say $\{0,1\}$. However, doing this requires self-loops $q \xrightarrow{\sigma} q$ to be replaced by nontrivial cycles $q \xrightarrow{\sigma_1} \dots \xrightarrow{\sigma_n} q$, which are not permitted in poNFAs.

We settle this open problem by showing that PSpace-hardness is retained even for binary alphabets.

This negative result leads us to ask if there is a natural subclass of poNFAs for which universality does become simpler. We consider *restricted* poNFAs (rpoNFAs), which require self-loops to be deterministic in the sense that the automaton contains no transition as in Figure 1. Large parts of the former hardness proof hinge on transitions of this form, which, speaking intuitively, allow the automaton to navigate to an arbitrary position in the input (using the loop) and, thereafter, continue checking an arbitrary pattern. Indeed, we find that the universality becomes coNP-complete for rpoNFAs with a fixed alphabet.

However, this reduction of complexity is not preserved for unrestricted alphabets. We use a novel construction of rpoNFAs that characterise certain exponentially long words to show that universality is PSpace-complete even for rpoNFAs if the alphabet may grow polynomially. Our complexity results are summarised in Table 1.

As a by-product, we show that rpoNFAs provide another characterisation of $\mathcal{R}$-trivial languages introduced and studied by Brzozowski and Fich [9], and we establish the complexity of detecting $\mathcal{R}$-triviality and k-$\mathcal{R}$-triviality for rpoNFAs.

The complexity of the inclusion and equivalence problems of regular expressions of several special forms has been investigated by Martens et al. [22]. Some of them are expressible by poNFAs. The results have been established for alphabets of unbounded size. We point out here that our results also apply to the inclusion and equivalence problems. The complexity of universality provides the lower bound. The upper bound for the case of PSpace-complete problems then follows from the complexity for general NFAs, whereas for the coNP-complete problems it is shown in Theorem 15. Hence the results of Table 1 also hold for inclusion and equivalence.

Finally, we mention the relationship to deterministic regular expressions (DRE) [7], which are of interest in schema languages for XML data – Document Type Definition (DTD) and XML Schema Definition (XSD) – since the World Wide Web Consortium standards require that the regular expressions in their specification are deterministic. The important question is then whether a regular expression or an NFA is expressible as a DRE. This problem has been shown to be PSpace-complete [12]. Since the non-DRE-definable language $(a+b)^*b(a+b)$ [7] can be expressed by a poNFA, the problem is nontrivial for poNFAs. Its complexity (PSpace-complete), however, follows from the existing results, namely from the proof given in [5] showing PSpace-hardness of DRE-definability for regular expressions, since the regular expression constructed there can be expressed as a poNFA. On the other hand, all rpoNFA languages are DRE-definable by the automata characterization presented in [7].

Proofs omitted in the text can be found in the corresponding technical report [19].

2 Preliminaries and Definitions

We assume that the reader is familiar with automata theory [1]. The cardinality of a set A is denoted by $|A|$ and the power set of A by 2^A. An *alphabet* Σ is a finite nonempty set. A *word* over Σ is any element of the free monoid Σ^*, the *empty word* is denoted by ε. A *language* over Σ is a subset of Σ^*. For a language L over Σ, let $\overline{L} = \Sigma^* \setminus L$ denote its complement.

A *subword* of w is a word u such that $w = w_1uw_2$, for some words w_1, w_2; u is a *prefix* of w if $w_1 = \varepsilon$ and it is a *suffix* of w if $w_2 = \varepsilon$.

A *nondeterministic finite automaton* (NFA) is a quintuple $\mathcal{A} = (Q, \Sigma, \cdot, I, F)$, where Q is a finite nonempty set of states, Σ is an input alphabet, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states, and $\cdot : Q \times \Sigma \to 2^Q$ is the transition function that can be extended to the domain $2^Q \times \Sigma^*$ by induction. The language *accepted* by $\mathcal{A}$ is the set $L(\mathcal{A}) = \{w \in \Sigma^* \mid I \cdot w \cap F \neq \emptyset\}$. We often omit $\cdot$ and write simply Iw instead. The NFA $\mathcal{A}$ is *complete* if for every state q and every letter a in Σ, the set $q \cdot a$ is nonempty. It is *deterministic* (DFA) if $|I| = 1$ and $|q \cdot a| = 1$ for every state q in Q and every letter a in Σ.

A *path* π from a state q_0 to a state q_n under a word $a_1a_2\cdots a_n$, for some $n \geq 0$, is a sequence of states and input symbols $q_0a_1q_1a_2\ldots q_{n-1}a_nq_n$ such that $q_{i+1} \in q_i \cdot a_{i+1}$, for $i = 0, 1, \ldots, n-1$. Path π is *accepting* if $q_0 \in I$ and $q_n \in F$. A path is *simple* if all the states are pairwise distinct.

A *deterministic Turing machine* (DTM) is a tuple $M = (Q, T, I, \delta, \sqcup, q_o, q_f)$, where Q is the finite state set, T is the tape alphabet, $I \subseteq T$ is the input alphabet, $\sqcup \in T \setminus I$ is the blank symbol, q_o is the initial state, q_f is the accepting state, and δ is the transition function mapping $Q \times T$ to $Q \times T \times \{L, R, S\}$, see the details in [1].

The *universality problem* asks, given an automaton $\mathcal{A}$ over Σ, whether $L(\mathcal{A}) = \Sigma^*$.

3 Partially Ordered NFAs

In this section, we introduce poNFAs, recall their characterisation in terms of the Straubing-Thérien hierarchy, and show that universality remains PSPACE-complete even when restricting to binary alphabets. Merely the case of unary alphabets turns out to be simpler.

▶ **Definition 2.** Let $\mathcal{A}$ be an NFA. A state q is *reachable* from a state p, written $p \leq q$, if there is a word $w \in \Sigma^*$ such that $q \in p \cdot w$. We write $p < q$ if $p \leq q$ and $p \neq q$. $\mathcal{A}$ is a *partially ordered NFA* (*poNFA*) if $\leq$ is a partial order.

The expressive power of poNFAs can be characterised by the *Straubing-Thérien (ST) hierarchy* [35, 37]. For an alphabet Σ, level 0 of this hierarchy is defined as $\mathcal{L}(0) = \{\emptyset, \Sigma^*\}$. For integers $n \geq 0$, the levels $\mathcal{L}(n)$ and $\mathcal{L}(n + \frac{1}{2})$ are as follows:

- $\mathcal{L}(n + \frac{1}{2})$ consists of all finite unions of languages $L_0 a_1 L_1 a_2 \dots a_k L_k$, with $k \geq 0$, $L_0, \dots, L_k \in \mathcal{L}(n)$, and $a_1, \dots, a_k \in \Sigma$;
- $\mathcal{L}(n + 1)$ consists of all finite Boolean combinations of languages from level $\mathcal{L}(n + \frac{1}{2})$.

Note that the levels of the hierarchy contain only *star-free* languages by definition. It is known that the hierarchy does not collapse on any level [8], but the problem of deciding if a language belongs to some level k is largely open for $k > \frac{5}{2}$ [2, 27]. The ST hierarchy further has close relations to the *dot-depth hierarchy* [11, 8, 36] and to complexity theory [38].

Interestingly, the languages recognised by poNFAs are exactly the languages on level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [31]. Since the hierarchy is proper, this means that poNFAs can only recognise a strict subset of star-free regular languages. In spite of this rather low expressive power, the universality problem of poNFAs has the same worst-case complexity as for general NFAs, even when restricting to a fixed alphabet with only a few letters.

▶ **Theorem 3.** *For every alphabet Σ with $|\Sigma| \geq 2$, the universality problem for poNFAs over Σ is* PSPACE*-complete.*

Proof. Membership follows from the fact that universality is in PSPACE for NFAs [14].

To show hardness, we modify the construction of Aho et al. [1] to work on a two-letter alphabet. Consider a polynomial p and a p-space-bounded DTM $M = \langle Q, T, I, \delta, \sqcup, q_o, q_f \rangle$. Without loss of generality, we assume $q_0 \neq q_f$. We define an encoding of runs of M as a word over a given alphabet. For any input $x \in I^*$, we construct, in polynomial time, a regular expression R_x that represents all words that do *not* encode an accepting run of M on x. Therefore, R_x matches all words iff M does not accept x. The claim then follows by showing that R_x can be encoded by a poNFA.

A configuration of M on an input x consists of a current state $q \in Q$, the position $0 \leq \ell \leq p(|x|)$ of the read/write head, and the current tape contents $\theta_0, \dots, \theta_{p(|x|)}$ with $\theta_i \in T$. We represent it by a sequence $\langle \theta_0, \varepsilon \rangle \cdots \langle \theta_{\ell-1}, \varepsilon \rangle \langle \theta_\ell, q \rangle \langle \theta_{\ell+1}, \varepsilon \rangle \cdots \langle \theta_{p(|x|)}, \varepsilon \rangle$ of symbols from $T \times (Q \cup \{\varepsilon\})$. We denote $T \times (Q \cup \{\varepsilon\})$ by Δ. A potential run of M on x is represented by word $\# w_1 \# w_2 \# \cdots \# w_m \#$, where $w_i \in \Delta^{p(|x|)}$ and $\# \notin \Delta$ is a fresh separator symbol. One can construct a regular expression recognising all words over $\Delta \cup \{\#\}$ that do not correctly encode a run of M at all, or that encode a run that is not accepting [1].

We encode symbols of $\Delta \cup \{\#\}$ using a fixed alphabet $\Sigma = \{0, 1\}$. For each $\delta \in \Delta \cup \{\#\}$, let $\hat{\delta}_1 \cdots \hat{\delta}_K \in \{0, 1\}^K$ be a unique binary encoding of length $K = \lceil \log(|\Delta \cup \{\#\}|) \rceil$. We define $\mathit{enc}(\delta)$ to be the binary sequence $001\hat{\delta}_1 1\hat{\delta}_2 1 \cdots \hat{\delta}_K 1$ of length $L = 2K + 3$. We extend enc to words and sets of symbols as usual: $\mathit{enc}(\delta_1 \cdots \delta_m) = \mathit{enc}(\delta_1) \cdots \mathit{enc}(\delta_m)$ and $\mathit{enc}(\Delta') = \{\mathit{enc}(\delta) \mid \delta \in \Delta'\}$. Importantly, any word of the form $\mathit{enc}(\delta_1 \cdots \delta_m)$ contains 00 only at positions that are multiples of L, marking the start of one encoded symbol.

We now construct the regular expression R_x that matches all words of Σ^* that do not represent an accepting computation of M on x. We proceed in four steps: (A) we detect all words that contain words from Σ^* that are not of the form $\mathit{enc}(\delta)$; (B) we detect all words that do not start with the initial configuration; (C) we detect all words that do not encode a valid run since they violate a transition rule; and (D) we detect all words that encode non-accepting runs, or runs that end prematurely.

For (A), note that a word $w \in \Sigma^*$ that is not of the form $\mathit{enc}(v)$ for any word $v \in (\Delta \cup \{\#\})^*$ must either (A.1) start with 1 or 01; (A.2) end with 0; (A.3) contain a word $00\Sigma^{L-2}$ that is not in $\mathit{enc}(\Delta \cup \{\#\})$; (A.4) contain a word from $\mathit{enc}(\Delta \cup \{\#\})\{1, 01\}$; or (A.5) end in a word $00\Sigma^M$ with $M < L - 2$. Using E to abbreviate $\mathit{enc}(\Delta \cup \{\#\})$ and $\bar{E}$ to abbreviate $00\Sigma^{L-2} \setminus E$ (both sets of polynomially many binary sequences), we can express (A.1)–(A.5) in the regular expression

$$(1\Sigma^* + 01\Sigma^*) + (\Sigma^* 0) + \left(\Sigma^* \bar{E} \Sigma^*\right) + (\Sigma^* E (1 + 01) \Sigma^*) + \left(\Sigma^* 00 (\Sigma + \Sigma^2 + \ldots + \Sigma^{L-3})\right) \quad (1)$$

where we use finite sets $\{e_1, \ldots, e_m\}$ to denote regular expressions $(e_1 + \ldots + e_m)$, as usual. All sets in (1) are polynomial in size, so that the overall expression is polynomial. The expression (1) can be captured by a poNFA since the only cycles required arise when translating Σ^*; they can be expressed as self-loops. All other repetitions of the form Σ^i in (1) can be expanded to polynomial-length sequences without cycles.

For (B), we want to detect all words that do not start with the word $w = \mathit{enc}(\#\langle x_1, q_0\rangle \langle x_2, \varepsilon\rangle \cdots \langle x_{|x|}, \varepsilon\rangle \langle \sqcup, \varepsilon\rangle \cdots \langle \sqcup, \varepsilon\rangle \#)$ of length $(p(|x|) + 2)L$. This happens if (B.1) the word is shorter than $(p(|x|) + 2)L$, or (B.2), starting at position jL for $0 \leq j \leq p(|x|) + 1$, there is a word from the polynomial set $\Sigma^L \setminus \{\mathit{enc}(w_j)\}$, which we abbreviate by $\bar{E}_j$. We can capture (B.1) and (B.2) in the regular expression

$$\left(\varepsilon + \Sigma + \Sigma^2 + \ldots + \Sigma^{L(p(|x|)+2)-1}\right) + \sum_{0 \leq j \leq p(|x|)+1} (\Sigma^{jL} \cdot \bar{E}_j \cdot \Sigma^*) \quad (2)$$

The empty expression ε is used for readability; it can easily be expressed in the NFA encoding. As before, it is easy to see that this expression is polynomial and does not require any nontrivial cycles when encoded in an NFA. Note that we ensure that the surrounding # in the initial configuration are present.

For (C), we need to check for incorrect transitions. Consider again the encoding $\# w_1 \# \ldots \# w_m \#$ of a sequence of configurations with a word over $\Delta \cup \{\#\}$, where we can assume that w_1 encodes the initial configuration according to (A) and (B). In an encoding of a valid run, the symbol at any position $j \geq p(|x|) + 2$ is uniquely determined by the symbols at positions $j - p(|x|) - 2$, $j - p(|x|) - 1$, and $j - p(|x|)$, corresponding to the cell and its left and right neighbour in the previous configuration. Given symbols $\delta_\ell, \delta, \delta_r \in \Delta \cup \{\#\}$, we can therefore define $f(\delta_\ell, \delta, \delta_r) \in \Delta \cup \{\#\}$ to be the symbol required in the next configuration. The case where $\delta_\ell = \#$ or $\delta_r = \#$ corresponds to transitions applied at the left and right edge of the tape, respectively; for the case that $\delta = \#$, we define $f(\delta_\ell, \delta, \delta_r) = \#$, ensuring that the separator # is always present in successor configurations as well. We can then check for invalid transitions using the regular expression

$$\sum_{\delta_\ell, \delta, \delta_r \in \Delta \cup \{\#\}} \Sigma^* \cdot \mathit{enc}(\delta_\ell \delta \delta_r) \cdot \Sigma^{L(p(|x|)-1)} \cdot \mathit{enc}(\overline{f}(\delta_\ell, \delta, \delta_r)) \cdot \Sigma^* \quad (3)$$

where $\overline{f}(\delta_\ell, \delta, \delta_r) = \Delta \cup \{\#\} \setminus \{f(\delta_\ell, \delta, \delta_r)\}$. Polynomiality and poNFA-expressibility are again immediate. Note that expression (3) only detects wrong transitions if a (long enough) next configuration exists. The case that the run stops prematurely is covered next.

Finally, for (D) we detect all words that either (D.1) end in a configuration that is incomplete (too short) or (D.2) end in a configuration that is not in the final state q_f. Abbreviating $T \times (Q \setminus \{q_f\})$ as $\bar{E}_f$, and using similar ideas as above, we obtain

$$\Big(\Sigma^* enc(\#)(\Sigma^L + \ldots + \Sigma^{p(|x|)L})\Big) + \Big(\Sigma^* \bar{E}_f(\varepsilon + \Sigma^L + \ldots + \Sigma^{(p(|x|)-1)L})enc(\#)\Big) \qquad (4)$$

and this can again be expressed as a polynomial poNFA.

The expressions (1)–(4) together then detect all non-accepting or wrongly encoded runs of M. In particular, if we start from the correct initial configuration ((2) does not match), then for (3) not to match, all complete future configurations must have exactly one state and be delimited by encodings of $\#$. Expressing the regular expressions as a single poNFA of polynomial size, we have thus reduced the word problem of polynomially space-bounded Turing machines to the universality problem of poNFAs. ◀

Ellul et al. give an example of a regular expression over a 5-letter alphabet such that the shortest non-accepted word is of exponential length, and which can also be encoded as a poNFA [13, Section 5]. Our previous proof shows such an example for an alphabet of two letters, if we use a Turing machine that runs for exponentially many steps before accepting. Note, however, that this property alone would not imply Theorem 3.

Unary Alphabet. Reducing the size of the alphabet to one leads to a reduction in complexity. This is expected, since the universality problem for NFAs over a unary alphabet is merely coNP-complete [34]. For poNFAs, however, the situation is even simpler:

▶ **Theorem 4.** *The universality problem for poNFAs over a unary alphabet is in* P.

Proof. If the language is infinite, then there must be a simple path from an initial state to an accepting state via a state with a self-loop. Let k denote the length of this path, which is bounded by the number of states. Then this path accepts all words of length at least k, that is, all words of the form $a^k a^*$. It remains to check that all words up to length k are also accepted, which can be done in polynomial time. ◀

4 Restricted Partially Ordered NFAs

We now introduce restricted poNFAs, which are distinguished by the forbidden pattern of Figure 1. We relate them to the known class of $\mathcal{R}$-trivial languages, and we establish complexity results for deciding if a language falls into this class.

▶ **Definition 5.** A *restricted partially ordered NFA (rpoNFA)* is a poNFA such that, for every state q and symbol a, if $q \in q \cdot a$ then $q \cdot a = \{q\}$.

We will show below that rpoNFAs characterise $\mathcal{R}$-trivial languages [9]. To introduce this class of languages, we first require some auxiliary definitions. A word $v = a_1 a_2 \cdots a_n$ is a *subsequence* of a word w, denoted $v \preccurlyeq w$, if $w \in \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$. For $k \geq 0$, we write $sub_k(v) = \{u \in \Sigma^* \mid u \preccurlyeq v, |u| \leq k\}$ for the set of all subsequences of v of length up to k. Two words w_1, w_2 are *$\sim_k$-equivalent*, written $w_1 \sim_k w_2$, if $sub_k(w_1) = sub_k(w_2)$. Then $\sim_k$ is a congruence (for $\cdot$) of finite index (i.e., with finitely many equivalence classes) [32]. $\mathcal{R}$-trivial languages are defined by defining a related congruence $\sim_k^{\mathcal{R}}$ that considers subsequences of prefixes:

▶ **Definition 6.** Let $x, y \in \Sigma^*$ and $k \geq 0$. Then $x \sim_k^{\mathcal{R}} y$ if and only if

- for each prefix u of x, there exists a prefix v of y such that $u \sim_k v$, and
- for each prefix v of y, there exists a prefix u of x such that $u \sim_k v$.

A regular language is *k-$\mathcal{R}$-trivial* if it is a union of $\sim_k^{\mathcal{R}}$ classes, and it is *$\mathcal{R}$-trivial* if it is k-$\mathcal{R}$-trivial for some $k \geq 0$.

It is known that $x \sim_k^{\mathcal{R}} y$ implies $x \sim_k y$ and (if $k \geq 1$) $x \sim_{k-1}^{\mathcal{R}} y$ [9]. Therefore, every k-$\mathcal{R}$-trivial language is also $(k+1)$-$\mathcal{R}$-trivial. Moreover, it has been shown that a language L is $\mathcal{R}$-trivial if and only if the minimal DFA recognising L is partially ordered [9]. We can lift this result to characterise the expressive power of rpoNFAs. Namely, it is known that a language is $\mathcal{R}$-trivial if and only if it is a finite union of $\mathcal{R}$-expressions, i.e., expressions of the form $\Sigma_1^* a_1 \Sigma_2^* a_2 \cdots \Sigma_m^* a_m \Sigma_{m+1}^*$, for some $m \geq 0$, where $a_i \notin \Sigma_i$ for $1 \leq i \leq m$. The characterization goes back to Eilenberg and can be found, e.g., in [26]. Thus, we have the following.

▶ **Theorem 7.** *A regular language is $\mathcal{R}$-trivial if and only if it is accepted by an rpoNFA.*

This characterisation in terms of automata with forbidden patterns can be compared to results of Glaßer and Schmitz, who use DFAs with a forbidden pattern to obtain another characterisation of level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [15, 29].

We can further relate the *depth* of rpoNFAs to k-$\mathcal{R}$-trivial languages. Recall that the depth of an atomaton $\mathcal{A}$, denoted $\mathit{depth}(\mathcal{A})$, is the number of input symbols on the longest simple path of $\mathcal{A}$ that starts in an initial state.

▶ **Theorem 8.** *The language recognised by a complete rpoNFA $\mathcal{A}$ is $\mathit{depth}(\mathcal{A})$-$\mathcal{R}$-trivial.*

Similar relationships have been studied for $\mathcal{J}$-trivial languages [18, 23], but we are not aware of any such investigation for $\mathcal{R}$-trivial languages.

Finally, we may ask how difficult it is to decide whether a given NFA $\mathcal{A}$ accepts a language that is $\mathcal{R}$-trivial or k-$\mathcal{R}$-trivial for a specific $k \geq 0$. For most levels of the ST hierarchy, it is not even known if this problem is decidable, and when it is, exact complexity bounds are often missing [27]. The main exception are $\mathcal{J}$-trivial languages – level 1 of the hiearchy – which have recently attracted some attention, motivated by applications in algebra and XML databases [16, 18, 24].

The following result is a special case of a more general result in [17, Theorem 3.1].

▶ **Theorem 9.** *Given an NFA $\mathcal{A}$, it is* PSPACE*-complete to decide if the language accepted by $\mathcal{A}$ is $\mathcal{R}$-trivial.*

To the best of our knowledge, the following complexity results for recognising $(k\text{-})\mathcal{R}$-trivial languages had not been obtained previously.

▶ **Theorem 10.** *Given an NFA $\mathcal{A}$ and $k \geq 0$, it is* PSPACE*-complete to decide if the language accepted by $\mathcal{A}$ is k-$\mathcal{R}$-trivial.*

In both previous theorems, hardness is shown by reduction from the universality problem for NFAs, hence it holds even for binary alphabets [14]. For a unary alphabet, we can obtain the following result.

▶ **Theorem 11.** *Given an NFA $\mathcal{A}$ over a unary alphabet, the problems of deciding if the language accepted by $\mathcal{A}$ is $\mathcal{R}$-trivial, or k-$\mathcal{R}$-trivial for a given $k \geq 0$, respectively, are both* coNP*-complete.*

5 Deciding Universality of rpoNFAs

In this section, we return to the universality problem for the case of rpoNFAs. We first show that we can indeed obtain the hoped-for reduction in complexity when using a fixed alphabet. For the general case, however, we can recover the same PSPACE lower bound as for poNFAs, albeit with a more involved proof. Even for fixed alphabets, we can get a coNP lower bound:

▶ **Lemma 12.** *The universality problem of rpoNFAs is* coNP*-hard even when restricting to alphabets with two letters.*

The proof proceeds by a direct reduction of propositional logic satisfiability to the emptiness of rpoNFAs. For a matching upper bound, we use some results from the literature.

▶ **Lemma 13** ([9]). *Every congruence class of $\sim_k^{\mathcal{R}}$ contains a unique element of minimal length. If $a_1, a_2, \ldots, a_n \in \Sigma$, then $a_1 a_2 \cdots a_n$ is minimal if and only if $sub_k(\varepsilon) \subsetneq sub_k(a_1) \subsetneq sub_k(a_1 a_2) \subsetneq \ldots \subsetneq sub_k(a_1 a_2 \ldots a_n)$.*

The maximal length of such a word has also been studied [24].

▶ **Lemma 14** ([24]). *Let Σ be an alphabet of cardinality $|\Sigma| \geq 1$, and let $k \geq 1$. The length of a longest word, w, such that $sub_k(w) = \{v \in \Sigma^* \mid |v| \leq k\}$, and, for any two distinct prefixes w_1 and w_2 of w, $sub_k(w_1) \neq sub_k(w_2)$, is $\binom{k+|\Sigma|}{k} - 1$. The bound is tight.*

Lemma 13 and 14 provide the main ingredients for showing that, if the size $|\Sigma|$ of the alphabet is bounded, then non-universality is witnessed by a word of polynomial length. Together with Lemma 12, this allows us to establish the following result, which we state in a more general form.

▶ **Theorem 15.** *Let Σ be a fixed alphabet, and let $\mathcal{A}$ and $\mathcal{B}$ be two complete rpoNFAs over Σ. Then the problem whether $L(\mathcal{A}) \subseteq L(\mathcal{B})$ is* coNP*-complete.*

Proof. Hardness follows from Lemma 12. To prove membership, we denote $|\Sigma| = m$. Let $k = \max\{depth(\mathcal{A}), depth(\mathcal{B})\}$; k is bounded by the number of states of $\mathcal{A}$ and $\mathcal{B}$. By Theorem 8, languages $L(\mathcal{A})$ and $L(\mathcal{B})$ are k-$\mathcal{R}$-trivial, which means that they are a finite union of $\sim_k^{\mathcal{R}}$ classes. According to Lemmas 13 and 14, the length of the unique minimal representatives of the $\sim_k^{\mathcal{R}}$ classes is at most $\binom{k+m}{k} - 1 < \frac{(k+m)^m}{m!}$. Since m is a constant, the bound is polynomial in k. Therefore, if the language $L(\mathcal{A})$ is not a subset of $L(\mathcal{B})$, then there exists a polynomial certificate, which can be guessed by a nondeterministic algorithm. ◀

▶ **Corollary 16.** *Let Σ be a fixed alphabet. Then the universality problem for rpoNFAs over Σ is* coNP*-complete.*

Without fixing the alphabet, universality remains PSPACE-hard even for rpoNFAs, but a proof along the lines of Theorem 3 is not straightforward. In essence, rpoNFAs lose the ability to navigate to an arbitrary position within a word for checking some pattern there. Expressions of the form $(\Sigma^* \cdots)$, which we frequently used, e.g., in (1), are therefore excluded. This is problematic since the run of a polynomially space-bounded Turing machine may be of exponential length, and we need to match patterns across the full length of our (equally exponential) encoding of this run. How can we navigate such a long word without using Σ^*? Our answer is to first define an rpoNFA that accepts all words except for a single, exponentially long word. This word will then be used as an rpoNFA-supported "substrate" for our Turing machine encoding, which again follows Theorem 3.

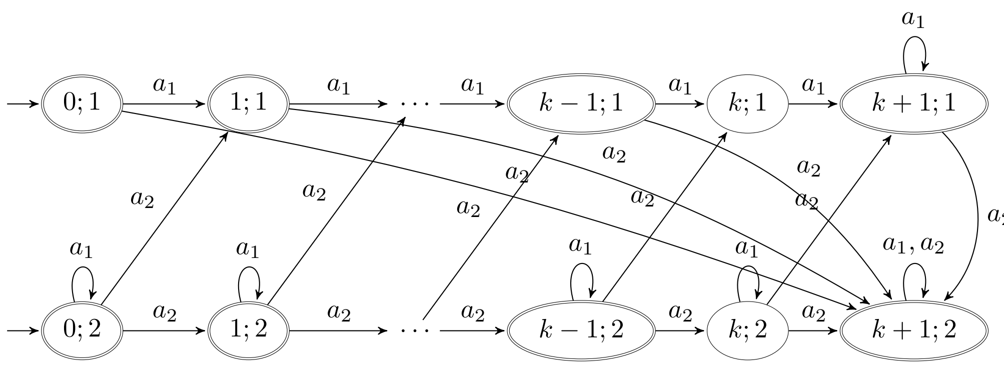


Figure 2 The rpoNFA $\mathcal{A}_{k,2}$ with $2(k+2)$ states.

Table 2 Recursive construction of words $W_{k,n}$ as used in the proof of Lemma 17.

$k\backslash n$	1	2	3
1	a_1	a_1a_2	$a_1a_2a_3$
2	a_1^2	$a_1^2a_2a_1a_2$	$a_1^2a_2a_1a_2a_3a_1a_2a_3$
3	a_1^3	$a_1^3a_2a_1^2a_2a_1a_2$	$a_1^3a_2a_1^2a_2a_1a_2a_3a_1^2a_2a_1a_2a_3a_1a_2a_3$
4	a_1^4	$a_1^4a_2a_1^3a_2a_1^2a_2a_1a_2$	$a_1^4a_2a_1^3a_2a_1^2a_2a_1a_2a_3a_1^3a_2a_1^2a_2a_1a_2a_3a_1^2a_2a_1a_2a_3a_1a_2a_3$

▶ **Lemma 17.** *For all positive integers k and n, there exists an rpoNFA $\mathcal{A}_{k,n}$ over an n-letter alphabet with $n(k+2)$ states such that the unique word not accepted by $\mathcal{A}_{k,n}$ is of length* $\binom{k+n}{k}-1$.

Proof sketch. For integers $k, n \geq 1$, we recursively define words $W_{k,n}$ over the alphabet $\Sigma_n = \{a_1, a_2, \ldots, a_n\}$. For the base cases, we set $W_{k,1} = a_1^k$ and $W_{1,n} = a_1a_2 \ldots a_n$. The cases for $k, n > 1$ are defined recursively by setting

$$\begin{aligned} W_{k,n} &= W_{k,n-1}\, a_n\, W_{k-1,n} \\ &= W_{k,n-1}\, a_n\, W_{k-1,n-1}\, a_n\, W_{k-2,n} \\ &= W_{k,n-1}\, a_n\, W_{k-1,n-1}\, a_n\, \cdots\, a_n\, W_{1,n-1}\, a_n\,. \end{aligned} \tag{5}$$

The recursive construction is illustrated in Table 2. The length of $W_{k,n}$ is $\binom{k+n}{n}-1$ [24]. We further set $W_{k,n} = \varepsilon$ whenever $kn = 0$, since this is useful for defining $\mathcal{A}_{k,n}$ below.

We construct an rpoNFA $\mathcal{A}_{k,n}$ over Σ_n that accepts the language $\Sigma_n^* \setminus \{W_{k,n}\}$. For $n = 1$ and $k \geq 0$, let $\mathcal{A}_{k,1}$ be the minimal DFA accepting the language $\{a_1\}^* \setminus \{a_1^k\}$. It consists of the $k+2$ states of the form $(i;1)$ in the upper part of Figure 2, together with the given transitions. All states but $(k;1)$ are final, and $(0;1)$ is initial.

Given $\mathcal{A}_{k,n-1}$, we recursively construct $\mathcal{A}_{k,n}$ as defined next. The construction for $n = 2$ is illustrated in Figure 2. We obtain $\mathcal{A}_{k,n}$ from $\mathcal{A}_{k,n-1}$ by adding $k+2$ states $(0;n), (1;n), \ldots, (k+1;n)$, where $(0;n)$ is added to the initial states, and all states other than $(k;n)$ are added to the final states. $\mathcal{A}_{k,n}$ therefore has $n(k+2)$ states.

The additional transitions of $\mathcal{A}_{k,n}$ consist of four groups: (1) self-loops $(i;n) \xrightarrow{a_j} (i;n)$ for every $i = 0, \ldots, k+1$ and $a_j = a_1, \ldots, a_{n-1}$. (2) transitions $(i;n) \xrightarrow{a_n} (i+1;n)$ for every $i = 0, \ldots, k$. (3) transitions $(i;n) \xrightarrow{a_n} (i+1;m)$ for every $i = 0, \ldots, k$ and $m = 1, \ldots, n-1$. (4) transitions $(i;m) \xrightarrow{a_n} (k+1;n)$ for every accepting state $(i;m)$ of $\mathcal{A}_{k,n-1}$.

The additional states of $\mathcal{A}_{k,n}$ and transitions (1) and (2) ensure acceptance of every word that does not contain exactly k occurrences of a_n. The transitions (3) ensure acceptance of all words in $(\Sigma_{n-1}^* a_n)^{i+1} L(\mathcal{A}_{k-(i+1),n-1}) a_n \Sigma_n^*$, for which the word between the $(i+1)$st and the $(i+2)$nd occurrence of a_n is not of the form $W_{k-(i+1),n-1}$, hence not a correct subword of $W_{k,n} = W_{k,n-1} a_n \cdots a_n W_{k-(i+1),n-1} a_n \cdots a_n W_{1,n-1} a_n$. The transitions (4) ensure that all words with a prefix $w \cdot a_n$ are accepted, where w is any word $\Sigma_{n-1}^* \setminus \{W_{k,n-1}\}$ accepted by $\mathcal{A}_{k,n-1}$. Together, these conditions ensure that $\mathcal{A}_{k,n}$ accepts every input other than $W_{k,n}$

It remains to show that $\mathcal{A}_{k,n}$ does not accept $W_{k,n}$, which we do by induction on (k,n). We start with the base cases. For $(0,n)$ and any $n \geq 1$, the word $W_{0,n} = \varepsilon$ is not accepted by $\mathcal{A}_{0,n}$, since the initial states $(0,m) = (k,m)$ of $\mathcal{A}_{0,n}$ are not accepting. Likewise, for $(k,1)$ and any $k \geq 0$, we find that $W_{k,1} = a_i^k$ is not accepted by $\mathcal{A}_{k,1}$ (the upper part of Figure 2).

For the inductive case $(k,n) \geq (1,2)$, assume $\mathcal{A}_{k',n'}$ does not accept $W_{k',n'}$ for any $(k',n') < (k,n)$. We have $W_{k,n} = W_{k,n-1} a_n W_{k-1,n}$, and $W_{k,n-1}$ is not accepted by $\mathcal{A}_{k,n-1}$ by induction. In addition, there is no transition under a_n from any non-accepting state of $\mathcal{A}_{k,n-1}$ in $\mathcal{A}_{k,n}$. Therefore, if $W_{k,n}$ is accepted by $\mathcal{A}_{k,n}$, it must be accepted in a run starting from the initial state $(0;n)$. Since $W_{k,n-1}$ does not contain a_n, we find that $\mathcal{A}_{k,n}$ can only reach the states $(0;n) \cdot W_{k,n-1} a_n = \{(1;m) \mid 1 \leq m \leq n\}$ after reading $W_{k,n-1} a_n$. These are the initial states of automaton $\mathcal{A}_{k-1,n}$, which does not accept $W_{k-1,n}$ by induction. Hence $W_{k,n}$ is not accepted by $\mathcal{A}_{k,n}$. ◀

As a corollary, we find that there are rpoNFAs $\mathcal{A} = \mathcal{A}_{n,n}$ for which the shortest non-accepted word is exponential in the size of $\mathcal{A}$. Note that $\binom{2n}{n} \geq 2^n$.

▶ **Corollary 18.** *For every integer $n \geq 1$, there is an rpoNFA $\mathcal{A}_n$ over an n-letter alphabet with $n(n+2)$ states such that the shortest word not accepted by $\mathcal{A}_n$ is of length at least $\binom{2n}{n} - 1$. Therefore, any minimal DFA accepting the same language has at least $\binom{2n}{n}$ states.*

To simulate exponentially long runs of a Turing machine, we start from an encoding of runs using words $\# w_1 \# \ldots \# w_m \#$ as in Theorem 3, but we combine every letter of this encoding with one letter of the alphabet of $\mathcal{A}_n$. We then accept all words for which the projection to the alphabet of $\mathcal{A}_n$ is accepted by $\mathcal{A}_n$, i.e., all but those words of exponential length that are based on the unique word not accepted by $\mathcal{A}_n$. We ensure that, if there is an accepting run, it will have an encoding of this length. It remains to eliminate (accept) all words that correspond to a non-accepting or wrongly encoded run. We can check this as in Theorem 3, restricting to the first components of our combined alphabet. The self-loop that was used to encode Σ^* in poNFAs is replaced by a full copy of $\mathcal{A}_n$, with an additional transition from each state that allows us to leave this "loop." This does not simulate the full loop, but it allows us to navigate the entirety of our exponential word, which is all we need.

▶ **Theorem 19.** *The universality problem for rpoNFAs is* PSPACE*-complete.*

Proof. The membership follows since universality is in PSPACE for NFAs. For hardness, we proceed as explained above. Consider a p-space-bounded DTM $M = \langle Q, T, I, \delta, \sqcup, q_o, q_f \rangle$ as in the proof of Theorem 3. We encode runs of M as words over $T \times (Q \cup \{\varepsilon\}) \cup \{\#\}$ as before. We can use an unrestricted alphabet now, so no binary encoding is needed, and the regular expressions can be simplified accordingly.

If M has an accepting run, then it has one without repeated configurations. For an input word x, there are $C(x) = (|T \times (Q \cup \{\varepsilon\})|)^{p(|x|)}$ distinct configuration words in our encoding. Considering separator symbols $\#$, the maximal length of the encoding of a run without repeated configurations therefore is $1 + C(x)(p(|x|) + 1)$. Let n be the least number such that $|W_{n,n}| \geq 1 + C(x)(p(|x|) + 1)$. Since $|W_{n,n}| + 1 = \binom{2n}{n} \geq 2^n$, it follows that n is smaller than $\lceil \log(1 + C(x)(p(|x|) + 1)) \rceil$ and hence polynomial in the size of M and x.

Consider the automaton $\mathcal{A}_{n,n}$ with alphabet $\Sigma_n = \{a_1, \ldots, a_n\}$ of Lemma 17, and define $\Delta_{\#\$} = T \times (Q \cup \{\varepsilon\}) \cup \{\#, \$\}$. We consider the alphabet $\Pi = \Sigma_n \times \Delta_{\#\$}$, where the second letter is used for encoding a run as in Theorem 3. Since $|W_{n,n}|$ may not be a multiple of $p(|x|) + 1$, we add \$ to fill up any remaining space after the last configuration. For a word $w = \langle a_{i_1}, \delta_1 \rangle \cdots \langle a_{i_\ell}, \delta_\ell \rangle \in \Pi^\ell$, we define $w[1] = a_{i_1} \cdots a_{i_\ell} \in \Sigma_n^\ell$ and $w[2] = \delta_1 \ldots \delta_\ell \in \Delta_{\#\$}^\ell$. Conversely, for a word $v \in \Delta_{\#\$}^*$, we write *enc*$(v)$ to denote the set of all words $w \in \Pi^{|v|}$ with $w[2] = v$. Similarly, for $v \in \Sigma_n^*$, *enc*(v) denotes the words $w \in \Pi^{|v|}$ with $w[1] = v$. We extend this notation to sets of words.

We say that a word w encodes an accepting run of M on x if $w[1] = W_{n,n}$ and $w[2]$ is of the form $\# w_1 \# \cdots \# w_m \# \$^j$ such that there is an $i \in \{1, \ldots, m\}$ for which we have that

- $\# w_1 \# \cdots \# w_i \#$ encodes an accepting run of M on x as in the proof of Theorem 3,
- $w_k = w_i$ for all $k \in \{i+1, \ldots, m\}$, and
- $j \leq p(|x|)$.

In other words, we extend the encoding by repeating the accepting configuration until we have less than $p(|x|) + 1$ symbols before the end of $|W_{n,n}|$ and fill up the remaining places with \$.

The modified encoding requires slightly modified expressions for capturing conditions (A)–(D) from the proof of Theorem 3. Condition (A) is not necessary, since we do not encode symbols in binary. Condition (B) can use the same expression as in (2), adjusted to our alphabet:

$$\left(\varepsilon + \Pi + \Pi^2 + \ldots + \Pi^{p(|x|)+1}\right) + \sum_{0 \leq j \leq p(|x|)+1} (\Pi^j \cdot \bar{E}_j \cdot \Pi^*) \tag{6}$$

where $\bar{E}_j$ is the set $\Sigma_n \times (\Delta_{\#\$} \setminus \{w_j\})$ where w_j encodes the jth symbol on the initial tape as in Theorem 3. All uses of Π^i in this expression encode words of polynomial length, which can be represented in rpoNFAs. Trailing expressions Π^* do not lead to the forbidden pattern of Figure 1.

Condition (C) uses the same ideas as in Theorem 3, especially the transition encoding function f, which we extend to $f : \Delta_{\#\$}^3 \to \Delta_{\#\$}$. For allowing the last configuration to be repeated, we define f as if the final state q_f of M had a self loop (a transition that does not modify the tape, state, or head position). Moreover, we generally permit \$ to occur instead of the expected next configuration symbol. We obtain:

$$\Pi^* \sum_{\delta_\ell, \delta, \delta_r \in \Delta_{\#\$}} \mathit{enc}(\delta_\ell \delta \delta_r) \cdot \Pi^{p(|x|)-1} \cdot \hat{f}(\delta_\ell, \delta, \delta_r) \cdot \Pi^* \tag{7}$$

where $\hat{f}(\delta_\ell, \delta, \delta_r)$ is $\Pi \setminus \mathit{enc}(\{f(\delta_\ell, \delta, \delta_r), \$\})$. Expression (7) is not readily encoded in an rpoNFA, due to the leading Π^*. To address this, we replace Π^* by the expression $\Pi^{\leq |W_{n,n}|-1}$, which matches every word $w \in \Pi^*$ with $|w| \leq |W_{n,n}| - 1$. Clearly, this suffices for our case. As $|W_{n,n}| - 1$ is exponential, we cannot encode this directly as for other expressions Π^i before and we use $\mathcal{A}(n,n)$ instead.

In detail, let E be the expression obtained from (7) when omitting the initial Π^*, and let $\mathcal{A}$ be an rpoNFA that accepts the language of E. We can construct $\mathcal{A}$ so that it has a single initial state. Moreover, let *enc*$(\mathcal{A}_{n,n})$ be the automaton $\mathcal{A}_{n,n}$ of Lemma 17 with each transition $q \xrightarrow{a_i} q'$ replaced by all transitions $q \xrightarrow{\pi} q'$ with $\pi \in \mathit{enc}(a_i)$. We construct an rpoNFA $\mathcal{A}'$ that accepts the language of $(\Pi^* \setminus \{W_{n,n}\}) + (\Pi^{\leq |W_{n,n}|-1} \cdot E)$ by merging *enc*$(\mathcal{A}_{n,n})$ with $n(n+1)$ copies of $\mathcal{A}$, where we identify the initial state of each such copy with a unique final state of *enc*$(\mathcal{A}_{n,n})$. The fact that *enc*$(\mathcal{A}_{n,n})$ alone already accepts $(\Pi^* \setminus \{\mathit{enc}(W_{n,n})\})$

was shown in the proof of Lemma 17. This also implies that it accepts all words of length $\leq |W_{n,n}| - 1$ as needed to show that $(\Pi^{\leq |W_{n,n}|-1} \cdot E)$ is accepted. Entering states of (a copy of) $\mathcal{A}$ after accepting a word of length $\geq |W_{n,n}|$ is possible, but all words accepted in such a way are longer than $W_{n,n}$ and hence in $(\Pi^* \setminus \{enc(W_{n,n})\})$.

Note that the acceptance of $(\Pi^* \setminus \{enc(W_{n,n})\})$, which is a side effect of this encoding, does not relate to expressing (7) but is still useful for our intended overall encoding.

The final condition (D) is minimally modified to allow for up to $p(|x|)$ trailing \$. For a word v, we use $v^{\leq i}$ to abbreviate $(\varepsilon + v + \ldots + v^i)$, and we define $\bar{E}_f = (T \times (Q \setminus \{q_f\}))$ as before. Since (C) does not accept words with too many trailing \$, we add this here instead. Moreover, we need to check that all the symbols \$ appear only at the end, that is, the last expression accepts all inputs where \$ is followed by a different symbol.

$$\begin{aligned}
&\Pi^* enc(\#)(\Pi + \ldots + \Pi^{p(|x|)})enc(\$)^{\leq p(|x|)} + \\
&\Pi^* enc(\bar{E}_f)(\varepsilon + \Pi + \ldots + \Pi^{p(|x|)-1})enc(\#)enc(\$)^{\leq p(|x|)} + \\
&\Pi^* enc(\$)^{p(|x|)+1} + \\
&(\Pi \setminus enc(\$))^* enc(\$)enc(\$)^*(\Pi \setminus enc(\$))\Pi^*
\end{aligned} \tag{8}$$

As before, we cannot encode the leading Π^* directly as an rpoNFA, but we can perform a similar construction as in (7) to overcome this problem.

The union of the rpoNFAs for (6)–(8) constitutes an rpoNFA that is polynomial in the size of M and x, and that is universal if and only if M does not accept x. ◀

6 Conclusion

Our results regarding the complexity of deciding universality for partially ordered NFAs are summarised in Table 1. We found that poNFAs over a fixed, two-letter alphabet are still powerful enough to recognise the language of all non-accepting computations of a PSPACE Turing machine. Restricting poNFAs further by forbidding the pattern of Figure 1, we could establish lower coNP complexity bounds for universality for alphabets of bounded size. We can view this as the complexity of universality of rpoNFAs in terms of the size of the automaton when keeping the alphabet fixed. Unfortunately, the complexity is PSPACE-complete even for rpoNFAs over arbitrary (unbounded) alphabets. The proof uses an interesting construction where the encoding of a Turing machine computation is "piggybacked" on an exponentially long word, for which a dedicated rpoNFA is constructed.

We have characterised the expressive power of rpoNFAs by relating them to the class of $\mathcal{R}$-trivial languages. It is worth noting that the complexity bounds we establish for recognising $\mathcal{R}$-triviality for a given NFA agrees with the complexity of the rpoNFA universality problem for both fixed and arbitrary alphabets. Our results on universality therefore extend beyond rpoNFAs to arbitrary NFAs that recognise $\mathcal{R}$-trivial languages.

Moreover, the results on universality further extend to the complexity of inclusion and equivalence, as explained in the introduction.

Our work can be considered as a contribution to the wider field of studying subclasses of star-free regular languages. The Straubing-Thérien hierarchy provides a large field for interesting future work in this area.

Acknowledgements. We would like to thank Wim Martens for pointing out his paper [22] to our attention and to an anonymous reviewer for pointing out paper [17] and its consequence to Theorem 9.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

2 Jorge Almeida, Jana Bartoňová, Ondřej Klíma, and Michal Kunc. On decidability of intermediate levels of concatenation hierarchies. In *Developments in Language Theory*, volume 9168 of *LNCS*, pages 58–70. Springer, 2015.

3 Yehoshua Bar-Hillel, Micha A. Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 14:143–172, 1961.

4 Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying regular graph patterns. *Journal of the ACM*, 61(1):8:1–8:54, 2014.

5 Geert Jan Bex, Wouter Gelade, Wim Martens, and Frank Neven. Simplifying XML schema: Effortless handling of nondeterministic regular expressions. In *ACM SIGMOD International Conference on Management of Data*, pages 731–744. ACM, 2009.

6 Ahmed Bouajjani, Anca Muscholl, and Tayssir Touilim. Permutation rewriting and algorithmic verification. *Information and Computation*, 205(2):199–224, 2007.

7 Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

8 Janus A. Brzozowski and Robert Knast. The dot-depth hierarchy of star-free languages is infinite. *Journal of Computer and System Sciences*, 16(1):37–55, 1978.

9 Janusz A. Brzozowski and Faith E. Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980.

10 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Reasoning on regular path queries. *SIGMOD Record*, 32(4):83–92, 2003.

11 Rina S. Cohen and Janusz A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.

12 Wojciech Czerwinski, Claire David, Katja Losemann, and Wim Martens. Deciding definability by deterministic regular expressions. In *International Conference on Foundations of Software Science and Computation Structures*, volume 7794 of *LNCS*, pages 289–304. Springer, 2013.

13 Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-Wei Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.

14 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

15 Christian Glaßer and Heinz Schmitz. Languages of dot-depth 3/2. *Theory of Computing Systems*, 42(2):256–286, 2008.

16 Piotr Hofman and Wim Martens. Separability by short subsequences and subwords. In *International Conference on Database Theory*, volume 31 of *LIPIcs*, pages 230–246, 2015.

17 Harry B. Hunt III and Daniel J. Rosenkrantz. Computational parallels between the regular and context-free languages. *SIAM Journal on Computing*, 7(1):99–114, 1978.

18 Ondřej Klíma and Libor Polák. Alternative automata characterization of piecewise testable languages. In *Developments in Language Theory*, volume 7907 of *LNCS*, pages 289–300. Springer, 2013.

19 Markus Krötzsch, Tomáš Masopust, and Michaël Thomazo. On the complexity of universality for partially ordered NFAs. Technical report. URL: `https://ddll.inf.tu-dresden.de/web/Inproceedings3086`.

20 Manfred Kufleitner and Alexander Lauser. Partially ordered two-way Büchi automata. *International Journal of Foundations of Computer Science*, 22(8):1861–1876, 2011.

21 Kamal Lodaya, Paritosh K. Pandya, and Simoni S. Shah. Around dot depth two. In *Developments in Language Theory*, volume 6224 of *LNCS*, pages 303–315. Springer, 2010.

22 Wim Martens, Frank Neven, and Thomas Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM Journal on Computing*, 39(4):1486–1530, 2009.

23 Tomáš Masopust. Piecewise testable languages and nondeterministic automata. In *Mathematical Foundations of Computer Science*, volume 58 of *LIPIcs*, pages 68:1–68:14, 2016.

24 Tomáš Masopust and Michaël Thomazo. On the complexity of k-piecewise testability and the depth of automata. In *Developments in Language Theory*, volume 9168 of *LNCS*, pages 364–376. Springer, 2015.

25 Alfred R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Symposium on Switching and Automata Theory (SWAT/FOCS)*, pages 125–129. IEEE Computer Society, 1972.

26 Jean-Éric Pin. *Varieties Of Formal Languages*. Plenum Press, New York, 1986.

27 Thomas Place and Marc Zeitoun. Separation and the successor relation. In *Symposium on Theoretical Aspects of Computer Science*, volume 30 of *LIPIcs*, pages 662–675, 2015.

28 Narad Rampersad, Jeffrey Shallit, and Zhi Xu. The computational complexity of universality problems for prefixes, suffixes, factors, and subwords of regular languages. *Fundamenta Informatica*, 116(1-4):223–236, 2012.

29 Heinz Schmitz. *The forbidden pattern approach to concatenation hierachies*. PhD thesis, University of Würzburg, 2000.

30 Marcel P. Schützenberger. Sur le produit de concatenation non ambigu. *Semigroup Forum*, 13(1):47–75, 1976.

31 Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *Developments in Language Theory*, volume 2295 of *LNCS*, pages 239–250. Springer, 2001.

32 Imre Simon. *Hierarchies of Events with Dot-Depth One*. PhD thesis, Department of Applied Analysis and Computer Science, University of Waterloo, Canada, 1972.

33 Giorgio Stefanoni, Boris Motik, Markus Krötzsch, and Sebastian Rudolph. The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. *Journal of Artificial Intelligence Research*, 51:645–705, 2014.

34 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *ACM Symposium on the Theory of Computing*, pages 1–9. ACM, 1973.

35 Howard Straubing. A generalization of the Schützenberger product of finite monoids. *Theoretical Computer Science*, 13:137–150, 1981.

36 Howard Straubing. Finite semigroup varieties of the form $\mathbf{V}*\mathbf{D}$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.

37 Denis Thérien. Classification of finite monoids: The language approach. *Theoretical Computer Science*, 14:195–208, 1981.

38 Klaus W. Wagner. Leaf language classes. In *Machines, Computations, and Universality*, volume 3354 of *LNCS*, pages 60–81. Springer, 2004.

Eulerian Paths with Regular Constraints

Orna Kupferman[*1] **and Gal Vardi**[2]

1 School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel

2 School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel

Abstract

Labeled graphs, in which edges are labeled by letters from some alphabet Σ, are extensively used to model many types of relations associated with actions, costs, owners, or other properties. Each path in a labeled graph induces a word in Σ^* – the one obtained by concatenating the letters along the edges in the path. Classical graph-theory problems give rise to new problems that take these words into account. We introduce and study the *constrained Eulerian path* problem. The input to the problem is a Σ-labeled graph G and a specification $L \subseteq \Sigma^*$. The goal is to find an Eulerian path in G that satisfies L. We consider several classes of the problem, defined by the classes of G and L. We focus on the case L is regular and show that while the problem is in general NP-complete, even for very simple graphs and specifications, there are classes that can be solved efficiently. Our results extend work on Eulerian paths with edge-order constraints. We also study the *constrained Chinese postman* problem, where edges have costs and the goal is to find a cheapest path that contains each edge at least once and satisfies the specification. Finally, we define and study the *Eulerian language* of a graph, namely the set of words along its Eulerian paths.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Eulerian Paths, Regular Languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.62

1 Introduction

Many practical problems can be reduced to problems about graphs. A graph consists of vertices, which model objects, and edges, which model pairwise relations between the objects. Different settings call for different types of graphs. For example, when the relation between the objects is not symmetric, the graph is *directed*, and when it is not binary, the graph may have parallel edges or be *weighted*, say for modeling lengths or costs. In many applications, the edges of the graph carry information beyond weight. For example, edges may be associated with an action (say, in VLSI design), a query (say, in databases), properties like their owner or their security level (say, in a network of channels), and many more. Such applications require *labeled graphs*, in which each edge is labeled by a letter from some alphabet.[1]

Each path in a Σ-labeled graph induces a word in Σ^* – the one obtained by concatenating the letters along the edges in the path. Classical graph-theory problems give rise to new

[*] The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 278410, and from The Israel Science Foundation (grant no 1229/10).

[1] Alternatively, one could consider graphs with labels on vertices. It is not hard to alter our results to apply also for this setting.

problems that take these words into account. For example, rather than finding any *shortest path* between two given vertices in a graph [10], it is sometimes desirable, say in transportation planning [5], web searching [1], or network routing, to restrict attention to paths that satisfy some constraint [4]. The basic query mechanism in these applications retrieves all pairs of nodes connected by a path conforming to a given pattern. There have been plenty of theoretical and practical work on the subject of *regular path queries*, where we wish to find all objects reachable by paths whose labels form a word in a given regular language over the alphabet of the labels [8]. As another example, rather than finding a *maximal flow* along arbitrary routes in a graph [13], one may want to restrict the used routes to ones that satisfy some specification [23]. The specification may restrict the length of the routes, preventing long routes from consuming the system, it may restrict the number of different resources applied in a route, require an application of specific resources, require a specific event to trigger another specific event, and so on. As a third example, an extension of network formation games [2] assumes that the edges in the network are labeled and allows to lift the reachability objectives of the players to objectives that are arbitrary regular languages [3]. Paths constrained by regular languages were also considered in the context of finding efficient algorithms for processing database queries ([24, 1]). Finally, online algorithms for finding paths that satisfy regular constraints are given in [7].

An interesting question is how enriching the setting with labels and constrains influences the complexity of classical problems. Note that now there are two parameters to the problem: the graph itself, as well as a formal language $L \subseteq \Sigma^*$, which is usually regular and is given by means of a finite automaton or a regular expression. We refer to L as a *specification.* Existing work shows that the picture is diverse. For example, in the case of shortest paths, finding a shortest path that satisfies regular and even context-free specifications can still be done in polynomial time [4]. Their algorithm is generalized in [6] for graphs with both negative and positive edge weights (but without negative-weight cycles). However, research in regular path queries shows that the problem of finding a shortest simple path that satisfies a regular specification is NP-complete (note that a shortest path that satisfies a specification need not be simple, even when all weights are positive) [24]. In the context of maximal flow, it is shown in [23] that even simple regular restrictions on the routes make the problem APX-hard, namely it is even hard to approximate. Likewise, adding regular objectives to network formation games result in games that are much less stable: they need not have a Nash Equilibrium, and their Price of Stability is higher than in the case of reachability objective [3].

An *Eulerian path* in a graph is a path that traverses all the edges of the graph, each edge exactly once. The problem of deciding whether a given graph has an Eulerian path (the EP problem, for short) was introduced in 1736, in what is considered the first paper in the history of graph theory. The problem can be solved in linear time by examining the parity of the degree of the vertices. Back in 1736, the motivation to study the problem was the challenge of traversing the seven bridges of Königsberg. Nowadays, the problem and its many variants have applications in planning [21], coding [9], synchronization [19], DNA sequencing [27], and many more. In many of these applications, it is useful to restrict attention to Eulerian paths that satisfy some constraint. For example, [21] studies Eulerian paths that satisfy precedence constraints on the edges, specified by linear orders on subsets of the edges. In [27], the input to the EP problem contains a set of paths, and the goal is to find an Eulerian path that contains all the paths in the set as sub-paths. Another related problem is studied in [17]: each edge e in a graph with m edges is associated with an interval $I_e = [l_e, h_e]$ inside $[1, m]$, and the goal is to find an Eulerian path so that the position of every edge e in the path belongs to I_e.

Once we move to consider labeled graphs, the type of restrictions can be much richer. Eulerian paths in labeled graphs were considered in [25], where the problem of finding an Eulerian cycle with a lexicographically minimal label is shown to be NP-complete. Note that the constraint in [25] is not given by means of a specification $L \subseteq \Sigma^*$. Rather, the letters in Σ are ordered and the constraint refers to the lexicographic order between the Eulerian cycles, possibly with respect to a given word. In this work we study Eulerian paths with regular constraints. Formally, the *constrained Eulerian path* problem (CEP problem, for short) is defined as follows: given a Σ-labeled graph G and a regular language $L \subseteq \Sigma^*$, find an Eulerian path in G that satisfies L. We consider several classes of the problem, according to the classes of G and L. We first show that the general CEP problem is NP-complete, and that it is NP-hard already for very restricted graphs and specifications: when the graph does not have parallel edges or loops, and when the specification is a regular expression of a fixed size that can be expressed by a two-state deterministic automaton. In fact, the problem stays NP-hard even when the specification L is a singleton (that is, requiring the Eulerian path to be labeled with a specific given word). We then describe classes of regular languages for which the CEP problem can be solved in polynomial time. For this, we relate the CEP problem with the problem of finding edge-disjoint paths in a graph. In particular, we show that the CEP problem can be solved in polynomial time for regular expressions of the form $R = b_1 \ldots b_k$, where k is fixed, for every $1 \leq i \leq k$, we have $b_i = \sigma_i^*$ or $b_i = \sigma_i$ for some $\sigma_i \in \Sigma$, and for every $\sigma \in \Sigma$, the expression σ^* appears at most three times in R. Alternatively, $b_i = w_i^*$ or $b_i = w_i$ for some $w_i \in \Sigma^*$, and every $\sigma \in \Sigma$ appears at most once in R. We demonstrate the usefulness of such expressions in specifying desired behaviors of paths. We also consider multi-labeled graphs, where an edge may be labeled by several letters, and show that then, the problem is NP-hard even for specifications given by a regular expression of the form a^*b^*, which essentially partitions the path into two types of labels.

An optimization variant or the EP problem is the *Chinese postman problem*. There, each edge in the graph has a non-negative cost, and the goal is to find a *postman path* – one that contains each edge in the graph at least once, of a minimal cost. Clearly, when the graph has an Eulerian path, it induces an optimal postman path. Otherwise, the goal is to get as close as possible to an Eulerian path. Researchers have studied useful restrictions on the allowed postman paths [12]. In particular, a natural extension of the precedence restrictions studied for the EP problem is the *hierarchical Chinese postman problem* [11, 16, 22]. Here, the edges of the graph are partitioned into clusters $E_1, \ldots, E_k$, and a precedence relation $\prec$ specifies the order in which the clusters should be traversed. That is, we seek the cheapest path that visits each edge at least once and so that if $E_i \prec E_j$, then all the edges in E_i are visited for the first time before an edge in E_j is visited. The problem is NP-hard in general, but can be solved in polynomial time in some cases. We consider labeled graphs and study the *constrained Chinese postman* problem (the CCP problem, for short), where the postman path should satisfy a regular specification. We study the complexity of the CCP problem, show that it is in general NP-complete, but point to useful polynomial cases.

A labeled graph G can be viewed as a generator of formal languages. The traditional way to do this is to designate some of the vertices of G as initial and as final vertices. The language of the obtained *automaton* is then the set of words that label paths from some initial to some final vertex. We introduce and study the *Eulerian language* of G, denoted $EL(G)$, namely the set of words that label Eulerian paths in G. Clearly, deciding whether $EL(G) \neq \emptyset$ amount to deciding whether G has an Eulerian path, and can be done in linear time. More interesting questions about the Eulerian language of G relate it to nontrivial languages: whether $EL(G)$ is contained in some specification L, whether some set L of desired behaviors is contained in $EL(G)$, and the relation between the Eulerian languages of two different graphs. Since $EL(G)$ contains only words of a fixed length, we know that $EL(G)$ is finite and hence regular. On the other hand, given a regular language $L \subseteq \Sigma^*$ it is not clear whether there is a graph G such that $EL(G) = L$. We study all the above problems and show that they belong to different levels of the polynomial hierarchy.

Due to lack of space, detailed proofs can be found in the full version, in the authors' URLs.

2 Preliminaries

2.1 Graphs and Eulerian paths

A *graph* $G = \langle V, E\rangle$ consists of a set V of vertices and a set E of directed edges. The graph G may contain loops and parallel edges.[2] A graph is *simple* if it does not contain loops or parallel edges. A *path* P in G is a sequence of edges $e_1, \ldots, e_k$ such that there are $k+1$ vertices $v_0, \ldots, v_k$ and $e_i = (v_{i-1}, v_i)$ for all $1 \le i \le k$. We say that P is a path of length k from v_0 to v_k. If $v_0 = v_k$, then P is a *cycle*. We sometimes refer to P as a sequence of vertices, referring to the vertices $v_0, \ldots, v_k$. For an alphabet Σ, a Σ-*labeled graph* is a tuple $G = \langle V, E, l\rangle$, where $\langle V, E\rangle$ is a graph and $l : E \to \Sigma$ maps each edge to a letter in Σ. The label of a path $P = e_1, \ldots, e_k$, denoted $l(P)$, is the word $l(e_1) \cdot \ldots \cdot l(e_k)$ obtained by concatenating the labels of the edges along P. A *specification* for G is a language $L \subseteq \Sigma^*$. We say that P *satisfies* a specification L if $l(P) \in L$.

Consider a graph $G = \langle V, E\rangle$. For a vertex $v \in V$, the *in degree* and *out degree* of v, denoted *indeg*(v) and *outdeg*(v), are the number of edges entering and leaving v, respectively. An edge $(u_1, u_2) \in E$ is *incident to* v if $u_1 = v$ or $u_2 = v$. We say that G is *strongly connected* if for every two vertices $u, v \in V$ there is a path from u to v. An *undirected path* in G is a sequence of edges $e_1, \ldots, e_k$ such that there are $k+1$ vertices $v_0, \ldots, v_k$ and $e_i \in \{(v_{i-1}, v_i), (v_i, v_{i-1})\}$ for all $1 \le i \le k$. We say that G is *connected* if for every $u, v \in V$ there is an undirected path from u to v. The *size* of G, denoted $|G|$, is the number of vertices and edges in G.

A path in a graph $G = \langle V, E\rangle$ is *Eulerian* (EP, for short) if it visits every edge in E exactly once. We say that G is *Eulerian* if it has an Eulerian cycle. For two edges e_1 and e_2 in E, an EP with $e_1 \prec e_2$ is an EP in which the edge e_1 is visited before the edge e_2. The following is well known.

▶ **Theorem 1.** *Consider a directed graph $G = \langle V, E\rangle$.*

1. *The graph G is Eulerian iff G is connected and for every $v \in V$, we have indeg(v) = outdeg(v).*
2. *The graph G has an Eulerian path from s to t (with $s \neq t$) iff G is connected, outdeg(s) = indeg$(s) + 1$, indeg(t) = outdeg$(t) + 1$, and for every $v \notin \{s, t\}$, we have indeg(v) = outdeg(v).*

By Theorem 1, one can decide in time linear in $|G|$ whether G is Eulerian or has an EP between two given vertices. Furthermore, if G has an EP from s to t, then such path can be found as follows: Follow some path from s until reaching t. It is not possible to get stuck at any vertex other than t, because when the path enters another vertex v there must be an

[2] Since G may contain parallel edges, we do not refer to E as a subset of $V \times V$. However, for simplicity of notations, whenever there is no cause for confusion, we still denote an edge by a pair $(v_1, v_2) \in V \times V$.

unused edge leaving v. The path formed in this way may not cover all edges of the initial graph. As long as there exists a vertex v that belongs to the current path but that has incident edges not part of the path, start another path from v, following unused edges until returning to v (as above, this process does not get stuck), and join the path formed in this way to the previous path.

A path (cycle) in a graph $G = \langle V, E\rangle$ is *Hamiltonian* if it visits every vertex in V exactly once. The Hamiltonian path (cycle) problem, namely deciding whether a given graph has a Hamiltonian path (cycle), is NP-hard already when the graph is simple [15].

2.2 Regular languages and the constrained Eulerian path problem

A *nondeterministic finite automaton* (NFA, for short) is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, F\rangle$, where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, and $F \subseteq Q$ is a set of final states. Given a word $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_l \in \Sigma^*$, a *run* of $\mathcal{A}$ on w is a sequence $r = q_0, q_1, \ldots, q_l$ of states such that $q_0 \in Q_0$ and $q_{i+1} \in \delta(q_i, \sigma_{i+1})$ for all $i \geq 0$. The run is accepting if $q_l \in F$. The NFA $\mathcal{A}$ *accepts* the word w iff it has an accepting run on it. The language of $\mathcal{A}$, denoted $L(\mathcal{A})$ is the set of words that $\mathcal{A}$ accepts. If $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all $q \in Q$ and $\sigma \in \Sigma$, then $\mathcal{A}$ is *deterministic*. Note that a deterministic finite automaton (DFA) has at most one run on each word.

A *regular expression* (RE, for short) over Σ is defined as follows.

- $\emptyset$, ϵ, and σ, for $\sigma \in \Sigma$, are REs.
- If R_1 and R_2 are REs, then so are $R_1 + R_2$, $R_1 \cdot R_2$, and R_1^*.

The language of an RE R, denoted $L(R)$, is defined inductively on the structure of R, where $+$, $\cdot$, and $*$ stand for union, concatenation, and Kleene star, respectively.

We say that R is a *chain* RE if it is of the form $b_1 \cdot b_2 \cdots b_k$, where for every $1 \leq i \leq k$, we have $b_i = w_i^*$ or $b_i = w_i$ for a word $w_i \in \Sigma^*$. We call every such b_i a *block*. Note that a chain RE does not contain the symbol “+" and does not contain nested Kleene stars. For $l \geq 1$, we say that a chain RE R is *l-wide* if $|w_i| \leq l$ for all $1 \leq i \leq k$ with $b_i = w_i^*$. Note that when R is 1-wide, then for all $1 \leq i \leq k$, we have that $b_i = \sigma^*$ or $b_i = \sigma$, for some $\sigma \in \Sigma$. For $d \geq 1$, we say that R is *d-diverse* if each letter in Σ appears in R at most d times. We say that R is *d-star-diverse* if for each letter $\sigma \in \Sigma$, the expression σ^* appears in R at most d times. Note that if R is d-diverse then it is also d-star-diverse. For example, $a^*(ba)^*c^*(bc)^*$ is a 2-wide, 2-diverse, 1-star-diverse chain RE. Then, $a^*b^*a^*$ is a 1-wide, 2-diverse, 2-star-diverse chain RE, and $(a + b)^*(b + c)^*$ is not a chain RE.

The *constrained Eulerian path* problem (CEP problem, for short) is defined as follows: given a labeled graph G and a regular language L, given by means of an NFA, DFA, or RE, find an Eulerian path in G that satisfies L.

▶ **Example 2.** We describe some chain REs that are useful specifications.

Zone patrolling and periodic checks. Consider a communication, social, or physical network. Let a, b, c, and d be labels of edges in different zones of the network. We may want to patrol the network in a certain pattern. For example, in security, one may want a guard to patrol the zones of a physical network in a certain order, and in commercial applications, one may want to do the same with an advertisement that traverses a social network. Likewise, several communication protocols are based on the fact that a message must patrol different zones of the network in some predefined order before reaching its destination; e.g., in Onion routing, where the message is encrypted in layers, or in *proof-of-work* protocols that are used to deter denial of service attacks and other service abuses such as spam. For this, REs of the form $(a^+b^+c^+d^+)^*$, where σ^+ stands for $\sigma\sigma^*$, may be useful. If the pattern of visits is of a known bounded length, it can be specified as a conjunction of 1-wide chain REs, say $a^+b^+c^+d^+a^+b^+c^+d^+$.

As another example, let s label edges in which a checksum is performed on the message, and let σ label all other edges. We may want a message to be periodically checked for corruptions. This can be specified by the chain RE $(\sigma^i s)^*$, for the duration i after which a check should be performed. If we want the specification to be more flexible, say allow different durations between successive checks, all bounded by i, this is possible, but the RE is no longer a chain RE.

Bounded-delay response and FIFO. Let r, r_1, and r_2 label edges in which requests are submitted, possibly parameterized by the user that submits them, and let g, g_1, and g_2 label edges in which requests are granted, again possibly parameterized by the granted user. The semantics of requests and grants depend on the type of the network. Suppose there can be at most one request in the graph and we want a request, if submitted, to be followed by a grant within 3 steps. Let σ label all edges that are not labeled r or g. This can be specified by $R = \sigma^* + \sigma^* r g \sigma^* + \sigma^* r \sigma g \sigma^* + \sigma^* r \sigma \sigma g \sigma^*$. Note that R is a union of 1-wide 4-diverse chain REs. In case of a grant bounded by k steps, the REs are 1-wide $(k+1)$-diverse chain REs. If there are two requests and we want them to be granted in a FIFO order, the specification is $R = \sigma^* + \sigma^* r_1 \sigma^* g_1 \sigma^* + \sigma^* r_1 \sigma^* r_2 \sigma^* g_1 \sigma^* g_2 \sigma^* + \sigma^* r_1 \sigma^* g_1 \sigma^* r_2 \sigma^* g_2 \sigma^*$, and dually when only r_2 is submitted or when r_2 is before r_1. Note that R is a union of 1-wide 5-diverse chain REs.

3 It Is Hard

In this section we study the general CEP problem and show that it is NP-complete for specifications given by NFAs, DFAs, or REs. The reductions in this section are simple. We give them here for completeness and as a warm-up before things get more complicated in the next sections.

▶ **Theorem 3.** *The CEP problem is NP-complete.*

Proof. We start with membership in NP. Checking the membership of a given word in the language of a given NFA, DFA, or RE can be done in polynomial time. Hence, given a labeled graph $G = \langle V, E, l \rangle$ and a regular language L given by means of an NFA, DFA, or RE, checking whether a sequence P of $|E|$ edges is an EP in G and that $l(P) \in L$ can be done in polynomial time.

We prove NP-hardness by a reduction from the Hamiltonian-path problem. We prove hardness for specifications given by DFAs. Hardness for NFAs follows immediately, and hardness for REs follows from the polynomial translation of DFAs to REs. Given a graph $G = \langle V, E \rangle$, we construct a labeled graph G' and a DFA $\mathcal{A}$ such that there is a Hamiltonian path in G iff there is an EP in G' that satisfies $L(\mathcal{A})$. The labeled graph G' is over the alphabet V. It consists of a single vertex u with $|V|$ parallel self loops, each labeled by a different vertex in V. That is, the EPs of G' correspond to permutations of V. We define the specification DFA $\mathcal{A}$ so that $L(\mathcal{A})$ includes exactly all words that label paths in G. It is easy to define $\mathcal{A}$ as above by adding to G an initial state that has a transition to all vertices, and labeling all transitions, including the new ones, by their destination vertex. All the states in $\mathcal{A}$ are final. Now, there is a Hamiltonian path in G iff there is a permutation of V that forms a path in G iff there is an EP in G' that satisfies $L(\mathcal{A})$. ◀

The graph G' used in the proof of Theorem 3 is not simple. It is easy, however, to add a new letter # to the alphabet V and obtain a simple graph G'' by replacing every (parallel) self-loop labeled v in G' by a (disjoint) cycle labeled $v\#$ in G''. Hence the following theorem.

▶ **Theorem 4.** *The CEP problem is NP-hard already for simple graphs.*

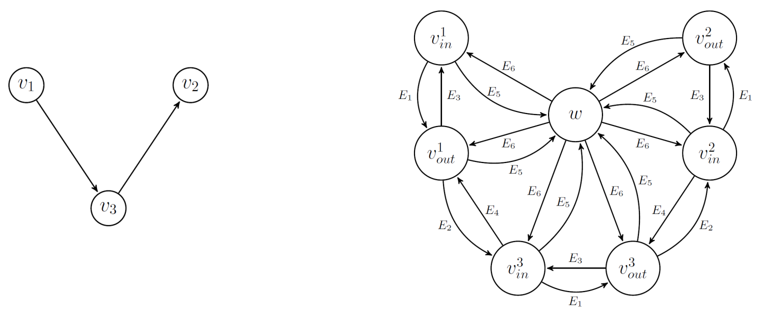


Figure 1 A graph and its Eulerian closure.

4 It Is Very Hard

While the graph G'' defined in the reduction in the proof of Theorem 4 is simple, the DFA $\mathcal{A}$ used in the proof is essentially the graph G. This suggests that we may do better with specifications of a constant size. In this section we show that the CEP problem is NP-hard already for more restricted cases, in particular for specification of a constant size. As we then show in Section 5, the cases we point to are tight, in the sense that tightening the restrictions makes the problem feasible.

We first define the *Eulerian closure* of a given graph – a construction that is going to be useful in some of our reductions. Given a graph $G = \langle V, E\rangle$, the *Eulerian closure of* G is the graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E}\rangle$, defined as follows (see an Example in Figure 1).

For each vertex $v \in V$ we include in $\mathcal{V}$ two vertices v_{in} and v_{out}. We also include in $\mathcal{V}$ a new vertex w. That is, $\mathcal{V} = \{v_{in} : v \in V\} \cup \{v_{out} : v \in V\} \cup \{w\}$. The set of edges of $\mathcal{G}$ is the union $\mathcal{E} = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6$, defined below:

- $E_1 = \{(v_{in}, v_{out}) : v \in V\}$. Thus, E_1 includes an edge for each vertex in G.
- $E_2 = \{(u_{out}, v_{in}) : (u, v) \in E\}$. Thus, E_2 includes an edge for every edge in G.
- $E_3 = \{(v_{out}, v_{in}) : v \in V\}$. Thus, E_3 includes an opposite edge for every edge in E_1.
- $E_4 = \{(v_{in}, u_{out}) : (u, v) \in E\}$. Thus, E_4 includes an opposite edge for every edge in E_2.
- $E_5 = \{(v, w) : v \in \mathcal{V} \setminus \{w\}\}$.
- $E_6 = \{(w, v) : v \in \mathcal{V} \setminus \{w\}\}$.

Note that, by Theorem 1, the graph $\mathcal{G}$ is Eulerian. Indeed, the edges in E_5 and E_6 guarantee that $\mathcal{G}$ is strongly connected regardless of the connectivity of G. Also, for every vertex $v \in \mathcal{V}$, we have $indeg(v) = outdeg(v)$. Finally, if G is simple, then so is $\mathcal{G}$.

▶ **Theorem 5.** *The CEP problem is NP-hard already for a simple graph and a specification given by fixed-size RE that can be expressed by a DFA with two states.*

Proof. We show a reduction from the problem of Hamiltonian path for simple graphs. Given a simple graph $G = \langle V, E\rangle$, we construct a simple labeled graph G' and a RE R that can be expressed by a DFA with two states, such that there is a Hamiltonian path in G iff there is an EP in G' that satisfies $L(R)$. The graph G' is defined by $G' = \langle \mathcal{V}, \mathcal{E}, l\rangle$ where $\langle \mathcal{V}, \mathcal{E}\rangle$ is the Eulerian closure of G, and $l(e)$ is $\{a\}$ if $e \in E_1$, is $\{b\}$ if $e \in E_2$, and is $\{c\}$ otherwise. Let $R = (a + b)^*(b + c)^*$. Note that $L(R)$ can be expressed by a DFA with two states. In the full version of the paper we prove that the reduction is correct. For this, we show that the way we have defined the Eulerian closure of G guarantees that if there is a Hamiltonian path in G, then it induces a path in G' that contains only edges labeled by a or b and can be extended to an Eulerian cycle by appending a path that contains only edges labeled by b or

c. Also, every EP in G' that satisfies $L(R)$ starts with a subpath that induces a Hamiltonian path in G. ◀

We continue and show that the CEP problem is NP-hard already for singleton specifications, namely when the specification consists of a single word, and for specifications given by a fixed-size RE without union and without nested Kleene star operators.

▶ **Theorem 6.** *The CEP problem is NP-hard for singleton specifications and for specifications given by a fixed-size* 2*-wide* 2*-diverse chain RE.*

Proof. In both cases, we show a reduction from the problem of Hamiltonian path for simple graphs. Let $G = \langle V, E\rangle$ be a simple graph, and let $V = \{v^1, \dots, v^n\}$. We define the simple graph $G' = \langle \mathcal{V}, \mathcal{E}, l\rangle$, where $\langle \mathcal{V}, \mathcal{E}\rangle$ is the Eulerian closure of G, and $l(e)$ is $\{a\}$ if $e \in E_1$, is $\{b\}$ if $e \in E_2 \cup E_6$, and is $\{c\}$ otherwise.

Consider the word $x = a(ba)^{n-1}c^{2n-1}(cb)^{|E|+n+1}$. We prove that there is a Hamiltonian path in G iff there is an EP in G' that is labeled by x. First, it is not hard to see that if G' has an EP labeled x, then the prefix $a(ba)^{n-1}$ of x induces a Hamiltonian path in G. Now, assume that there is a Hamiltonian path P in G from v^1 to v^n. We construct an Eulerian cycle in G' as follows: The Eulerian cycle starts with the path Q in G' corresponding to P. This path starts with the edge (v^1_{in}, v^1_{out}), ends with the edge (v^n_{in}, v^n_{out}), and visits the edge (v^i_{in}, v^i_{out}) for every i exactly once. Note that $l(Q) = a(ba)^{n-1}$. In the full version of the paper we show how the way the Eulerian closure is defined enables us to continue Q as required.

We continue to the second class. Let R be the RE $R = a(ba)^*c^*(cb)^*$. Note that R is indeed a fixed-size 2-wide 2-diverse chain RE. We show that there is a Hamiltonian path in G iff there is an EP in G' that satisfies $L(R)$. Again, it is not hard to see that if G' has an EP that satisfies $L(R)$, then the prefix $a(ba)^*$ of R induces a Hamiltonian path in G. Also, if there is a Hamiltonian path in G, then an Eulerian cycle in G' that satisfies $L(R)$ can be constructed as in the case of the word x.[3] ◀

5 But Sometimes It Is Easy

In this section we show classes of regular languages for which the CEP problem can be solved in polynomial time. By Theorem 6, the CEP problem is NP-hard for fixed-size 2-wide 2-diverse chain REs. We show that when one of the width and diversity parameters is tightened, the problem becomes feasible. We start with diversity and show that when a chain RE R is 1-diverse, we can solve the CEP problem for it in polynomial time even when R and its blocks are not of a fixed size.

▶ **Theorem 7.** *The CEP problem can be solved in polynomial time for specifications given by a* 1*-diverse chain RE.*

Proof. Let $G = \langle V, E, l\rangle$ be a labeled graph and let $R = b_1 \dots b_k$ be a RE, where for every i we have $b_i = w_i^*$ or $b_i = w_i$ for some $w_i \in \Sigma^*$, and every $\sigma \in \Sigma$ appears at most once in R. We assume that every letter that appears in G, appears also in R, because otherwise the CEP problem is trivial. We show how to find an EP that satisfies $L(R)$ and that starts in a vertex $v_1 \in V$. Note that since every $\sigma \in \Sigma$ appears at most once in R, then the subpath that

[3] Note that we could have defined the RE R to be $a^*(ba)^*c^*(cb)^*$, implying that NP-hardness applies already for fixed-size 2-wide 2-diverse chain REs in which all blocks have a Kleene star.

corresponds to a block b_i must be an EP in the subgraph G_i induced by the edges labeled by letters in w_i. Therefore, the first vertex in every subpath determines the last vertex in this subpath: if the degrees in every vertex in G_i are balanced, that is, the in degree is equal to the out degree for every vertex, then an EP must be a cycle; if the degrees are not balanced then an EP must start in the only vertex s_i where $outdeg(s_i) = indeg(s_i) + 1$ and end in the only vertex t_i where $indeg(t_i) = outdeg(t_i) + 1$. Thus, the algorithm checks whether G_1 has an EP from v_1 that corresponds to b_1, and if it does then it finds the vertex v_2 in which this EP ends. Then the algorithm checks whether G_2 has an EP from v_2 that corresponds to b_2 and continues similarly.

We now show how to find an EP in $G_i = \langle V_i, E_i, l\rangle$ that starts in a vertex v_i and corresponds to b_i. If $b_i = w_i$, then we check whether G_i contains every letter in w_i exactly once, and whether w_i induces a path in G_i from the vertex v_i. Assume now that $b_i = w_i^*$, and $w_i = \sigma_0 \ldots \sigma_{n-1}$. We construct a graph $G'_i = \langle V'_i, E'_i\rangle$, where $V'_i = \{\langle v, j\rangle : v \in V_i$ and $0 \le j \le n-1\}$ and $E'_i = \{(\langle u, j\rangle, \langle v, j+1 \ (mod\ n)\rangle) : (u, v) \in E_i$ and $l((u, v)) = \sigma_j\}$. Thus, G'_i includes n copies of G_i such that every edge (u, v) with $l((u, v)) = \sigma_j$ in G_i induces an edge from u in the j-th copy to v in the $(j+1)$-th copy in G'_i. Note that there is an EP in G_i from the vertex v_i that corresponds to b_i, iff there is an EP in G'_i from the vertex $\langle v_i, 0\rangle$ to some vertex $\langle u, 0\rangle$. Therefore, the problem is reduced to finding an EP from $\langle v_i, 0\rangle$ in G'_i. ◀

We continue and show that tightening the width also makes the problem solvable in polynomial time. We first describe another well-studied problem that we show to be strongly related to our problem. Let $G = \langle V, E\rangle$ be a directed or undirected graph and let (s_i, t_i), for $i = 1, \ldots, k$, be k ordered pairs of vertices. In the *edge-disjoint paths* problem (EDP problem, for short), we need to find, for every $1 \le i \le k$, a path in G from s_i to t_i such that the paths are edge-disjoint; that is, an edge cannot appear in more than one path. The EDP problem is NP-complete for both directed and undirected graphs [29]. When the graph is undirected and k is fixed, there is a polynomial-time algorithm ([28, 20]). For directed graphs, the problem is NP-complete already when $k = 2$ [14]. The directed graph $G_D = \langle V, E_D\rangle$ where $E_D = \{(t_i, s_i) : i = 1 \ldots k\}$ is called the *demand graph*.

Consider the graph $G + G_D = \langle V, E \cup E_D\rangle$ obtained by adding to G the edges from G_D. When $G + G_D$ is Eulerian, we say that there is an *Eulerian promise on the demand*. It is shown in [18] that when there is an Eulerian promise on the demand, there is a polynomial-time algorithm for the EDP problem with $k = 3$. It is also conjectured in [18] that when there is an Eulerian promise on the demand, there is a polynomial-time algorithm for the EDP problem for every fixed k. To the best of our knowledge, however, this problem is still open (it is also declared open in [29, 26]).

We first relate the EDP problem to the problem of finding an EP that respects a linear order on the subset of the edges.

▶ **Lemma 8.** *Consider a directed graph $G = \langle V, E\rangle$ and two vertices $s, t \in V$. Let $e_1, \ldots, e_k$ be some edges in E and $\tau = e_1 \prec \ldots \prec e_k$ be an order on them. If $k \le 2$, then finding an EP from s to t that respects τ can be done in polynomial time. If $k > 2$ is fixed, then finding an EP from s to t that respects τ can be solved in polynomial time iff the EDP problem for a directed graph with an Eulerian promise on the demand can be solved in polynomial time for $k + 1$ paths.*

We now relate the CEP problem for fixed-size 1-wide chain REs to the problem of finding an EP that respects a linear order on a subset of the edges. Lemma 8 then enables us to relate the former also to the EDP problem, implying that restricting the width also leads to a polynomial complexity.

▶ **Theorem 9.** *The CEP problem can be solved in polynomial time for specifications given by a fixed-size* 1*-wide* 3*-star-diverse chain RE. For specifications given by a fixed-size* 1*-wide* d*-star-diverse chain RE, the CEP problem can be solved in polynomial time iff the EDP problem for a directed graph with an Eulerian promise on the demand can be solved in polynomial time for* d *paths.*

Proof. Let $R = b_1 \dots b_k$ for a fixed k, where for every i we have $b_i = \sigma_i^*$ or $b_i = \sigma_i$ for some $\sigma_i \in \Sigma$. We assume that every letter that appears in G appears also in R. Indeed, otherwise the CEP problem is trivial. We run over all the options for choosing $k+1$ vertices $v_0, \dots, v_k \in V$ (there are $|V|^{k+1}$ such options), and check whether G has an EP from v_0 to v_k that satisfies $L(R)$ and can be partitioned into k subpaths, such that the i-th subpath starts in v_{i-1}, ends in v_i and corresponds to b_i. We now describe how to perform this check.

For every i such that $b_i = \sigma$ for some $\sigma \in \Sigma$, the graph G must have an edge $e = (v_{i-1}, v_i)$ with $l(e) = \sigma$. All the other subpaths corresponding to b_j for $j \neq i$, cannot use the edge e. In particular, if b_j with $j \neq i$ is a single-letter block, then it cannot use the edge e. In the rest of this proof we assume that for every single-letter block $b_n = \sigma_n$ there is an edge $e_n = (v_{n-1}, v_n)$ such that $l(e_n) = \sigma_n$, and that if b_m is a single-letter block with $m \neq n$ then e_m and e_n are different edges (they can be parallel edges). We denote the set of edges that correspond to a single-letter block by E_s.

Let $\sigma \in \Sigma$ and let $b_{i_1}, \dots, b_{i_m}$ be the blocks in R such that for every $1 \leq j \leq m$ we have $b_{i_j} = \sigma^*$. Let G_σ be the subgraph of G induced by the edges $\{e \in E \setminus E_s : l(e) = \sigma\}$. Let G'_σ be the graph obtained from G_σ by adding, for every $1 \leq j \leq m-1$, an edge $e_{i_j} = (v_{i_j}, v_{i_{j+1}-1})$; that is, we add for every i_j an edge from the end of the block i_j to the beginning of the block i_{j+1}. We check whether there is an EP in G'_σ from v_{i_1-1} to v_{i_m} such that $e_{i_1} \prec e_{i_2} \prec \dots \prec e_{i_{m-1}}$. In the full version we prove that such an EP exists in G'_σ for every σ iff the required EP in G exists.

Thus, we reduce the CEP problem to the problem of finding an EP that respects a linear order on a subset of the edges. By Lemma 8, the latter can be reduced to the EDP problem. Accordingly, we have a polynomial-time algorithm if for every $\sigma \in \Sigma$ the expression σ^*, appears at most three times in R (that is, R is 3-star-diverse). Also, if the EDP problem for a directed graph with an Eulerian promise on the demand can be solved in polynomial time for d paths, then, according to Lemma 8, we have a polynomial-time algorithm also if R is d-star-diverse.

Now, assume that there is a polynomial-time algorithm for the case R is d-star-diverse. Let $H = \langle V_H, E_H \rangle$ be a graph and let $e_1, \dots, e_{d-1} \in E_H$. Let H' be a labeled graph obtained from H by assigning a label σ_i for every edge e_i, and assigning a label σ for every other edge in E_H. Note that H has an EP with $e_1 \prec \dots \prec e_{d-1}$ iff H' has an EP that satisfies $L(R)$ for $R = \sigma^* \sigma_1 \sigma^* \sigma_2 \dots \sigma^* \sigma_{d-1} \sigma^*$. By our assumption, the latter problem can be solved in polynomial time. Therefore, by Lemma 8, the EDP problem for a directed graph with an Eulerian promise on the demand can be solved in polynomial time for d paths. ◀

We conclude that there is a polynomial-time algorithm for every specification that is a disjunction of polynomially many REs of the forms handled in Theorems 7 and 9. As demonstrated in Example 2, such disjuncts can specify useful behaviors. We note that with some additional ‘’technical acrobatics”, it is possible to squeeze the lemon some more and point to additional classes of chain REs that can be solved in polynomial time. For example, if $R = b_1 \dots b_k$, where k is a fixed number and for every $1 \leq i \leq k$ we have $b_i = w_i^*$ for some

$w_i \in \Sigma^*$, or $b_i = \sigma_i$ for some $\sigma_i \in \Sigma$, then it is possible to restrict the diversity of the letters, but allow repetitions of identical blocks, so that polynomial-time algorithms can be obtained by combining ideas used in the proofs of Theorems 9 and 7. We do not find, however, these special cases or technical acrobatics too interesting. A good intuitive and practical conclusion is that when the specification is a chain RE, it is recommended to use the ideas here in order to find the complexity of the CEP problem for it, and possibly to decompose or alter it to a disjunction of specifications of lower width and diversity for which a polynomial algorithm is possible.

6 Multi-Labeled Graphs

A *multi-labeled graph* is a tuple $G = \langle V, E, l \rangle$ where $\langle V, E \rangle$ is a graph and $l : E \to 2^{\Sigma}$ maps every edge to a subset of letters from the alphabet Σ. For a path $P = e_1, \ldots, e_k$ we define $l(P) = \{\sigma_1 \ldots \sigma_k : \sigma_i \in l(e_i)$ for every $1 \leq i \leq k\}$. We say that P satisfies a specification $L \subseteq \Sigma^*$ if $l(P) \cap L \neq \emptyset$. Note that we take here the existential approach in model checking, where satisfaction amounts to an existence of a correct path.

We show that if the graph is multi-labeled, the CEP problem is NP-hard already for the RE a^*b^*. Note that a^*b^* is a fixed-size 1-wide 1-diverse chain RE. Thus, by both Theorems 9 and 7, the CEP problem in graphs in which each edge is labeled by a single letter can be solved in polynomial time.

▶ **Theorem 10.** *The CEP problem for multi-labeled simple graphs is NP-hard already for the specification given by the RE $R = a^*b^*$.*

Proof. We show a reduction from the problem of Hamiltonian path for simple graphs. Given a simple graph $G = \langle V, E \rangle$, we construct a multi-labeled simple graph G', such that there is a Hamiltonian path in G iff there is an EP in G' that satisfies $L(R)$. The graph G' is defined by $G' = \langle \mathcal{V}, \mathcal{E}, l \rangle$ where $\langle \mathcal{V}, \mathcal{E} \rangle$ is the Eulerian closure of G, and $l(e)$ is $\{a\}$ if $e \in E_1$, is $\{a, b\}$ if $e \in E_2$, and is $\{b\}$ otherwise. In the full version of the paper we prove the correctness of the reduction. ◀

7 The Constrained Chinese Postman Problem

A *weighted graph* is a tuple $G = \langle V, E, c \rangle$, where $\langle V, E \rangle$ is a graph and $c : E \to \mathbb{R}^+$ maps every edge to a non-negative cost. The cost of a path $P = e_1, \ldots, e_k$, denoted $c(P)$, is $\sum_{i=1}^{k} c(e_i)$; that is, the sum of the costs of the edges along the path. A *postman path* in G is a path that visits every edge in E at least once. An *optimal postman path* is a least-cost postman path. Similar definitions apply to cycles. In the well-studied *Chinese postman problem*, we are given a weighted graph G and need to find an optimal postman path. Clearly, when G has an EP, it induces an optimal postman path. Otherwise, the goal is to get as close as possible to an EP. Thus, the Chinese postman problem can be viewed as an optimization variant of the EP problem. The combinatorial simplicity of the EP problem is carried over to the Chinese postman problem. In particular, it has a well-known polynomial-time algorithm [12].

A *labeled weighted graph* is a tuple $G = \langle V, E, l, c \rangle$, where $\langle V, E, l \rangle$ is a labeled graph and $\langle V, E, c \rangle$ is a weighted graph. For a regular language L, an *L-postman path* in G is a path that satisfies L and visits every edge in E at least once. An *optimal L-postman path* is a least-cost L-postman path. In the *constrained Chinese postman problem* (CCP problem, for short), we are given a labeled weighted graph G and a regular language L and need to find an optimal L-postman path in G. In this section we study the CCP problem.

First, we show that the corresponding decision problem is NP-complete, and is NP-hard already for restricted classes of graphs and specifications.

▶ **Theorem 11.** *Consider a labeled weighted graph G and a regular language L given by an NFA, DFA, or RE. For $k \in I\!R^+$, deciding whether there is an L-postman path P with $c(P) \leq k$ is NP-complete. Furthermore, it is NP-hard already when G is simple, when L is a singleton, when L is given by a DFA with two states, and when L is given by a fixed-size 2-wide 2-diverse chain RE.*

Proof. First, note that a labeled graph $H = \langle V_H, E_H, l_H \rangle$ has an EP that satisfies L iff the labeled weighted graph $H' = \langle V_H, E_H, l_H, c\rangle$ with $c(e) = 1$ for every $e \in E_H$ has an L-postman path with cost $|E|$. Thus, the lower bounds follow from Theorems 4, 5, and 6.

We prove the upper bound for L given by an NFA $\mathcal{A}$. The other cases follow. Let $G = \langle V, E, l, c\rangle$, $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, F\rangle$, and let $\mathcal{N}$ be the product NFA of $\mathcal{A}$ and G. Formally, $\mathcal{N} = \langle \Sigma, Q \times V, Q_0 \times V, \delta', F \times V\rangle$, where $\delta'(\langle q, v\rangle, \sigma) = \{\langle q', v'\rangle : q' \in \delta(q, \sigma), (v, v') \in E$, and $l((v, v')) = \sigma\}$. Note that $\mathcal{N}$ ignores parallel edges in G. Also, a path P in G satisfies $L(\mathcal{A})$ iff there is an accepting run in $\mathcal{N}$ whose projection on V corresponds to P. We claim that there is an $L(\mathcal{A})$-postman path P in G with $c(P) \leq k$ iff there is an $L(\mathcal{A})$-postman path $P' = e_1, \ldots, e_n$ in G with $c(P') \leq k$ of length $n \leq |E| \cdot |Q| \cdot |V|$. Let r be an accepting run in $\mathcal{N}$ whose projection on V corresponds to a path $P = e_1, \ldots, e_n$ in G, and assume that P includes every edge in G (including parallel edges) at least once. Let $i_1 < \ldots < i_{|E|}$ be the indices in which all edges appear for the first time in P. If for some j there are more than $|Q| \cdot |V|$ states between the appearance in r of the transition that corresponds to the edge e_{i_j} and the appearance of the transition that corresponds to the edge $e_{i_{j+1}}$, then r has a loop that can be avoided. By removing these loops we end up with a path of length $n \leq |E| \cdot |Q| \cdot |V|$. Thus, a witness for having an $L(\mathcal{A})$-postman path P in G with $c(P) \leq k$ is of size at most $n \leq |E| \cdot |Q| \cdot |V|$. ◀

Since the CCP problem is at least as hard as the CEP problem, we turn to consider cases for which the CEP problem is solvable in polynomial time. In particular, we restrict further the class of fixed-size 2-wide 2-diverse chain RE. First by restricting the width, and then the diversity.

▶ **Theorem 12.** *The CCP problem can be solved in polynomial time for specifications given by a fixed-size 1-wide 2-star-diverse chain RE.*

Proof. Let $G = \langle V, E, l, c\rangle$ and let $R = b_1 \ldots b_k$ be a 1-wide 2-star-diverse chain RE. We assume that every letter that appears in G, appears also in R. Indeed, otherwise the CCP problem is trivial. We run over all the (polynomially many) options for choosing $k+1$ vertices $v_0, \ldots, v_k \in V$, and find a least-cost path in G from v_0 to v_k that satisfies $L(R)$, contains all edges, and can be partitioned into k subpaths such that the i-th subpath starts in v_{i-1}, ends in v_i, and satisfies $L(b_i)$.

First, assume that $b_i = \sigma_i^*$ for every i; that is, R does not contain single-letter blocks. Let $\sigma \in \Sigma$. If σ^* appears exactly once in R and $\sigma = \sigma_i$, then we find an optimal postman path from v_{i-1} to v_i in the subgraph G_σ induced by the edges in G labeled by σ. If σ^* appears twice in R, let $\sigma = \sigma_i = \sigma_j$ with $i < j$. We construct a weighted graph G'_σ by adding to G_σ the edge (v_i, v_{j-1}) with a large cost. Now we need to find a least-cost path in G'_σ from v_{i-1} to v_j in which every edge appears at least once, and the new edge (v_i, v_{j-1}) appears exactly once. Since the edge (v_i, v_{j-1}) has a large cost, it can be done simply by finding an optimal postman path from v_{i-1} to v_j in G'_σ. Finally, as in Theorem 9, we construct a path

P by concatenating the corresponding paths for every block b_i in R. In the full version we describe how to handle the case where R contains single-letter blocks. ◀

The case of 1-diverse chain REs follows similar considerations and applies the ideas used in the proof of Theorem 7 in the case of the CEP problem.

▶ **Theorem 13.** *The CCP problem can be solved in polynomial time for specifications given by a* 1*-diverse chain RE with a fixed number of blocks.*

8 Eulerian Languages

The *Eulerian language* of a Σ-labeled graph G, denoted $EL(G)$, is the set of words read along Eulerian paths in G. Formally, $EL(G) = \{l(P) \in \Sigma^* : P \text{ is an EP in } G\}$. Clearly, the *nonemptiness problem*, namely deciding whether $EL(G) \neq \emptyset$, coincides with the EP problem and can thus be solved in polynomial time. Given a regular language $L \subseteq \Sigma^*$, the *satisfaction problem* for G and L is to decide whether $EL(G) \cap L \neq \emptyset$. It is easy to see that the satisfaction problem coincides with the CEP problem, and is thus NP-complete (Theorem 3). Given a word $w \in \Sigma^*$, the *membership problem* for G and w is to decide whether $w \in EL(G)$. By Theorem 6, the CEP problem is NP-complete also for singleton specifications, implying that so is the membership problem.

In this section we study additional problems about the Eulerian language of G. Problems that compare it with other languages, given by an NFA, DFA, or RE, or given as the Eulerian language of another graph. Not all our complexities are tight, but we are able to place all problems in different levels of the polynomial hierarchy.

▶ **Theorem 14.** *Consider a labeled graph G and a specification L given by an NFA, DFA, or RE. Deciding whether $EL(G) \subseteq L$ is co-NP-complete. Furthermore, it is co-NP-hard already for a fixed-size specification.*

Proof. For the upper bound, note that a witness for $EL(G) \not\subseteq L$, namely an EP in G that does not satisfy L, can be verified in polynomial time. For the lower bound, recall that the CEP problem is NP-hard already for fixed-size specifications (Theorem 5). Observe that there is an EP that satisfies L iff there is an EP that does not satisfy $\Sigma^* \setminus L$. Since L is given by a fixed-size NFA, DFA, or RE, the size of an NFA, DFA, or RE for its complement $\Sigma^* \setminus L$ is also fixed, and we are done. ◀

▶ **Theorem 15.** *Consider a labeled graph G and a specification L given by an NFA, DFA, or RE. Deciding whether $L \subseteq EL(G)$ is in Π_2^p and is NP-hard.*

Proof. The lower bound follows from the NP-hardness of the membership problem. The upper bound follows from the fact that deciding whether $L \not\subseteq EL(G)$ can be done with a nondeterministic polynomial-time Turing machine that uses an oracle for the membership problem. ◀

▶ **Theorem 16.** *Consider two labeled graphs G and G'. Deciding whether $EL(G') \subseteq EL(G)$ and deciding whether $EL(G') \cap EL(G) \neq \emptyset$ is in Π_2^p and Σ_2^p respectively, and is NP-hard.*

Proof. For the lower bound, we show a reduction from the membership problem. Given a word w and a graph G, we construct a graph G' such that $EL(G') = \{w\}$. Now, $w \in EL(G)$ iff $EL(G') \subseteq EL(G)$ iff $EL(G') \cap EL(G) \neq \emptyset$. For the upper bound, observe that deciding whether $EL(G') \not\subseteq EL(G)$ and whether $EL(G') \cap EL(G) \neq \emptyset$ can be done with a nondeterministic polynomial-time Turing machine that uses an oracle for the membership problem. ◀

Since $EL(G)$ contains only words of a fixed length, we know that $EL(G)$ is finite and hence regular. On the other hand, given a regular language $L \subseteq \Sigma^*$, even one all whose words are of the same length, it is not clear whether there is a graph G such that $EL(G) = L$. For example, it is possible to find a two-state labeled graph G such that $EL(G) = \{abcd, adbc, cbad, cdba\}$ (the reader is encouraged to search for it). but it is impossible to add $abdc$ to the Eulerian language. An upper bound for the problem follows from our ability to bound the number of edges in the candidate graph G. The tight complexity, however, is still open.

▶ **Theorem 17.** *For a language L given by an NFA, DFA, or RE, deciding whether there is a labeled graph G such that $EL(G) = L$ is in Σ_3^p.*

Proof. Follows from the fact that it can be done by a nondeterministic polynomial-time Turing machine with oracles for the problems described in Theorems 14 and 15. ◀

References

1 S. Abiteboul and V. Vianu. Regular path queries with constraints. *J. Comput. Syst. Sci.*, 58(3):428–452, 1999.

2 E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.

3 G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. In *Proc. 17th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 8412 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 2014.

4 C. Barrett, R. Jacob, and M. Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

5 V. Blue, J. Adler, and G. List. Real-time multiple-objective path search for in-vehicle route guidance systems. *Journal of the Transportation Research Board*, 1588:10–17, 1997.

6 P.G. Bradford and D.A. Thomas. Labeled shortest paths in digraphs with negative and positive edge weights. *RAIRO-Theoretical Informatics and Applications*, 43(03):567–583, 2009.

7 A.L. Buchsbaum, P.C. Kanellakis, and J.S. Vitter. A data structure for arc insertion and regular path finding. *Annals of Mathematics and Artificial Intelligence*, 3(2-4):187–210, 1991.

8 D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Reasoning on regular path queries. *ACM SIGMOD Record*, 32(4):83–92, 2003.

9 W. Cheng and M. Pedram. Power-optimal encoding for a DRAM address bus. *IEEE Trans. VLSI Syst.*, 10(2):109–118, 2002.

10 T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 1990.

11 M. Dror, H. Stern, and P. Trudeau. Postman tour on a graph with precedence relation on arcs. *Networks*, 17(3):283–294, 1987.

12 H.A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995.

13 L.R. Ford and D.R. Fulkerson. *Flows in networks.* Princeton Univ. Press, Princeton, 1962.

14 S. J. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:11–121, 1980.

15 M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. Freeman and Co., 1979.

16 G. Ghiani and G. Improta. An algorithm for the hierarchical chinese postman problem. *Operations Research Letters*, 26(1):27–32, 2000.

17 S. Hannenhalli, W. Feldman, H.F. Lewis, S.S. Skiena, and P.A. Pevzner. Positional sequencing by hybridization. *Computer applications in the biosciences: CABIOS*, 12(1):19–24, 1996.

18 T. Ibaraki and S. Poljak. Weak three-linking in eulerian digraphs. *SIAM journal on Discrete Mathematics*, 4(1):84–98, 1991.

19 J. Kari. Synchronizing finite automata on Eulerian digraphs. *Theoretical Computer Science*, 295:223–232, 2003.

20 K.I. Kawarabayashi, Y. Kobayashi, and B. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424–435, 2012.

21 H.L.M. Kerivin, M. Lacroix, and A.R. Mahjoub. On the complexity of the Eulerian closed walk with precedence path constraints problem. *Theoretical Computer Science*, 439:16–29, 2012.

22 P. Korteweg and T. Volgenant. On the hierarchical chinese postman problem with linear ordered classes. *European Journal of Operational Research*, 169(1):41–52, 2006.

23 O. Kupferman and T. Tamir. Properties and utilization of capacitated automata. In *Proc. 34th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 29 of *LIPIcs*, pages 33–44. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2014.

24 A.O. Mendelzon and P.T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

25 E. Moreno and M. Matamala. Minimal Eulerian circuit in a labeled digraph. In *LATIN 2006: Theoretical Informatics*, pages 737–744. Springer, 2006.

26 G. Naves and A. Sebő. Multiflow feasibility: an annotated tableau. In *Research Trends in Combinatorial Optimization*, pages 261–283. Springer, 2009.

27 P.A. Pevzner, H. Tang, and M.S. Waterman. An Eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

28 N. Robertson and P.D. Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

29 J. Vygen. Disjoint paths. report no. 94816. *Research Institute for Discrete Mathematics, University of Bonn*, 1994.

On the Exact Learnability of Graph Parameters: The Case of Partition Functions

Nadia Labai[*1] **and Johann A. Makowsky**[†2]

1 **Department of Informatics, Vienna University of Technology, Vienna, Austria**
labai@forsyte.at

2 **Department of Computer Science, Technion - Israel Institute of Technology, Haifa, Israel**
janos@cs.technion.ac.il

Abstract

We study the exact learnability of real valued graph parameters f which are known to be representable as partition functions which count the number of weighted homomorphisms into a graph H with vertex weights α and edge weights β. M. Freedman, L. Lovász and A. Schrijver have given a characterization of these graph parameters in terms of the k-connection matrices $C(f, k)$ of f. Our model of learnability is based on D. Angluin's model of exact learning using membership and equivalence queries. Given such a graph parameter f, the learner can ask for the values of f for graphs of their choice, and they can formulate hypotheses in terms of the connection matrices $C(f, k)$ of f. The teacher can accept the hypothesis as correct, or provide a counterexample consisting of a graph. Our main result shows that in this scenario, a very large class of partition functions, the rigid partition functions, can be learned in time polynomial in the size of H and the size of the largest counterexample in the Blum-Shub-Smale model of computation over the reals with unit cost.

1998 ACM Subject Classification I.2.6 Parameter learning, G.2.1 Combinatorics

Keywords and phrases Exact learning, partition function, weighted homomorphism, connection matrices

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.63

1 Introduction

A graph *parameter* $f : \mathcal{G} \to \mathcal{R}$ is a function from all finite graphs $\mathcal{G}$ into a ring or field $\mathcal{R}$, which is invariant under graph isomorphisms.

In this paper we initiate the study of exact learnability of graph parameters with values in $\mathcal{R}$, which is assumed to be either $\mathbb{Z}, \mathbb{Q}$ or $\mathbb{R}$. As this question seems new, we focus here on the special case of graph parameters given as partition functions, [10, 14]. We adapt the model of exact learning introduced by D. Angluin [1]. Our research extends the work of [3, 11], where exact learnability of languages (set of words or labeled trees) recognizable by multiplicity automata (aka weighted automata) was studied, to graph parameters with values in $\mathcal{R}$.

[*] Supported by the National Research Network RiSE (S114), and the LogiCS doctoral program (W1255) funded by the Austrian Science Fund (FWF).
[†] Partially supported by a grant of Technion Research Authority. This work was done [in part] while the author was visiting the Simons Institute for the Theory of Computing.

1.1 Exact learning

In each step, the learner may make *membership queries* VALUE(x) in which they ask for the value of the target f on specific input x. This is the analogue of the MEMBERSHIP queries used in the original model of exact learning, [2]. The learner may also propose a hypothesis h by sending an EQUIVALENT(h) query to the teacher. If the hypothesis is correct, the teacher returns "YES" and if it is incorrect, the teacher returns a counterexample. A class of functions is *exactly learnable* if there is a learner that for each target function f, outputs a hypothesis h such that $f(x) = h(x)$ for all x and does so in time polynomial in the size of a shortest representation of f and the size of a largest counterexample returned by the teacher.

1.2 Formulating a hypothesis

To make sense one has to specify the formalism (language) $\mathfrak{L}$ in which a hypothesis has to be formulated. It will be obvious in the sequel, that the restriction imposed by the choice of $\mathfrak{L}$ will determine whether f is learnable or not.

Let us look at the seemingly simpler case of learning integer functions $f : \mathbb{Z} \to \mathbb{Z}$ or integer valued functions of words $w \in \Sigma^\star$ over an alphabet in Σ.

(i) If f can be any function $f : \mathbb{Z} \to \mathbb{Z}$ or $f : \Sigma^\star \to \mathbb{Z}$, there are uncountably many candidate functions as hypotheses, and no finitary formalism $\mathfrak{L}$ is suitable to formulate a hypothesis.

(ii) If f is known to be a polynomial $p(X) = \sum_i a_i X^i \in \mathbb{Z}[X]$, we can formulate the hypothesis as a vector $\mathbf{a} = (a_1, \ldots, a_m)$ in $\mathbb{Z}^m$. Learning is successful if the learner finds the hypothesis $h = \mathbf{a}$ in the required time. Here Lagrange interpolation will be used to formulate the hypotheses.

(iii) If f is known to satisfy some recurrence relation, the hypothesis will consist of the coefficients and the length of the recurrence relation, and exact learnability will depend on the class of recurrence relations one has in mind.

(iv) If $f : \Sigma^\star \to \mathbb{Z}$ is a word function recognizable by a multiplicity automaton MA, the hypotheses are given by the weighted transition tables of MA, cf. [3].

Looking now at a graph parameter $f : \mathcal{G} \to \mathcal{R}$ what can we expect? Again we have to restrict our treatment to a class of parameters where each member can be described by a finite string in a formalism $\mathfrak{L}$.

We illustrate the varying difficulty of the learning problem with the example of the chromatic polynomial $\chi(G; X \in \mathbb{N}[X])$ for a graph G. For $X = k$, the evaluation of $\chi(G; k)$ counts the number of proper colorings of G with at most k colors. It is well known that for fixed G, $\chi(G; k)$ is indeed a polynomial in k, [4, 7]. A graph parameter f is a *chromatic invariant over $\mathcal{R}$* if

(i) it is multiplicative, i.e., for the disjoint union $G_1 \sqcup G_2$ of G_1 and G_2, it holds that $f(G_1 \sqcup G_2) = f(G_1) \cdot f(G_2)$, and

(ii) there are $\alpha, \beta, \gamma \in \mathcal{R}$ such that $f(G) = \alpha \cdot f(G_{-e}) + \beta \cdot f(G_{/e})$ and $f(K_1) = \gamma$.

K_n denotes the complete graph on n vertices, and G_{-e} and $G_{/e}$ are, respectively, the graphs obtained from deleting the edge e from G and contracting e in G.

The parameter $\chi(G; k)$ is a chromatic invariant with $\alpha = 1, \beta = -1$ and $\gamma = k$. Finally, $\chi(G; k)$ has an interpretation by counting homomorphisms:

$$\chi(G; m) = \sum_{t: G \to K_m} 1,$$

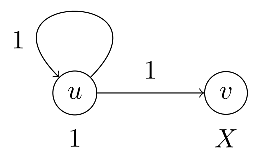


Figure 1 The weighted graph H_{indep}.

This is a special case of the homomorphism counting function for a fixed graph H:

$$\mathsf{hom}(G, H) = \sum_{t:G \to H} 1,$$

where t is a homomorphism $t : G \to H$.

Now, let a graph parameter $f : \mathcal{G} \to \mathcal{R}$ be the target of a learning algorithm.

(i) If f is known to be an instance of $\chi(G; X)$, a hypothesis consists of a value $X = a$. But in this case we know that $\chi(K_1; X) = X$, so it suffices to ask for $f(K_1) = a$.

(ii) If f is known to be a chromatic invariant, the hypothesis consists of the triple (α, β, γ). In this case a hypothesis can be computed from the values of $f(P_m)$ for undirected paths P_m for sufficiently many values of m.

(iii) If f is known to be an instance of $\mathsf{hom}(-, H)$, a hypothesis would consist of a target graph H.

1.3 Counting weighted homomorphisms aka partition functions

A *weighted graph* $H(\alpha, \beta)$ is a graph $H = (V(H), E(H))$ on $n = |V(H)|$ vertices together with a vertex weight function $\alpha : V(H) \to \mathbb{R}$, viewed as a vector of length n, and an edge weights function $\beta : V(H)^2 \to \mathbb{R}$ viewed as an $n \times n$ matrix, with $\beta(u, v) = 0$ if $(u, v) \notin E(H)$.

A *partition function*[1] $\mathsf{hom}(-, H(\alpha, \beta))$ is the generalization of $\mathsf{hom}(-, H)$ to weighted graphs, whose value on a graph G is defined as follows:

$$\mathsf{hom}(G, H(\alpha, \beta)) = \sum_{t:G \to H} \prod_{v \in V(G)} \alpha(t(v)) \prod_{(u,v) \in V(G)^2} \beta(t(u), t(v))$$

To illustrate the notion of a partition function, let H_{indep} be the graph with two vertices $\{u, v\}$ and the edges $\{(u, v), (u, u)\}$, shown in Figure 1. Let $\alpha(u) = 1, \alpha(v) = X$ and $\beta(u, v) = 1, \beta(u, u) = 1$. Then $\mathsf{hom}(-, H_{indep}(\alpha, \beta))$ is the *independence polynomial*,

$$\mathsf{hom}(G, H_{indep}(\alpha, \beta)) = I(G; X) = \sum_j ind_j(G) X^j$$

where $ind_j(G)$ is the number of independent sets of size j in the graph G.

We say a partition function $\mathsf{hom}(-, H(\alpha, \beta))$ is *rigid* aka *asymmetric* [2], if H has no proper automorphisms. Note that automorphisms in a weighted graph also respect vertex and edge weights. In our examples above, the evaluations of the independence polynomial are rigid partition functions, whereas the evaluations of the chromatic polynomial are not. It is known that almost all graphs are rigid:

[1] In the literature $\mathsf{hom}(-, H(\alpha, \beta))$ is also denoted by $Z_{H(\alpha,\beta)}(G)$, e.g., in [19]. We follow the notation of [14].

[2] Some authors say G is asymmetric if G has no proper automorphisms, and G is rigid if G has no proper endomorphisms, [12]. Wikipedia uses rigid as we use it here.

Algorithm 1 Learning algorithm for rigid partition functions

```
1: $n = 1$
2: while True do
3:     augment $M$ with($B_n$)
4:     $P$ = find basis($M$)
5:     $h$ = generate hypothesis($P$)
6:     if EQUIVALENT($h$) = YES then
7:         return $h$
8:     else
9:         $n = n + 1$
10:        $B_n$ = EQUIVALENT($h$)                    ▷ $B_n$ receives a counterexample
11:    end if
12: end while
```

▶ **Theorem 1** ([9, 12]). *Let G be a uniformly selected graph on n vertices. The probability that G is rigid tends to 1 as $n \to \infty$.*

If the target f is known to be a (rigid) partition function $\mathsf{hom}(-, H(\alpha, \beta))$ then the hypothesis consists of a (rigid) weighted graph $H(\alpha, \beta)$.

In Section 2 we give the characterization of rigid and non-rigid partition functions from [10, 15, 14] in terms of connection matrices.

For technical reasons discussed in Section 5, in this paper we deal only with the learnability of rigid partition functions, and leave the general case to future work.

1.4 Main result

Our main result can now be stated:

▶ **Theorem 2.** *Let f be a graph parameter which is known to be a rigid partition function $f(G) = \mathsf{hom}(G, H(\alpha, \beta))$. Then f can be learned in time polynomial in the size of H and the size of the largest counterexample in the Blum-Shub-Smale model of computation over the reals with unit cost.*

▶ **Remark 3.** *If f takes values in $\mathbb{Q}$ rather than in $\mathbb{R}$ we can also work in the Turing model of computation with logarithmic cost for the elements in $\mathbb{Q}$.*

To prove Theorem 2 we will use the characterization of rigid partition functions in terms of connection matrices, [14, Theorem 5.54], stated as Theorem 4 and Corollary 6 in Section 2. The difficulty of our result lies not in finding a learning algorithm by carefully manipulating the counterexamples to meet the complexity constraints, but in proving the algorithm correct. In order to do this we had to identify and extract the suitable algebraic properties underlying the proof of Theorem 4 and Corollary 6.

The learning algorithm is given in pseudo-code as Algorithm 1. It maintains a matrix M used in the generation of the hypothesis h from VALUE and EQUIVALENT query results. After an initial setup of M, in each iteration the algorithm generates a hypothesis h, queries the teacher for equivalence between h and the target and either terminates, or updates M accordingly and moves on to the next iteration.

It uses three black-boxes; find basis which uses M to find a certain basis P of a graph algebra associated with the target function (see Section 2), generate hypothesis which uses this basis and VALUE queries to construct a hypothesis h, and augment M which augments the matrix M after a counterexample is received, using VALUE queries.

We briefly overview the complexity of the algorithm to illustrate that rigid partition functions are indeed exactly learnable. Proofs of validity and detailed analysis of the complexity are given in later sections. For a target $H(\alpha,\beta)$ on q vertices, the procedure find basis solves $O(q)$ systems of linear equations, and systems of linear matrix equations, all of dimension $O(\text{poly}(q))$. The procedure generate hypothesis performs $O(q)$ graph operations of polynomial time complexity on graphs of size $O(\text{poly}(q,|x|))$, where $|x|$ is the size of the largest counterexample, and $O(q^2)$ VALUE queries. The procedure augment M performs $O(q)$ VALUE queries. Thus, each iteration takes time $O(\text{poly}(q,|x|))$. Lemma 18 will show that there are $O(q)$ iterations, so the total run time of the algorithm is polynomial in the size q of $H(\alpha,\beta)$ and the size $|x|$ of the largest counterexample.

Organization

In Section 2 we give the necessary background on partition functions and the graph algebras induced by them. Section 3 presents the algorithm in detail and in Section 4 we prove its validity and analyze its time complexity. We discuss the results and future work in Section 5. Due to space limitations, the appendix is included in the arXiv version, [13].

2 Preliminaries

Let $k \in \mathbb{N}$. A *k-labeled graph* G is a finite graph in which k vertices, or less, are labeled with labels from $[k] = \{1,\dots,k\}$. We denote the class of k-labeled graphs by $\mathcal{G}_k$. The *k-connection* of two k-labeled graphs $G_1, G_2 \in \mathcal{G}_k$ is given by taking the disjoint union of G_1 and G_2 and identifying vertices with the same label. This produces a k-labeled graph $G = G_1G_2$. Note that k-connections are commutative.

2.1 Quantum graphs

A formal linear combination of a finite number of k-labeled graphs $F_i \in \mathcal{G}_k$ with coefficients from $\mathbb{R}$ is called a *k-labeled quantum graph*. $\mathcal{Q}_k$ denotes the set of k-labeled quantum graphs.

Let x, y be k-labeled quantum graphs: $x = \sum_{i=1}^n a_iF_i$, and $y = \sum_{i=1}^n b_iF_i$. Note that some of the coefficients may be zero. $\mathcal{Q}_k$ is an infinite dimensional vector space, with the operations: $x+y = (\sum_{i=1}^n a_iF_i)+(\sum_{i=1}^n b_iF_i) = \sum_{i=1}^n (a_i+b_i)F_i$, and $\alpha\cdot x = \sum_{i=1}^n (\alpha a_i)F_i$.

k-connections extend to k-labeled quantum graphs by $xy = \sum_{i,j=1}^n (a_ib_j)(F_iF_j)$. Any graph parameter f extends to k-labeled quantum graphs linearly: $f(x) = \sum_{i=1}^n a_if(F_i)$.

2.2 Equivalence relations for quantum graphs

The *k-connection matrix* $C(f,k)$ of a graph parameter $f : \mathcal{G} \to \mathcal{R}$ is a bi-infinite matrix over $\mathcal{R}$ whose rows and columns are labeled with k-labeled graphs, and its entry at the row labeled with G_1 and the column labeled with G_2 contains the value of f on G_1G_2:

$$C(f,k)_{G_1,G_2} = f(G_1G_2).$$

Given a connection matrix $C(f,k)$, we associate with a k-labeled graph $G \in \mathcal{G}_k$ the (infinite) row vector R_G^k appearing in the row labeled by G in $C(f,k)$. If k is clear from context we write R_G. Similarly, we associate an infinite row vector R_x with k-labeled quantum graphs

$x = \sum_{i=1}^{n} a_i F_i$, defined as $R_x = \sum_{i=1}^{n} a_i R_{F_i}$ where R_{F_i} is the row in $C(f,k)$ labeled by the k-labeled graph F_i.

We say $C(f,k)$ has *finite rank* if there are finitely many k-labeled graphs $\mathcal{B}_{C(f,k)} = \{B_1, \dots, B_n\}$ whose rows $\mathcal{R}_{C(f,k)} = \{R_{B_1}, \dots, R_{B_n}\}$ linearly span $C(f,k)$. Meaning, for any k-labeled graph G, there exists a linear combination of the rows in $\mathcal{R}_{C(f,k)}$ which equals the row vector R_G. We say that $C(f,k)$ has rank n and denote $r(f,k) = n$ if any set of less than n graphs does not linearly span $C(f,k)$.

The main result we use is the characterization of partition functions in terms of connection matrices. We do not need its complete power, so we state the relevant part:

▶ **Theorem 4** (Freedman, Lovász, Schrijver, [10]). *Let f be a graph parameter that is equal to $hom(-, H(\alpha,\beta))$ for some $H(\alpha,\beta)$ on q vertices. Then $r(f,k) \le q^k$ for all $k \ge 0$.*

The exact rank $r(f,k)$ was characterized in [15], but first we need some definitions. A weighted graph $H(\alpha,\beta)$ is said to be *twin-free* if β does not contain two separate rows that are identical to each other [3]. Let $H(\alpha,\beta)$ be a weighted graph on q vertices, and let $\mathrm{Aut}(H(\alpha,\beta))$ be the automorphism group of $H(\alpha,\beta)$. $\mathrm{Aut}(H(\alpha,\beta))$ acts on ordered k-tuples of vertices $[q]^k = \{\phi : [k] \to [q]\}$ by $(\sigma \circ \phi)(i) = \sigma(\phi(i))$ for $\sigma \in \mathrm{Aut}(H(\alpha,\beta))$. The *orbit* of ϕ is the set of ordered k-tuples ψ of vertices such that $\sigma \circ \phi = \psi$ for an automorphism $\sigma \in \mathrm{Aut}(H(\alpha,\beta))$. The *number of orbits* of $\mathrm{Aut}(H(\alpha,\beta))$ on $[q]^k$ is the number of different orbits for elements $\phi \in [q]^k$.

▶ **Theorem 5** (Lovász, [15]). *Let $f = hom(-, H(\alpha,\beta))$ for a twin-free weighted graph $H(\alpha,\beta)$ on q vertices. Then $r(f,k)$ is equal to the number of orbits of $\mathrm{Aut}(H(\alpha,\beta))$ on $[q]^k$ for all $k \ge 0$.*

We use the special case:

▶ **Corollary 6.** *Let $f = hom(-, H(\alpha,\beta))$ for a rigid twin-free weighted graph $H(\alpha,\beta)$ on q vertices. Then $r(f,k) = q^k$ for all $k \ge 0$.*

We define an equivalence relation $\equiv_{f,k}$ over $\mathcal{Q}_k$ where two k-labeled quantum graphs x and y are in the same equivalence class if and only if the infinite vectors R_x and R_y are identical: $x \equiv_{f,k} y \iff R_x^k = R_y^k$. Note that the set $\mathcal{Q}_k/f$ of equivalence classes of $\equiv_{f,k}$ is exactly the vector space $span(C(f,k))$ generated by linear combinations of rows in $C(f,k)$. k-connections extend to these vectors by: $R_x R_y = R_{xy}$.

Thus, if $r(f,k) = n$ with spanning rows $\mathcal{R}_{C(f,k)} = \{R_{B_1}, \dots, R_{B_n}\}$, they form a basis of $\mathcal{Q}_k/f = span(C(f,k))$. For brevity, we occasionally also refer to $\mathcal{B}_{C(f,k)}$ as a basis.

Let x be a k-labeled quantum graph whose equivalence class R_x is given as the linear combination $R_x = \sum_{i=1}^{n} \gamma_i R_{B_i}$. We call the column vector $\bar{c}_x = (\gamma_1, \dots, \gamma_n)^T$ the *coefficients vector* of x, or *representation* of x using $\mathcal{B}_{C(f,k)}$.

3 The learning algorithm in detail

In this section we present the learning algorithm in full detail. The commentary in this exposition foreshadows the arguments in Section 4, but otherwise validity is not considered here. We do not address complexity concerns in this section either, however, we reiterate

[3] If $H(\alpha,\beta)$ has twin vertices, they can be merged into one vertex by adding their vertex weights without changing the partition function. As the size of the target representation is the smallest possible, we assume all targets are twin-free.

for the sake of clarity that the algorithm runs on a Blum-Shub-Smale machine, [6, 5], over the reals. In such a machine, real numbers are treated as atomic objects; they are stored in single cells, and arithmetic operations are performed on them in a single step.

The objects the algorithm primarily works with are real matrices. In a context containing a basis $\mathcal{B}_{C(f,k)}$, we associate a real matrix A_x with each quantum graph x such that the following holds.

The coefficients vector $\bar{c}_{xy}$ of xy using $\mathcal{B}_{C(f,k)}$ is given by $A_x\bar{c}_y$. (*)

This device, as we will see in Section 4, will allow the algorithm to search for, and find, special quantum graphs that provide a translation of the answers of VALUE and EQUIVALENT queries into a hypothesis.

As mentioned earlier, Algorithm 1 maintains a matrix M which is a submatrix of $C(f,1)$. In each iteration the algorithm generates a hypothesis $h = (\alpha^{(h)}, \beta^{(h)})$ using M, and queries the teacher for equivalence between h and the target f. If the hypothesis is correct, the algorithm returns h, otherwise it augments M with a 1-labeled version of the counterexample, and moves on to the next iteration.

▶ **Remark 7.** *Strictly speaking, the teacher may be asked* VALUE *queries on (unlabeled) graphs, however, we freely write* VALUE(G) *for k-labeled graphs $G \in \mathcal{G}_k$. Additionally, the algorithm will need to know the value of the target on some quantum graphs. Since any graph parameter extend to quantum graphs linearly, for a quantum graph $x = \sum_{i=1}^{n} a_i F_i$ we write* VALUE(x) *as shorthand for $\sum_{i=1}^{n} a_i \cdot$* VALUE(F_i) *throughout the presentation.*

Incorporating counterexamples

The objective is to keep a non-singular submatrix M of $C(f,1)$. The first 1-labeled graph B_1 with which M is augmented is some arbitrarily chosen 1-labeled graph.

Upon receiving a B_n graph as counterexample, the 1-label is arbitrarily assigned to one of its vertices, making it a 1-labeled graph. Then augment M with(B_n) adds a row and a column to M labeled with the (now) 1-labeled graph B_n, and fills their entries with the values $f(B_nB_i) = f(B_iB_n)$, for $i \in [n]$, using VALUE queries.

The other functions are slightly more complex.

Finding an idempotent basis

The function find basis, given in pseudo-code as Algorithm 2, receives as input the matrix M. For reasons which will become apparent later, we are interested in finding a certain (idempotent) basis of the linear space generated by the rows of $C(f,1)$. For this purpose, in its first part find basis iteratively, over $k = 1, \ldots, n$, computes the entries of matrices A_x as in (*), where x are B_i, $i \in [n]$, by solving multiple systems $M\mathbf{x} = \mathbf{b}$ of linear equations, and using the solutions Γ of those systems to fill the entries of the matrices A_{B_i}, where the (k,j) entry of A_{B_i} is $\gamma^{ij}(k)$. Let p_i, $i \in [n]$ be those quantum graphs for which A_{p_i} is the $n \times n$ matrix with the value 1 in the entry (i,i) and zero in all other entries. Note that the matrices A_{p_i}, $i \in [n]$ are linearly independent. We will see that p_i, $i \in [n]$ are the idempotent basis, now we wish to find their representation using B_i, $i \in [n]$.

For $i \in [n]$, the representation $\bar{c}_{p_i}$ of the basic idempotent p_i using the basis elements $B_i, i \in [n]$ is found by solving a system $A\mathbf{X} = A_{p_i}$ of linear *matrix equations*, where A is a block matrix whose blocks are the matrices A_{B_i}, $i \in [n]$. Each solution is added to Δ.

Finally, find basis outputs the set Δ of these representations $\bar{c}_{p_i}$, $i \in [n]$. Then we have that $R_{p_i} = \sum_{k=1}^{n} \bar{c}_{p_i}(k) R_{B_k}$ where $\bar{c}_{p_i}$ is the coefficients vector of p_i using B_i, $i \in [n]$. The

Algorithm 2 find basis function

```
1: $\Gamma = \emptyset$
2: for each $i, j \in [n]$ do
3:     for $k = 1, \ldots, n$ do
4:         $\mathbf{b}(k) = \textsc{value}(B_i B_j B_k)$
5:     end for
6:     $\boldsymbol{\gamma}^{ij} =$ solve linear system$(M\mathbf{x} = \mathbf{b})$
7:     $\Gamma = \Gamma \cup \{\boldsymbol{\gamma}^{ij}\}$
8: end for
9: for $i \in [n]$ do
10:     $A_{B_i} =$ fill matrix$(i, \Gamma)$
11:     $A =$ add block$(A, i, A_{B_i})$          ▷ $A$ is a block matrix with $A_{B_i}$ on its $i$th block
12: end for
13: $\Delta = \emptyset$
14: for $i \in [n]$ do
15:     $\bar{c}_{p_i} =$ solve linear matrix system$(A\mathbf{X} = A_{p_i})$
16:     $\Delta = \Delta \cup \{\bar{c}_{p_i}\}$
17: end for
18: return $\Delta$
```

representations $\bar{c}_{p_i} \in \Delta$ of the elements p_i, $i \in [n]$, are what will provide a translation from results of VALUE queries to weights.

Generating a hypothesis

The function generate hypothesis, given in pseudo-code as Algorithm 3, receives as input the representations $\bar{c}_{p_i}$ of the 1-labeled quantum graphs p_i, $i \in [n]$, which it uses to find the entries of the vertex weights vector $\alpha^{(h)}$ directly through VALUE queries.

Then generate hypothesis finds the 2-labeled analogues of these 1-labeled quantum graphs. Those 2-labeled analogues form a basis of of $\mathcal{Q}_2/f$.

Denote by K_2 the 2-labeled graph composed of a single edge with both vertices labeled. Next, generate hypothesis finds the representation of $R^2_{K_2}$, that is the row labeled with K_2 in $C(f,2)$, using the basis $\mathcal{R}_{C(f,2)}$. We find the representation of this specific graph K_2 as the coefficients in $\bar{c}_{K_2}$ constitute the entries of the edge weights matrix $\beta^{(h)}$ (see Section 4).

This representation is found by solving a linear system of equations, similarly to how find basis uses solve linear system, but here we use the diagonal matrix N whose entries correspond to the elements of $\mathcal{B}_{C(f,2)}$.

The solution of said system, i.e., the coefficients vector $\bar{c}_{K_2}$ of K_2, is used to fill the edge weights matrix $\beta^{(h)}$. If needed, $\beta^{(h)}$ is made twin-free by contracting the twin vertices into one and summing their weights in $\alpha^{(h)}$.

Finally, generate hypothesis returns the hypothesis $h = (\alpha^{(h)}, \beta^{(h)})$ as output.

▶ **Remark 8** (Algorithm 3). *Let q_i be the 1-labeled quantum graph p_i interpreted as a 2-labeled quantum graph, and let q_j be p_j with the labels of its components renamed to 2, and also interpreted as a 2-labeled quantum graph. The result of $p_i \otimes p_j$ is the 2-labeled quantum graph $q_i \sqcup_2 q_j$.*

Algorithm 3 generate hypothesis function

```
1: for each $i \in [n]$ do
2:     $\alpha^{(h)}(i) = \textsc{value}(p_i)$
3: end for
4: $N = 0^{n^2 \times n^2}$                          ▷ $N$ is a zero matrix of dimensions $n^2 \times n^2$.
5: for $i = 1, \ldots, n$ do
6:     for $j = 1, \ldots, n$ do
7:         $p_{ij} = p_i \otimes p_j$                 ▷ See Remark 8.
8:         $N_{p_{ij}, p_{ij}} = \textsc{value}(p_{ij} p_{ij})$
9:         $\mathbf{b}(i_j) = \textsc{value}(K_2\, p_{ij})$
10:    end for
11: end for
12: $\beta^{(h)} =$ solve linear system$(N\mathbf{x} = \mathbf{b})$
13: make twin-free$(\alpha^{(h)}, \beta^{(h)})$
14: $h = (\alpha^{(h)}, \beta^{(h)})$
15: return $h$
```

4 Validity and complexity

As stated earlier, a class of functions is exactly learnable if there is a learner that for each target function f, outputs a hypothesis h such that f and h identify on all inputs, and does so in time polynomial in the size of a shortest representation of f and the size of a largest counterexample returned by the teacher. The proof of Theorem 2 argues that Algorithm 1 is such a learner for the class of rigid partition functions, through Theorem 9, which proves validity, and Theorem 22, which proves the complexity constraints are met.

To prove validity, we first state existing results on properties of graph algebras induced by partition functions, then show, through somewhat technical algebraic manipulations, how our algorithm successfully exploits these properties to generate hypotheses. We then show our algorithm eventually terminates with a correct hypothesis.

For the rest of the section, let $H(\alpha, \beta)$ be a rigid twin-free weighted graph on q vertices, and denote $f = \mathsf{hom}(-, H(\alpha, \beta))$.

▶ **Theorem 9.** *Given access to a teacher for f, Algorithm 1 outputs a hypothesis h such that $f(G) = h(G)$ for all graphs $G \in \mathcal{G}$.*

The proof of the theorem follows from arguing that:

▶ **Theorem 10.** *If M is of rank q, then* generate hypothesis *outputs a correct hypothesis.*

and that the rank of M is incremented with every counterexample:

▶ **Theorem 11.** *In the n^{th} iteration of Algorithm 1 on f, M has rank n.*

First we confirm the hypotheses Algorithm 1 generates are indeed in the class of graph parameters we are trying to learn, namely, rigid partition functions $\mathsf{hom}(-, H(\alpha, \beta))$ for twin-free weighted graphs $H(\alpha, \beta)$.

Given Theorem 11, for the hypothesis h returned in the n^{th} iteration, the rank of $C(h, 1)$ is at least n, since M is a submatrix of $C(h, 1)$. Thus, from Theorem 5, h cannot have proper automorphisms, as it would imply that the rank of $C(h, 1) < n$. The fact that h is twin-free is immediate from the construction in generate hypothesis.

4.1 From the idempotent bases to the weights – proof of Theorem 10

Let $\mathcal{Q}_k/f$ be of finite dimension n. The *idempotent basis* $p_1, \ldots, p_n$ of $\mathcal{Q}_k/f$ consists of those k-labeled quantum graphs p_i for which $p_i p_i \equiv_{f,k} p_i$ and $p_i p_j \equiv_{f,k} 0$ for $i, j \in [n]$, $i \neq j$. Recall how find basis found those 1-labeled quantum graphs p_i, $i \in [n]$ whose matrices A_{p_i} behaved in this way.

In our setting of rigid twin-free weighted graphs, by [14, Chapter 6], we have that if $p_1, \ldots, p_q$ are the idempotent basis of Q_1/f, then the idempotent basis of $\mathcal{Q}_2/f$ is given by $p_i \otimes p_j$, $i, j \in [q]$. These are the 2-labeled analogues mentioned in the description of generate hypothesis.

Furthermore by [14, Chapter 6], the vertex weights α of H are given by $\alpha(i) = f(p_i)$, $i \in [q]$, and if the representation of K_2 using $p_i \otimes p_j$, $i, j \in [q]$ is $\sum_{i,j \in [q]} \beta_{ij}(p_i \otimes p_j)$, then the edge weights matrix β is given by $\beta_{i,j} = \beta_{ij}$.

Equipped with these useful facts, we show that:

▶ **Lemma 12.** *If M is of rank q, then find basis outputs the idempotent basis of $\mathcal{Q}_1/f$.*

Then obtain Theorem 10 by showing how, if generate hypothesis receives the idempotent basis of $\mathcal{Q}_1/f$ as input, it outputs a correct hypothesis.

Finding the idempotent basis – proof of Lemma 12

Recall that in the presence of a basis $\mathcal{B}_{C(f,k)}$ we associate a real matrix A_x with each quantum graph x such that the following holds.

The coefficients vector $\bar{c}_{xy}$ of xy using $\mathcal{B}_{C(f,k)}$ is given by $A_x \bar{c}_y$.

For reasons we cannot list here, A_x will be diagonal. Let $B_i, B_j \in \mathcal{B}_{C(f,1)}$, and denote by $\sum_{k=1}^{n} \gamma_k^{i,j} R_{B_k}$ the representation of the row $R_{B_i B_j}$ using $\mathcal{R}_{C(f,1)}$, i.e., the row in $C(f,1)$ labeled with the graph resulting from the product $B_i B_j$.

▶ **Claim 13.** *Let x be some 1-labeled quantum graph such that $R_x = \sum_{i=1}^{n} a_i R_{B_i}$. The matrix A_x is given by $(A_x)_{\ell,m} = \sum_{i=1}^{n} a_i \gamma_\ell^{im}$.*

Note that for a basis graph $B_k \in \mathcal{B}_{C(f,1)}$, we have that $(A_{B_k})_{i,j} = \gamma_i^{k,j}$. The proof of this claim appears in the appendix of [13].

▶ **Proposition 14.** *The matrices $A_{B_1}, \ldots, A_{B_n}$ of the graphs in $\mathcal{B}_{C(f,1)}$ are linearly independent and span all matrices of the form A_x for a quantum graph x.*

If we know what are the matrices $A_{p_1}, \ldots, A_{p_n}$ of the idempotent basis $p_1, \ldots, p_n$, we can find their representation using $A_{B_1}, \ldots, A_{B_n}$ by solving systems of linear matrix equations. Then, given a representation $A_{p_i} = \sum_{k=1}^{n} \delta_k^{(i)} A_{B_k}$, we will have the representation of the basic idempotents using $\mathcal{B}_{C(f,1)}$ as $p_i = \sum_{k=1}^{n} \delta_k^{(i)} B_k$.

The definitions of A_x and idempotence lead to the observation that for idempotent basics p_i, p_j, it holds that $A_{p_i} A_{p_i} = A_{p_i}$ and $A_{p_i} A_{p_j} = 0$. From Corollary 6 we know the dimension of $\mathcal{Q}_1/f$ is q, so we conclude:

▶ **Proposition 15.** *The idempotent basis for $\mathcal{Q}_1/f$ consists of the quantum graphs p_i, $i \in [q]$ for which A_{p_i} is the $q \times q$ matrix with the value 1 in the entry (i,i) and zero in all other entries. That is,*

$$A_{p_i}(k,j) = \begin{cases} 1, & \text{if } (k,j) = (i,i) \\ 0, & \text{otherwise} \end{cases}$$

As find basis solves the systems of linear matrix equations for these matrices, it remains to show that find basis correctly computes the matrices A_{B_i}, $i \in [q]$.

Since M is of full rank, the representations $\sum_{k=1}^{n} \gamma_k^{i,j} R_{B_k}$ of graphs B_iB_j, $i,j \in [q]$ using $\mathcal{B}_{C(f,1)}$ are correctly computed by the solve linear system calls. And as noted before, the coefficients $\gamma_k^{i,j}$ are the entries of the matrices A_{B_i}, $i \in [q]$. Thus they indeed are correctly computed, and we have Lemma 12.

Since generate hypothesis directly queries the teacher for the values of $\alpha^{(h)}$, we have:

▶ **Corollary 16.** *If M is of rank q, then* generate hypothesis *outputs a correct vertex weights vector $\alpha^{(h)}$.*

It remains to show this is true also for the edge weights:

▶ **Proposition 17.** *If M is of rank q, then* generate hypothesis *outputs a correct edge weights matrix $\beta^{(h)}$.*

Proof. As $p_{ij} = p_i \otimes p_j$, $i,j \in [q]$ are the idempotent basis for $\mathcal{Q}_2/f$ we have that $p_{ij}p_{ij} \not\equiv_{f,2} 0$, so the matrix N is a diagonal matrix of full rank, and solve linear system indeed finds the representation of K_2 using p_{ij}, $i,j \in [q]$. ◀

From Corollary 16 and Proposition 17 we have Theorem 10.

Now we show that Algorithm 1 reaches that point in the first place.

4.2 Augmentation results in larger rank – proof of Theorem 11

Theorem 11 is proved using the fact that A_x are linearly independent for k-labeled quantum graphs which are not equivalent in $\equiv_{f,k}$.

▶ **Lemma 18.** *In the n^{th} iteration of Algorithm 1, if the teacher returns a counterexample x, then R_x is not spanned by $R_{B_1}, \ldots, R_{B_n}$ where $B_1, \ldots, B_n$ are the graphs associated with the rows and columns of M.*

Proof. If $n = 1$, M has rank n. Now let M have rank n.

For contradiction, assume that $R_x = \sum_{i=1}^{n} a_i R_{B_i}$. Then $x \equiv_{f,1} \sum_{i=1}^{n} a_i B_i$ and we have that $\mathsf{hom}(x,H) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i,H)$ for the target graph H. Denote by $h^{(n)}$ the hypothesis generated in this iteration. If x is a counterexample, it must hold that

$$\mathsf{hom}(x, h^{(n)}) \neq \mathsf{hom}(x,H) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i,H)$$

The solution of the system of equations for $\mathbf{b}_x$ would give

$$\mathsf{hom}(x, h^{(n)}) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i, h^{(n)}) = \sum_{i=1}^{n} a_i \mathsf{hom}(B_i,H)$$

So we conclude that $\sum_{i=1}^{n} a_i \mathsf{hom}(B_i, h^{(n)}) \neq \sum_{i=1}^{n} a_i \mathsf{hom}(B_i,H)$.

Since M is of full rank, one can solve a system of linear equations using M for $\mathbf{b}_x$ defined as $\mathbf{b}_x(k) = \textsc{value}(xB_k)$, $k \in [n]$. Now recall that the matrix M contains *correct* values $\mathsf{hom}(B_iB_j, H(\alpha,\beta))$, as it was augmented using $\textsc{value}$ queries, therefore M is a submatrix of $C(f,1)$. Thus the coefficients of the solution $\mathbf{a}$ of $M\mathbf{a} = \mathbf{b}_x$ equal a_i, $i \in [k]$, and we reach a contradiction. Therefore we conclude $x \not\equiv_{f,1} \sum_{i=1}^{n} a_i B_i$ and its row R_x is linearly independent from $R_{B_1}, \ldots, R_{B_n}$. ◀

This also implies that the matrix A_x associated with x is not spanned by $A_{B_1}, \ldots, A_{B_n}$. Therefore the submatrix of $C(f, 1)$ composed of the entries of the rows and columns of $B_1, \ldots, B_n, x$ is of full rank $n+1$. This is exactly the matrix M augmented with x, and we have Theorem 11. Combining this with Corollary 6, we have:

▶ **Corollary 19.** *Let f be a rigid partition function of a twin-free weighted graph on q vertices. Then Algorithm 1 terminates in q iterations.*

4.3 Complexity analysis

As the algorithm runs on a Blum-Shub-Smale machine for the reals and mostly solves systems of linear equations, it is not difficult to show that it runs in time polynomial in the size of target and the largest counterexample. First we observe:

▶ **Proposition 20.** *Let $G_1, G_2 \in \mathcal{G}_1$. Then $G_1 G_2$ can be computed in time $O(\mathrm{poly}(|G_1|, |G_2|))$.*

▶ **Remark 21.** *B_1 is of fixed size, and all other B_i, $i = 2, \ldots, n$, used in Algorithm 1 are counterexamples provided by the teacher, therefore they are all of size polynomial in the size $|x|$ of the graph x.*

▶ **Theorem 22.** *Let $H(\alpha, \beta)$ be a rigid twin-free weighted graph on q vertices and denote $f = \mathsf{hom}(-, H(\alpha, \beta))$. Given access to a teacher for f, Algorithm 1 terminates in time $O(\mathrm{poly}(q, |x|))$, where $|x|$ is the size of the largest counterexample provided by the teacher.*

Proof. From Corollary 19 it is enough to show that each iteration of Algorithm 1 does not take too long (Lemma 23). ◀

▶ **Lemma 23** ([13]). *In the n^{th} iteration of Algorithm 1, augment M, find basis, and generate hypothesis all run in time $O(\mathrm{poly}(n, |x|))$.*

▶ Remark. We note that, from [14, Theorem 6.45], the counterexamples provided by the teacher may be chosen to be of size at most $2(1+q^2)q^6$ where q is the size of the target weighted graph.

5 Conclusion and future work

This paper presented an adaptation of the exact model of learning of Angluin, [1], to the context of graph parameters f representable as partition functions of weighted graphs $H(\alpha, \beta)$. We presented an exact learning algorithm for the class of *rigid* partition functions defined by twin-free $H(\alpha, \beta)$.

If a weighted graph has proper automorphisms, its connection matrices $C(f, k)$ may have rank smaller than q^k. In this case, the translation from query results to a weighted graph would involve the construction of a submatrix of $C(f, k)$ for a sufficiently large k, and then find an idempotent basis for $\mathcal{Q}_{k+1}/f$. We will study the learnability of non-rigid partition functions in a sequel to this paper.

Theorems similar to Theorem 4 have been proved for variants of partition functions and connection matrices, [16, 8, 17, 18]. It seems reasonable to us that similar exact learning algorithms exist for these settings, but it is unclear how to modify our proofs here for this purpose.

Acknowledgements. We thank M. Jerrum and M. Hermann for their valuable remarks while listening to an early version of the introduction of the paper, and A. Schrijver for his interest and encouragement. We also thank two anonymous referees for their helpful remarks.

References

1 D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.
2 D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.
3 A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM (JACM)*, 47(3):506–530, 2000.
4 G. D. Birkhoff. A determinant formula for the number of ways of coloring a map. *Annals of Mathematics*, 14:42–46, 1912.
5 L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation*. Springer Science & Business Media, 2012.
6 L. Blum, M. Shub, S. Smale, et al. 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.
7 B. Bollobás. *Modern Graph Theory*. Springer, 1999.
8 J. Draisma, D. C. Gijswijt, L. Lovász, G. Regts, and A. Schrijver. Characterizing partition functions of the vertex model. *Journal of Algebra*, 350(1):197–206, 2012.
9 P. Erdős and A. Rényi. Asymmetric graphs. *Acta Mathematica Hungarica*, 14(3-4):295–315, 1963.
10 M. Freedman, L. Lovász, and A. Schrijver. Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society*, 20(1):37–51, 2007.
11 A. Habrard and J. Oncina. Learning multiplicity tree automata. In *Grammatical Inference: Algorithms and Applications*, pages 268–280. Springer, 2006.
12 J. Kötters. Almost all graphs are rigid—revisited. *Discrete Mathematics*, 309(17):5420–5424, 2009.
13 N. Labai and J. A. Makowsky. On the exact learnability of graph parameters: The case of partition functions. *arXiv preprint arXiv:1606.04056*, 2016. URL: `http://arxiv.org/abs/1606.04056`.
14 L. Lovász. *Large Networks and Graph Limits*, volume 60 of *Colloquium Publications*. AMS, 2012.
15 L. Lovász. The rank of connection matrices and the dimension of graph algebras. *European Journal of Combinatorics*, 27(6):962–970, 2006.
16 A. Schrijver. Graph invariants in the spin model. *J. Comb. Theory, Ser. B*, 99(2):502–511, 2009.
17 A. Schrijver. Characterizing partition functions of the spin model by rank growth. *Indagationes Mathematicae*, 24.4:1018–1023, 2013.
18 A. Schrijver. Characterizing partition functions of the edge-coloring model by rank growth. *Journal of Combinatorial Theory, Series A*, 136:164–173, 2015.
19 A. D. Sokal. The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In *Survey in Combinatorics, 2005*, volume 327 of *London Mathematical Society Lecture Notes*, pages 173–226, 2005.

A Preliminary Investigation of Satisfiability Problems Not Harder Than 1-In-3-SAT

Victor Lagerkvist[1] **and Biman Roy**[2]

1 **Institut für Algebra, TU Dresden, Dresden, Germany**
`victor.lagerqvist@tu-dresden.de`

2 **Department of Computer and Information Science, Linköping University, Linköping, Sweden**
`biman.roy@liu.se`

Abstract

The *parameterized satisfiability problem* over a set of Boolean relations Γ ($\mathrm{SAT}(\Gamma)$) is the problem of determining whether a conjunctive formula over Γ has at least one model. Due to Schaefer's dichotomy theorem the computational complexity of $\mathrm{SAT}(\Gamma)$, modulo polynomial-time reductions, has been completely determined: $\mathrm{SAT}(\Gamma)$ is always either tractable or NP-complete. More recently, the problem of studying the relationship between the complexity of the NP-complete cases of $\mathrm{SAT}(\Gamma)$ with restricted notions of reductions has attracted attention. For example, Impagliazzo et al. studied the complexity of k-SAT and proved that the worst-case time complexity increases infinitely often for larger values of k, unless 3-SAT is solvable in subexponential time. In a similar line of research Jonsson et al. studied the complexity of $\mathrm{SAT}(\Gamma)$ with algebraic tools borrowed from clone theory and proved that there exists an NP-complete problem $\mathrm{SAT}(R^{\neq\neq\neq01}_{1/3})$ such that there cannot exist any NP-complete $\mathrm{SAT}(\Gamma)$ problem with strictly lower worst-case time complexity: the *easiest NP-complete* $\mathrm{SAT}(\Gamma)$ *problem*. In this paper we are interested in classifying the NP-complete $\mathrm{SAT}(\Gamma)$ problems whose worst-case time complexity is lower than 1-in-3-SAT but higher than the easiest problem $\mathrm{SAT}(R^{\neq\neq\neq01}_{1/3})$. Recently it was conjectured that there only exists three satisfiability problems of this form. We prove that this conjecture does not hold and that there is an infinite number of such $\mathrm{SAT}(\Gamma)$ problems. In the process we determine several algebraic properties of 1-in-3-SAT and related problems, which could be of independent interest for constructing exponential-time algorithms.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.0 Discrete Mathematics General

Keywords and phrases Clone Theory, Universal Algebra, Satisfiability Problems

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.64

1 Introduction

The *parameterized satisfiability problem* ($\mathrm{SAT}(\Gamma)$) is the computational decision problem of, given a conjunctive formula over a constraint language Γ, determining whether this formula is satisfiable. Some notable examples of problems that can be formulated as $\mathrm{SAT}(\Gamma)$ problems include 1-in-3-SAT, k-SAT, SAT, and not-all-equal-SAT. For example, if we let $R_{1/3} = \{(0,0,1),(0,1,0),(1,0,0)\}$ then $\mathrm{SAT}(\{R_{1/3}\})$ can be seen as an alternative formulation of the monotone 1-in-3-SAT problem, i.e., 1-in-3-SAT without negation. Hence, $\mathrm{SAT}(\Gamma)$ is in general NP-complete. It is also known that $\mathrm{SAT}(\Gamma)$ is either tractable, i.e., solvable in polynomial time, or NP-complete, for all choices of Γ [25]. Assume that we instead are interested in a more fine-grained analysis of the worst-case time complexity of all

$$\begin{tikzcd}
\langle\{R_{1/3}\}\rangle_{\nexists} \\
\langle\{R^{01}_{1/3}\}\rangle_{\nexists} \arrow[u] \\
\langle\{R^{\neq 01}_{1/3}\}\rangle_{\nexists} \arrow[u] \\
\langle\{R^{\neq\neq 01}_{1/3}\}\rangle_{\nexists} \arrow[u] \\
\langle\{R^{\neq\neq\neq 01}_{1/3}\}\rangle_{\nexists} \arrow[u]
\end{tikzcd}$$

Figure 1 The conjectured structure of weak partial co-clones below $\langle\{R_{1/3}\}\rangle_{\nexists}$. A directed arrow $A \rightarrow B$ means that $A \subset B$.

NP-complete SAT(Γ) problems. Clearly, the fact that two problems SAT(Γ) and SAT(Δ) are both NP-complete does not reveal a great amount of information about their respective worst case time complexity, except that they are both unlikely to be solvable in polynomial time. For example, the monotone 1-in-3-SAT problem is solvable in $O(1.0984^n)$ time [30], where n denotes the number of variables in a given instance. On the other hand, 3-SAT is only known to be solvable in $O(1.308^n)$ time [11], and more generally it is known that the worst-case time complexity of k-SAT increases infinitely often for increasing values of k [13] – assuming 3-SAT is not solvable in $O(c^n)$ time for arbitrary $c > 1$. Hence, the family of NP-complete SAT(Γ) problems seems to contain members with wildly distinct worst-case time complexity, and it is safe to say that we currently cannot provide a complete explanation of this phenomena. More generally, say that SAT(Γ) is *easier* than SAT(Δ) if SAT(Γ) is solvable in $O(c^n)$ time whenever SAT(Δ) is solvable in $O(c^n)$ time, where n denotes the number of variables in a given instance. In symbols, we denote this by $\text{SAT}(\Delta) \leq \text{SAT}(\Gamma)$. Jonsson et al. studied the complexity of SAT(·) viz a viz the ordering $\leq$ using *partial clone theory* [15]. The details of this approach is explained in greater detail in Section 2, but for the moment let us be content with the fact that there exists a lattice $\mathcal{X}$ such that every constraint language Γ can be mapped to an element $\langle\Gamma\rangle_{\nexists} \in \mathcal{X}$, such that $\text{SAT}(\Gamma) \leq \text{SAT}(\Delta)$ if $\langle\Gamma\rangle_{\nexists} \subseteq \langle\Delta\rangle_{\nexists}$. An element $\langle\Gamma\rangle_{\nexists} \in \mathcal{X}$ is usually referred to as a *weak system*, or a *weak partial co-clone*, and is a well-studied relational algebra known to consist of all relations definable by conjunctive logical formulas over Γ [23]. Hence, the lattice of weak partial co-clones can be used to compare SAT(Γ) problems with respect to worst-case time complexity. With the help of this algebraic approach Jonsson et al. gave a classification of the minimal element $\langle\{R^{\neq\neq\neq 01}_{1/3}\}\rangle_{\nexists}$ of this lattice and proved that $\text{SAT}(\{R^{\neq\neq\neq 01}_{1/3}\})$ results in the *easiest NP-complete* SAT(·) problem [15].

In this paper we continue the classification of NP-complete SAT(Γ) problems that in a certain precise sense are small elements in the ordering $\leq$. More specifically, we are interested in determining the structure of constraint languages resulting in NP-complete SAT(·) problems which are not computationally harder than monotone 1-in-3-SAT. In symbols, this can be rephrased as determining all constraint languages Γ such that $\text{SAT}(\{R^{\neq\neq\neq 01}_{1/3}\}) \leq \text{SAT}(\Gamma) \leq \text{SAT}(\{R_{1/3}\})$, or, in the language of clone theory, determining all constraint languages Γ satisfying $\langle\{R^{\neq\neq\neq 01}_{1/3}\}\rangle_{\nexists} \subset \langle\Gamma\rangle_{\nexists} \subset \langle\{R_{1/3}\}\rangle_{\nexists}$. We begin by recapitulating the necessary technical prerequisites in Section 2, and give a brief introduction to the algebraic approach for studying the complexity of satisfiability problems. In Section 3 we introduce novel methods for better understanding the structure of algebras of the form $\langle\Gamma\rangle_{\nexists}$, and in particular the structure of $\langle\{R_{1/3}\}\rangle_{\nexists}$. This classification is then used in Section 4 where we give a preliminary description of the satisfiability problems below $\text{SAT}(\{R_{1/3}\})$ in the ordering $\leq$. We prove that this is a rich and complicated structure and that the cardinality of the set $\{\langle\Gamma\rangle_{\not\exists} \mid \langle\{R^{\neq\neq\neq01}_{1/3}\}\rangle_{\not\exists} \subset \langle\Gamma\rangle_{\not\exists} \subset \langle\{R_{1/3}\}\rangle_{\not\exists}\}$ is at least countably infinite. We remark that this contradicts a recent conjecture that this set consists of only three elements [20]. See Figure 1 for a visualization of the conjectured structure between $\langle\{R_{1/3}\}\rangle_{\not\exists}$ and $\langle\{R^{\neq\neq\neq01}_{1/3}\}\rangle_{\not\exists}$, and Section 2.4 for definitions of the involved relations.

From an algebraical point of view our results are a natural investigation of the largely unexplored lattice of weak partial co-clones. We remark that weak partial co-clones are useful not only for studying the exact complexity of problems [16, 19], but also for complexity classifications of optimisation problems and non-standard logical reasoning problems [3, 4, 27]. So far one of the limiting factors of this approach is the fact that very little is known of the relationship between weak partial co-clones and their dual objects, *partial polymorphisms*, which is in stark contrast to the status of the currently flourishing research program of classifying finite domain constraint satisfaction problems by properties of polymorphisms [2]. Similar observations have been made by for example Börner et al. [7], by Schölzel [28], and by Bulatov in the context of counting problems [8].

From a more pragmatic point of view, our results show that even for extremely simple constraint languages such as $\{R_{1/3}\}$, trying to fully characterize $\text{SAT}(\Gamma)$ problems with a lower worst-case time complexity is an extremely difficult task. However, as we discuss in Section 5, even though we cannot hope to achieve a complete understanding of the complexity of $\text{SAT}(\{R_{1/3}\})$ with our algebraic approach, we believe that similar studies are likely to open up new possibilities for studying the complexity of $\text{SAT}(\{R_{1/3}\})$ and related problems.

2 Preliminaries

In this section we briefly review some basic concepts that will be needed later on, starting with a formal definition of the parameterized $\text{SAT}(\cdot)$ problem and ending with universal algebra and partial clone theory.

2.1 The Parameterized SAT(·) Problem

Let $\mathbb{B} = \{0, 1\}$ and let $BR = \bigcup_{i=1} \mathbb{B}^i$ denote the set of all Boolean relations. Given $R \in \mathbb{B}^k$ we let $\text{ar}(R) = k$. A *constraint language* is a set of relations $\Gamma \subseteq BR$. The *parameterized satisfiability problem* over a constraint language Γ ($\text{SAT}(\Gamma)$) is defined as follows.

INSTANCE: A set V of variables and a set C of constraint applications $R(v_1, \ldots, v_k)$ where $R \in \Gamma$, $\text{ar}(R) = k$, and $v_1, \ldots, v_k \in V$.
QUESTION: Is there a function $f : V \to \mathbb{B}$ such that $(f(v_1), \ldots, f(v_k)) \in R$ for each $R(v_1, \ldots, v_k)$ in C?

When $\Gamma = \{R\}$ we typically write $\text{SAT}(R)$ instead of $\text{SAT}(\{R\})$. As an example, if we let $R_{1/3} = \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}$ then the problem $\text{SAT}(R_{1/3})$ can be seen as an alternative formulation of monotone 1-in-3-SAT, i.e., the 1-in-3-SAT problem without negation.

2.2 Polymorphisms, Clones and Co-Clones

A Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ is said to *preserve* a k-ary Boolean relation R if for every $t_1, \ldots, t_n \in R$ it holds that $(f(t_1[1], \ldots, t_n[1]), \ldots, f(t_1[k], \ldots, t_n[k])) \in R$. Here, $t_i[j]$ denotes the j-th element of the tuple t_i. If f preserves R we say that f is a *polymorphism* of R, and similarly we say that f is a polymorphism of a constraint language Γ if it preserves each relation in Γ. Given a constraint language Γ we let $\mathrm{Pol}(\Gamma)$ denote the set of all polymorphisms of Γ. Sets of the form $\mathrm{Pol}(\Gamma)$ are usually referred to as *clones* and it is well-known that clones are (1) closed under functional composition and (2) contain all functions which projects one of its arguments. To be a bit more precise, the first condition means that if $f, g_1, \ldots, g_m \in \mathrm{Pol}(\Gamma)$, where the f has arity m and the functions $g_1, \ldots, g_m$ all have the same arity n, then the *composition* $f \circ (g_1, \ldots, g_m)$, the function defined as $f \circ (g_1, \ldots, g_m)(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$ for all $x_1, \ldots, x_n \in \mathbb{B}$, is included in $\mathrm{Pol}(\Gamma)$. The second condition means that $\mathrm{Pol}(\Gamma)$ for each n and each $1 \leq i \leq n$ contains every function π^n_i defined as $\pi^n_i(x_1, \ldots, x_i, \ldots, x_n) = x_i$. Functions of the form π^n_i are called *projection functions*. We let $\Pi_{\mathbb{B}}$ denote the set of all Boolean projection functions.

There is also a similar notion to clones on the relational side. Say that a k-ary relation R has a *primitive positive definition* (p.p. definition) over a constraint language Γ if there exists a conjunctive formula over k variables $x_1, \ldots, x_k$ over Γ, possibly making use of existential quantification and the equality relation $\mathrm{Eq} = \{(0,0), (1,1)\}$, such that R is the set of models of this formula. In symbols, we denote such a p.p. definition as $R(x_1, \ldots, x_k) \equiv \exists y_1, \ldots, y_{k'} \,.\, R_1(\mathbf{x_1}) \wedge \ldots \wedge R_m(\mathbf{x_m})$, where each $R_i \in \Gamma \cup \{\mathrm{Eq}\}$ and each $\mathbf{x_i}$ is an $\mathrm{ar}(R_i)$-ary tuple of variables over $x_1, \ldots, x_k,\ y_1, \ldots, y_{k'}$. If we let $\langle \Gamma \rangle$ be the smallest set of relations containing Γ which is closed under p.p. definitions we obtain a *relational clone*, or a *co-clone*. The relationship between clones and co-clones is given in the following theorem.

▶ **Theorem 1** ([5, 6, 9]). *Let Γ and Γ' be two constraint languages. Then $\Gamma \subseteq \langle \Gamma' \rangle$ if and only if $\mathrm{Pol}(\Gamma') \subseteq \mathrm{Pol}(\Gamma)$.*

This inverse relationship between two closure operators is in general known as a *Galois connection*, and using Theorem 1 it is not difficult to prove the following result.

▶ **Theorem 2** ([14]). *Let Γ and Γ' be two finite constraint languages. If $\mathrm{Pol}(\Gamma') \subseteq \mathrm{Pol}(\Gamma)$, then $\mathrm{SAT}(\Gamma)$ is polynomial-time many-one reducible to $\mathrm{SAT}(\Gamma')$.*

Hence, the clone of a constraint language determines the complexity of a satisfiability problem up to polynomial time reductions. Unfortunately, as noted in Section 1, the mere fact that two $\mathrm{SAT}(\cdot)$ problems are polynomial-time equivalent does not offer any insight into their worst-case time complexity. To study this we need a more fine-grained algebra, which in our case consists of *partial functions* instead of total functions.

2.3 Partial Polymorphisms, Strong Partial Clones and Weak Partial Co-Clones

In this section we investigate clones based on partial functions instead of total functions, and show that Theorem 2 can be significantly strengthened with these notions. First, an n-ary Boolean partial function f is a map $f : X \to \mathbb{B}$ where $X \subseteq \mathbb{B}^n$. In other words f is a function that is allowed to be undefined for one or more sequences of arguments. Given a partial function $f : X \to \mathbb{B}$, $X \subseteq D^n$, we let $\mathrm{dom}(f) = X$ and $\mathrm{ar}(f) = n$. If $u = (x_1, \ldots, x_n) \in \mathrm{dom}(f)$ we use the shorthand notation $f(u)$ instead of $f(x_1, \ldots, x_n)$. A partial function g is said to be a *subfunction* of a partial function f if $\mathrm{dom}(g) \subseteq \mathrm{dom}(f)$ and $g(u) = f(u)$ for all $u \in \mathrm{dom}(g)$. A set of partial functions is *strong* if it is closed under taking subfunctions. If f is an n-ary partial function and $X \subseteq \mathrm{dom}(f)$ we write $f_{|X}$ for the subfunction of f satisfying $\mathrm{dom}(f_{|X}) = X$. Composition of partial functions is defined similarly to the case of total functions. Hence, if f is an m-ary partial function and $g_1, \ldots, g_m$ are n-ary partial functions then the composition is defined as $f \circ (g_1, \ldots, g_m)(x_1, \ldots, x_n) =$

$f(g_1(x_1,\ldots,x_n),\ldots,g_m(x_1,\ldots,x_n))$, and the resulting function is defined for every tuple $(x_1,\ldots,x_n) \in \bigcap_{i=1}^{m} \mathrm{dom}(g_i)$ such that $(g_1(x_1,\ldots,x_n),\ldots,g_m(x_1,\ldots,x_n)) \in \mathrm{dom}(f)$.

We now say that an n-ary partial function f is a *partial polymorphism* of a k-ary relation R if $(f(t_1[1],\ldots,t_n[1]),\ldots,f(t_1[k],\ldots,t_n[k])) \in R$ for all $t_1,\ldots,t_n \in R$ such that $\{(t_1[1],\ldots,t_n[1]),\ldots,(t_1[k],\ldots,t_n[k])\} \subseteq \mathrm{dom}(f)$. If we let $\mathrm{pPol}(\Gamma)$ denote the set of all partial polymorphisms of a constraint language Γ then the resulting set of partial functions is known as a *strong partial clone.* A strong partial clone $\mathrm{pPol}(\Gamma)$ is a set of partial functions which (1) is closed under composition of partial functions, i.e., if $f, g_1,\ldots,g_{\mathrm{ar}(f)} \in \mathrm{pPol}(\Gamma)$ then $f \circ (g_1,\ldots,g_{\mathrm{ar}(f)}) \in \mathrm{pPol}(\Gamma)$, (2) contains all projection functions, and (3) is closed under taking subfunctions. It is worth noting that the second and third conditions are equivalent to the condition that $\mathrm{pPol}(\Gamma)$ contains all partial projection functions, i.e., the total projection functions and all their possible subfunctions, and we let $\Pi_{\mathbb{B}}^{p}$ denote this set. Given a set of partial functions F we let $[F]_s$ denote the smallest strong partial clone which contains F. The set F is called a *base* of $[F]_s$. Similar to the relationship between clones and co-clones we can find a Galois connection between strong partial clones and sets of relation satisfying certain closure properties. In symbols, we say that a k-ary relation R has a *quantifier-free primitive positive definition* (q.f.p.p. definition) over a constraint language Γ if $R(x_1,\ldots,x_k) \equiv R_1(\mathbf{x_1}) \wedge \ldots \wedge R_m(\mathbf{x_m})$, where each $R_i \in \Gamma \cup \{\mathrm{Eq}\}$ and each $\mathbf{x_i}$ is an $\mathrm{ar}(R_i)$-ary tuple of variables over $x_1,\ldots,x_k$. We then let $\langle\Gamma\rangle_{\nexists}$ denote the smallest set of relations containing Γ which is closed under q.f.p.p. definitions, and as usual we write $\langle R\rangle_{\nexists}$ whenever $\Gamma = \{R\}$. Sets of the form $\langle\Gamma\rangle_{\nexists}$ are known as *weak partial co-clones*, or *weak systems.* We have the following Galois connection.

▶ **Theorem 3** ([9, 24]). *Let Γ and Γ' be two constraint languages. Then $\Gamma \subseteq \langle\Gamma'\rangle_{\nexists}$ if and only if $\mathrm{pPol}(\Gamma') \subseteq \mathrm{pPol}(\Gamma)$.*

Using this Galois connection Jonsson et al. [15] proved that the partial polymorphisms of a finite constraint language determines the complexity of the satisfiability problem up to $O(c^n)$ time complexity, where n denotes the number of variables in a given instance.

▶ **Theorem 4** ([15]). *Let Γ and Γ' be two finite constraint languages. If $\mathrm{pPol}(\Gamma) \subseteq \mathrm{pPol}(\Gamma')$ and $\mathrm{SAT}(\Gamma)$ is solvable in $O(c^n)$ time, then $\mathrm{SAT}(\Gamma')$ is solvable in $O(c^n)$ time, too.*

Hence, a better understanding of the lattice of Boolean strong partial clones could lead to a better understanding of the worst-case time complexity of satisfiability problems. Unfortunately, the cardinality of this lattice is equal to the continuum [1], and besides some minor results [18, 26], the details of this structure is largely unknown. In particular, it is known that the set $\{\mathrm{pPol}(\Gamma) \mid \mathrm{SAT}(\Gamma) \text{ is NP-complete}\}$ is of uncountably infinite cardinality. With a reformulation of Schaefer's dichotomy theorem [25] we can state this result even more precisely as follows (where $\neg x$ denotes the unary function $\neg x = 1 - x$).

▶ **Theorem 5** ([29]). *The sets $\{\mathrm{pPol}(\Gamma) \mid \mathrm{Pol}(\Gamma) = \Pi_{\mathbb{B}}\}$ and $\{\mathrm{pPol}(\Gamma) \mid \mathrm{Pol}(\Gamma) = [\neg x]\}$ are of uncountably infinite cardinality.*

Note that this theorem immediately implies that there exists strong partial clones of the form $\mathrm{pPol}(\Gamma)$ which does not admit a finite base. Perhaps more surprising is the following theorem which states that a strong partial clone of the form $\mathrm{pPol}(\Gamma)$ cannot have a finite base whenever Γ is a finite constraint language such that $\mathrm{Pol}(\Gamma) \subseteq [\neg x]$.

▶ **Theorem 6** ([21]). *Let Γ be a finite constraint language such that $\mathrm{Pol}(\Gamma) \subseteq [\neg x]$. Then $\mathrm{pPol}(\Gamma)$ does not admit a finite base.*

2.4 The Easiest NP-complete SAT(·) Problem

In Jonsson et al [15] it is proven that the "easiest" NP-complete SAT(·) problem can be seen as a variant of 1-in-3-SAT where each variable occurring in a constraint has a complementary variable and each constraint contains two variables forced to constant values. We can represent this problem as $\mathrm{SAT}(R_{1/3}^{\neq\neq\neq01})$ where $R_{1/3}^{\neq\neq\neq01}(x_1,x_2,x_3,x_4,x_5,x_6,c_0,c_1) \equiv R_{1/3}(x_1,x_2,x_3) \wedge R_{1/3}(c_0,c_0,c_1) \wedge R_{1/3}(x_1,x_4,c_0) \wedge R_{1/3}(x_2,x_5,c_0) \wedge R_{1/3}(x_3,x_6,c_0)$, where we have choosen the variable names c_0 and c_1 to indicate that they are forced constant values. Hence, we have that $R_{1/3}^{\neq\neq\neq01} = \{(0,0,1,1,1,0,0,1),(0,1,0,1,0,1,0,1),(1,0,0,0,1,1,0,1)\}$.

We are now interested in trying to determine the weak partial co-clones $\langle\Gamma\rangle_{\not\exists}$ such that $\langle R_{1/3}\rangle_{\not\exists} \supset \langle\Gamma\rangle_{\not\exists} \supset \langle R_{1/3}^{\neq\neq\neq01}\rangle_{\not\exists}$. In particular, is it possible to find a constraint language Γ such that $\langle\Gamma\rangle_{\not\exists}$ *covers* $\langle R_{1/3}^{\neq\neq\neq01}\rangle_{\not\exists}$? By this we mean that there does not exist any Δ such that $\langle R_{1/3}^{\neq\neq\neq01}\rangle_{\not\exists} \subset \langle\Delta\rangle_{\not\exists} \subset \langle\Gamma\rangle_{\not\exists}$. Since $R_{1/3}^{\neq\neq\neq01}$ has arity 8 and $R_{1/3}$ has arity 3, a reasonable first attempt to investigate this question is to gradually remove arguments from $R_{1/3}^{\neq\neq\neq01}$. Hence, let $R_{1/3}^{\neq\neq01}$, $R_{1/3}^{\neq01}$, and $R_{1/3}^{01}$ be the relations obtained from $R_{1/3}^{\neq\neq\neq01}$ by removing one, two, and three complemented arguments. That is, $R_{1/3}^{\neq\neq01} = \{(0,0,1,1,1,0,1),(0,1,0,1,0,0,1),(1,0,0,0,1,0,1)\}$, $R_{1/3}^{\neq01} = \{(0,0,1,1,0,1),(0,1,0,1,0,1),(1,0,0,0,0,1)\}$, and $R_{1/3}^{01} = \{(0,0,1,0,1),(0,1,0,0,1),(1,0,0,0,1)\}$. We remark that $\mathrm{Pol}(R_{1/3}^{\neq\neq\neq01}) = \mathrm{Pol}(R_{1/3}^{\neq\neq01}) = \mathrm{Pol}(R_{1/3}^{\neq01}) = \mathrm{Pol}(R_{1/3}^{01}) = \mathrm{Pol}(R_{1/3}) = \mathrm{II}_{\mathbb{B}}$, i.e., that the only total polymorphisms of these relations are the projections. In Lagerkvist [20] it was proven that $\langle R_{1/3}\rangle_{\not\exists} \subset \langle R_{1/3}^{01}\rangle_{\not\exists} \subset \langle R_{1/3}^{\neq01}\rangle_{\not\exists} \subset \langle R_{1/3}^{\neq\neq01}\rangle_{\not\exists} \subset \langle R_{1/3}^{\neq\neq\neq01}\rangle_{\not\exists}$. Hence, the inclusions in Figure 1 are correct. However, the question of whether these weak partial co-clones also cover one another was left open. We will see in Section 4 that there in fact exist an infinite number of weak partial co-clones between $\langle R_{1/3}\rangle_{\not\exists}$ and $\langle R_{1/3}^{\neq\neq\neq01}\rangle_{\not\exists}$.

3 The Partial Polymorphisms of $R_{1/3}$, $R_{1/3}^{01}$, $R_{1/3}^{\neq01}$, $R_{1/3}^{\neq\neq01}$ and $R_{1/3}^{\neq\neq\neq01}$

We now want to investigate the structure of weak partial co-clones between $\langle R_{1/3}\rangle_{\not\exists}$ and $\langle R_{1/3}^{\neq\neq\neq01}\rangle_{\not\exists}$, and determine to which extent the conjectured structure in Figure 1 is complete. Since investigating this problem purely from a relational perspective appears to be complicated, we in this section tackle the problem of characterizing the partial polymorphisms of $R_{1/3}$, $R_{1/3}^{01}$, $R_{1/3}^{\neq01}$, $R_{1/3}^{\neq\neq01}$, and $R_{1/3}^{\neq\neq\neq01}$. As a shorthand, we let $\vec{0}^n$ denote an n-ary tuple consisting only of zeroes, and similarly for $\vec{1}^n$. We write $\overline{x}$ for the complement of a tuple x. Recall that if $t_1,\ldots,t_n$ are k-ary tuples and f an n-ary partial function we by $f(t_1,\ldots,t_n)$ denote the k-ary tuple resulting from applying f componentwise to the elements of $t_1,\ldots,t_n$. To be able to more conveniently refer to the n-ary tuples of the form $(t_1[i],\ldots,t_n[i])$ we let $Cols(t_1,\ldots,t_n) = ((t_1[1],\ldots,t_n[1]),\ldots,(t_n[k],\ldots,t_n[k]))$, and $ColsSet(t_1,\ldots,t_n) = \{(t_1[1],\ldots,t_n[1]),\ldots,(t_n[k],\ldots,t_n[k])\}$. For example, we have that $Cols((0,0,1,0,1),(0,1,0,0,1),(1,0,0,0,1)) = ((0,0,1),(0,1,0),(1,0,0),(0,0,0),(1,1,1))$. We now introduce our method for describing domains of partial functions.

▶ **Definition 7.** We make the following definitions.

- A set $\{\omega_1,\ldots,\omega_k\} \subseteq \mathbb{B}^n$ is an *exact cover of* $R_{1/3}$ if (1) $2 \leq k \leq 3$ and (2) for every $i \in \{1,\ldots,n\}$ it holds that $\omega_1[i] + \ldots + \omega_k[i] = 1$.
- A set $C \cup \{\vec{0}^n, \vec{1}^n\} \subseteq \mathbb{B}^n$ is an *exact cover of* $R_{1/3}^{01}$ if C is an exact cover of $R_{1/3}$.
- A set $C \cup \{\omega, \vec{0}^n, \vec{1}^n\} \subseteq \mathbb{B}^n$ is an *exact cover of* $R_{1/3}^{\neq01}$ if (1) $\overline{\omega} \in C$ and (2) C is an exact cover of $R_{1/3}$.
- A set $C \cup \{\omega_1, \omega_2, \vec{0}^n, \vec{1}^n\} \subseteq \mathbb{B}^n$ is an *exact cover of* $R_{1/3}^{\neq\neq01}$ if (1) $\overline{\omega_1}, \overline{\omega_2} \in C$ and (2) C is an exact cover of $R_{1/3}$.
- A set $C \cup \{\omega_1, \omega_2, \omega_3, \vec{0}^n, \vec{1}^n\} \subseteq \mathbb{B}^n$ is an *exact cover of* $R_{1/3}^{\neq\neq\neq01}$ if (1) $\overline{\omega_1}, \overline{\omega_2}, \overline{\omega_3} \in C$ and (2) C is an exact cover of $R_{1/3}$.

For each $R \in \{R_{1/3}, R^{01}_{1/3}, R^{\neq 01}_{1/3}, R^{\neq\neq 01}_{1/3}, R^{\neq\neq\neq 01}_{1/3}\}$ we say that an exact cover C of R is *maximal* if there is no $C' \supset C$ which is an exact cover of R. Given $T \subseteq \mathbb{B}^n$ we let $\text{Cover}_R(T) = \{C \subseteq T \mid C \text{ is a maximal exact cover of } R\}$. We have the following link between exact covers and tuples from the relations $R_{1/3}, R^{01}_{1/3}, R^{\neq 01}_{1/3}, R^{\neq\neq 01}_{1/3}$, and $R^{\neq\neq\neq 01}_{1/3}$.

▶ **Lemma 8.** *Let $C \subseteq \mathbb{B}^n$. Then, for each $R \in \{R_{1/3}, R^{01}_{1/3}, R^{\neq 01}_{1/3}, R^{\neq\neq 01}_{1/3}, R^{\neq\neq\neq 01}_{1/3}\}$ it holds that C is maximal exact cover of R if and only if there exists $t_1, \dots, t_n \in R$ such that $ColsSet(t_1, \dots, t_n) = C$.*

Proof. We only prove the case when $R = R_{1/3}$ since the other relations can be proven with symmetrical arguments. Let $C \subseteq \mathbb{B}^n$ be a maximal exact cover of $R_{1/3}$. There are two cases to consider. First, assume that $|C| = 3$ and let $C = \{\omega_1, \omega_2, \omega_3\}$. For each $i \in \{1, \dots, n\}$ we by definition have that $\omega_1[i] + \omega_2[i] + \omega_3[i] = 1$, and hence, $(\omega_1[i], \omega_2[i], \omega_3[i]) \in R_{1/3}$. Therefore, it is easy to find $t_1, \dots, t_n \in R_{1/3}$ such that $ColsSet(t_1, \dots, t_n) = C$. Second, assume that $|C| = 2$. In this case $C = \{\vec{0}^n, \vec{1}^n\}$, from which it folows that $ColsSet(t_1, \dots, t_n) = \{\vec{0}^n, \vec{1}^n\}$ whenever $t_i = t_j$ for all $i, j \in \{1, \dots, n\}$. Now assume that $t_1, \dots, t_n \in R_{1/3}$. We must prove that $ColsSet(t_1, \dots, t_n) = \{\omega_1, \dots, \omega_k\}$ is an exact cover of $R_{1/3}$. But this is trivial since (1) $2 \leq |\{\omega_1, \dots, \omega_k\}| \leq 3$ and (2) $\omega_1[i] \dots + \omega_k[i] = 1$ for each $i \in \{1, \dots, n\}$. ◀

We now have everything in place to state the main results of this section.

▶ **Theorem 9.** *Let f be an n-ary function. Then $f \in \text{pPol}(R_{1/3})$ if and only if*
1. *$f_{|C} \in \Pi^p_{\mathbb{B}}$ for every $C \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$, or*
2. *$f(\vec{0}^n) = 1$, $\vec{1}^n \notin \text{dom}(f)$, and for every $C \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$ either (1) $f(\omega) = f(\overline{\omega}) = 0$ if $C = \{\vec{0}^n, \omega, \overline{\omega}\}$ or (2) $f_{|C} \in \Pi^p_{\mathbb{B}}$ if $\vec{0}^n \notin C$.*

Proof. We begin with the completeness part of the proof. Let $f \in \text{pPol}(R_{1/3})$ be an n-ary partial function. We must prove that f satisfies condition (1) or condition (2). Assume first that there exists $\{\omega_1, \omega_2, \omega_3\} \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$ such that $f_{|\{\omega_1, \omega_2, \omega_3\}}$ is not a partial projection function. Now assume that $\{\vec{0}^n, \omega, \overline{\omega}\} \notin \text{Cover}_{R_{1/3}}(\text{dom}(f))$. Then there exists $t_1, \dots, t_n \in R_{1/3}$ such that (1) $ColsSet(t_1, \dots, t_n) = \{\omega_1, \omega_2, \omega_3\}$ and (2) $|\{t_1, \dots, t_n\}| = 3$. This implies that $f_{|\{\omega_1, \omega_2, \omega_3\}}(t_1, \dots, t_n) = f(t_1, \dots, t_n) \notin R_{1/3}$. Since this contradicts the original assumption, it must be the case that $\vec{0}^n \in C$ for some $C \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$. Assume first that $f(\vec{0}^n) = 0$ and that $f(\omega) = f(\overline{\omega}) = 0$ for some $\{\vec{0}^n, \omega, \overline{\omega}\} \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$. In this case $f \notin \text{pPol}(R_{1/3})$, and similarly when $f(\vec{0}^n) = 0$ and $f(\omega) = f(\overline{\omega}) = 1$. Last, assume that $f(\vec{0}^n) = 0$ and that $f(\omega) = \overline{f(\overline{\omega})}$. In this case $f_{|\{\vec{0}^n, \omega, \overline{\omega}\}}$ is a partial projection, and, furthermore, $f_{|C}$ must be a partial projection for every $C \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$, i.e., we are in case (1). Assume now instead that $\vec{0}^n \in C$ for some $C \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$ and that $f(\vec{0}^n) = 1$. In this case one can easily verify that if there exists $\{\vec{0}^n, \omega, \overline{\omega}\} \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$ such that either $f(\omega) = 1$ or $f(\overline{\omega}) = 1$ then $f \notin \text{pPol}(R_{1/3})$. Hence, $f(\vec{0}^n) = 1$, $f(\omega) = f(\overline{\omega}) = 0$ for all $\{\vec{0}^n, \omega, \overline{\omega}\} \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$, and $f_{|C}$ is a partial projection for all $C \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$ such that $\vec{0}^n \notin C$. This means that we are in case (2).

For the soundness part of the proof, assume that f is an n-ary partial function fullfiling condition (1). Let $t_1, \dots, t_n \in R_{1/3}$ and let $\{\omega_1, \omega_2, \omega_3\} = ColsSet(t_1, \dots, t_n)$. We must prove that either $(f(\omega_1), f(\omega_2), f(\omega_3)) \in R_{1/3}$ or that $f(\omega_i)$ is undefined for some $i \in \{1, 2, 3\}$. Clearly, if $\{\omega_1, \omega_2, \omega_3\} \in \text{Cover}_{R_{1/3}}(\text{dom}(f))$ then, by assumption, $(f(\omega_1), f(\omega_2), f(\omega_3)) \in R_{1/3}$ since $f_{|\{\omega_1, \omega_2, \omega_3\}}$ is a partial projection. Otherwise, if $\{\omega_1, \omega_2, \omega_3\} \notin \text{Cover}_{R_{1/3}}(\text{dom}(f))$, then $f(\omega_i)$ must be undefined for some $i \in \{1, 2, 3\}$. Now assume that f is an n-ary partial function fullfiling condition (2). Let $t_1, \dots, t_n \in R_{1/3}$. Observe that if $\vec{0}^n \notin ColsSet(t_1, \dots, t_n)$

then it directly follows that $f(t_1,\ldots,t_n) \in R_{1/3}$ or that $f(t_1,\ldots,t_n)$ is undefined, by recapitulating the proof of the preceding paragraph. Hence, assume that $ColsSet(t_1,\ldots,t_n) = \{\vec{0}^n,\omega,\overline{\omega}\}$. By assumption we have that either $f(t_1,\ldots,t_n) \in R_{1/3}$ (since $f(\vec{0}^n) = 1$ and $f(\omega) = f(\overline{\omega}) = 0$), or that $\omega \notin \mathrm{dom}(f)$ or that $\overline{\omega} \notin \mathrm{dom}(f)$. ◀

Similarly we can characterize the partial polymorphisms of $R^{01}_{1/3}$, $R^{\neq 01}_{1/3}$, $R^{\neq\neq 01}_{1/3}$, and $R^{\neq\neq\neq 01}_{1/3}$.

▶ **Theorem 10.** *Let $R \in \{R^{01}_{1/3}, R^{\neq 01}_{1/3}, R^{\neq\neq 01}_{1/3}, R^{\neq\neq\neq 01}_{1/3}\}$. Let f be an n-ary partial function. Then $f \in \mathrm{pPol}(R)$ if and only if*

1. *$\vec{0}^n \notin \mathrm{dom}(f)$, or*
2. *$\vec{1}^n \notin \mathrm{dom}(f)$, or*
3. *$\{\vec{0}^n, \vec{1}^n\} \subseteq \mathrm{dom}(f)$ and $f_{|C} \in \Pi^p_{\mathbb{B}}$ for every $C \in \mathrm{Cover}_R(\mathrm{dom}(f))$.*

Proof. We only prove the case when $R = R^{\neq 01}_{1/3}$ since the other cases are entirely analogous. For soundness, assume that f is an n-ary partial function satisfying condition (1), (2), or (3). Let $t_1,\ldots,t_n \in R^{\neq 01}_{1/3}$. We have two cases to consider: either (1) $ColsSet(t_1,\ldots,t_n) \not\subseteq \mathrm{dom}(f)$ in which case $f(t_1,\ldots,t_n)$ is undefined; or (2) $ColsSet(t_1,\ldots,t_n) \subseteq \mathrm{dom}(f)$ in which case $ColsSet(t_1,\ldots,t_n) \in \mathrm{Cover}_{R^{\neq 01}_{1/3}}(\mathrm{dom}(f))$ and $f(t_1,\ldots,t_n) \in R^{\neq 01}_{1/3}$.

For completeness, let $f \in \mathrm{pPol}(R^{\neq 01}_{1/3})$ be an n-ary partial function such that $\{\vec{0}^n, \vec{1}^n\} \subseteq \mathrm{dom}(f)$. First, it is easy to verify that $f(\vec{0}^n) = 0$ and that $f(\vec{1}^n) = 1$, since otherwise $f \notin \mathrm{pPol}(R^{\neq 01}_{1/3})$. Let $C \in \mathrm{Cover}_{R^{\neq 01}_{1/3}}(\mathrm{dom}(f))$. We prove that $f_{|C}$ must be a partial projection function. There are a few different cases depending on the size of C. First note that the size of a maximum exact $R^{\neq 01}_{1/3}$-cover of $\mathrm{dom}(f)$ is always either 6, 4, or 2.

First, assume that $|C| = 6$ and let $C = \{\omega_1, \omega_2, \omega_3, \overline{\omega_1}, \vec{0}^n, \vec{1}^n\}$, where $\{\omega_1,\omega_2,\omega_3\}$ is an exact cover of $R_{1/3}$. According to Lemma 8, there exists $t_1,\ldots,t_n \in R^{\neq 01}_{1/3}$ such that $ColsSet(t_1,\ldots,t_n) = C$ and such that $\{t_1,\ldots t_n\} = R^{\neq 01}_{1/3}$. If $f_{|C}$ is not a projection on C then $f_{|C}(t_1,\ldots,t_n) = f(t_1,\ldots,t_n) \notin R^{\neq 01}_{1/3}$.

Second, assume that $|C| = 4$ and let $C = \{\omega, \overline{\omega}, \vec{0}^n, \vec{1}^n\}$. Then, according to Lemma 8 there exists $t_1,\ldots,t_n \in R^{\neq 01}_{1/3}$ such that $ColsSet(t_1,\ldots,t_n) = C$. Now note that if $f_{|C}$ is not a projection, then either $f(\omega) = f(\overline{\omega}) = 0$ or that $f(\omega) = f(\overline{\omega}) = 1$, and in both these cases $f \notin \mathrm{pPol}(R^{\neq 01}_{1/3})$. Hence, $f(\omega) = \overline{f(\overline{\omega})}$, from which it follows that $f_{|C}$ is a partial projection.

Third, assume that $|C| = 2$, and observe that this implies that $C = \{\vec{0}^n, \vec{1}^n\}$, due to the assumption that C is maximal. Since, by assumption, $f(\vec{0}^n) = 0$ and $f(\vec{1}^n) = 1$, it follows that $f_{|C}$ is a projection. ◀

Hence, even though $\mathrm{pPol}(R_{1/3})$, $\mathrm{pPol}(R^{01}_{1/3})$, $\mathrm{pPol}(R^{\neq 01}_{1/3})$, $\mathrm{pPol}(R^{\neq\neq 01}_{1/3})$, and $\mathrm{pPol}(R^{\neq\neq\neq 01}_{1/3})$ cannot be described through finite bases (by Theorem 6), we could still obtain a complete understanding of the involved partial functions. We remark that the partial polymorphisms of 1-in-k-SAT has been described in earlier work [22], but in contrast to Theorem 9 and Theorem 10, the proposed classificaton only describes a finite subset of partial polymorphisms.

4 The Structure Between $\langle R_{1/3}\rangle_{\not\exists}$ and $\langle R^{\neq\neq\neq 01}_{1/3}\rangle_{\not\exists}$

In this section we use the results from Section 3 in order to investigate the structure of the weak partial co-clones between $\langle R_{1/3}\rangle_{\not\exists}$ and $\langle R^{\neq\neq\neq 01}_{1/3}\rangle_{\not\exists}$. Before delving deeper into the forthcoming proofs the reader is advised to first consult Figure 2 for a visualization of the main results. We concentrate on weak partial co-clones below $\langle R^{01}_{1/3}\rangle_{\not\exists}$ since it is readily seen that the problems $\mathrm{SAT}(R_{1/3})$ and $\mathrm{SAT}(R^{01}_{1/3})$ have the same worst-case time complexity. It is in fact not difficult to prove that there cannot exist any R such that $|R| \leq 3$ and such that $\langle R^{01}_{1/3}\rangle_{\not\exists} \subset \langle R\rangle_{\not\exists} \subset \langle R^{\neq 01}_{1/3}\rangle_{\not\exists}$, and the same also holds for weak partial co-clones between

$$\begin{tikzcd}
 & \langle R_{1/3}\rangle_{\nexists} & \\
 & \langle R^{01}_{1/3}\rangle_{\nexists} \arrow[u] & \\
\langle \alpha^5\rangle_{\nexists} \arrow[ur] & \langle \alpha^i\rangle_{\nexists} \arrow[u] \arrow[l, "/" marking] \arrow[r, "/" marking] & \langle \alpha^{i+1}\rangle_{\nexists} \arrow[ul] \\
 & \langle R^{\neq 01}_{1/3}\rangle_{\nexists} \arrow[ul] \arrow[u] \arrow[ur] & \\
\langle \beta^5\rangle_{\nexists} \arrow[ur] & \langle \beta^i\rangle_{\nexists} \arrow[u] \arrow[l, "/" marking] \arrow[r, "/" marking] & \langle \beta^{i+1}\rangle_{\nexists} \arrow[ul] \\
 & \langle R^{\neq\neq 01}_{1/3}\rangle_{\nexists} \arrow[ul] \arrow[u] \arrow[ur] & \\
\langle \gamma^5\rangle_{\nexists} \arrow[ur] & \langle \gamma^i\rangle_{\nexists} \arrow[u] \arrow[l, "/" marking] \arrow[r, "/" marking] & \langle \gamma^{i+1}\rangle_{\nexists} \arrow[ul] \\
 & \langle R^{\neq\neq\neq 01}_{1/3}\rangle_{\nexists} \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 2 The structure of weak partial co-clones below $\langle R_{1/3}\rangle_{\nexists}$. An arrow of the form $A \to B$ means that $A \subset B$. An arrow of the form $A \not\to B$ means that $A \not\subset B$.

all other cases. Hence, to find elements between we must consider relations of cardinality strictly larger than 3. With this as a guidance we define the following class of relations, where $\neq$ denotes the binary inequality relation $\{(0,1),(1,0)\}$ and $R_{1/k}$ denotes the k-ary relation $\{(x_1,\ldots,x_k) \in \mathbb{B}^k \mid \Sigma_{i=1}^k x_i = 1)\}$.

▶ **Definition 11.** Let $k \geq 5$. The relation α^k is defined as

$$\alpha^k(x_1,\ldots,x_k,y_1,\ldots,y_{k-3},z_1,\ldots,z_{k-3},w_1,\ldots,w_{k-4},c_0,c_1) \equiv R_{1/k}(x_1,\ldots,x_k)\wedge$$
$$\bigwedge_{i=1}^{k-3} R_{1/i+2}(x_1,\ldots,x_{i+1},y_i) \wedge \bigwedge_{i=1}^{k-3} y_i \neq z_i \wedge \bigwedge_{i=3}^{k-2} R_{1/3}(x_1,x_i,w_{i-2}) \wedge R_{1/3}(c_0,c_0,c_1).$$

The relation β^k for $k \geq 5$ is defined similarly but with $k-2$ additional arguments which are the complement of $x_1, x_3, \ldots, x_{k-1}$. Hence, let

$$\beta^k(x_1,\ldots,x_k,y_1,\ldots,y_{k-3},z_1,\ldots,z_{k-3},w_1,\ldots,w_{k-4},v_1,\ldots,v_{k-2},c_0,c_1) \equiv$$
$$\alpha^k(x_1,\ldots,x_k,y_1,\ldots,y_{k-3},z_1,\ldots,z_{k-3},w_1,\ldots,w_{k-4},c_0,c_1) \wedge x_1 \neq z_1 \wedge \bigwedge_{i=3}^{k-1} x_i \neq v_{i-1}.$$

Finally, the relation γ^k for $k \geq 5$ can be defined as

$$\gamma^k(x_1,\ldots,x_k,y_1,\ldots,y_{k-3},z_1,\ldots,z_{k-3},w_1,\ldots,w_{k-4},v_1,\ldots,v_k,c_0,c_1) \equiv$$
$$\alpha^k(x_1,\ldots,x_k,y_1,\ldots,y_{k-3},z_1,\ldots,z_{k-3},w_1,\ldots,w_{k-4},c_0,c_1) \wedge \bigwedge_{i=1}^{k} x_i \neq v_i.$$

Later in this section we will see that $\langle R^{\neq\neq\neq 01}_{1/3}\rangle_{\nexists} \subset \langle \gamma^k\rangle_{\nexists} \subset \langle R^{\neq\neq 01}_{1/3}\rangle_{\nexists} \subset \langle \beta^k\rangle_{\nexists} \subset \langle R^{\neq 01}_{1/3}\rangle_{\nexists} \subset \langle \alpha^k\rangle_{\nexists} \subset \langle R^{01}_{1/3}\rangle_{\nexists}$ for each $k \geq 5$. The intuition behind the relation α^k is as follows.

- The k first arguments $x_1,\ldots,x_k$ encode a 1-in-k-constraint.
- Since $R_{1/k} \notin \langle R^{01}_{1/3}\rangle_{\nexists}$ [20], we have to add arguments to make it q.f.p.p. definable by $R^{01}_{1/3}$.
- These arguments are $y_1,\ldots,y_{k-3}$, their complements $z_1,\ldots,z_{k-3}$, and the two constant arguments c_0 and c_1.

- To make sure that the resulting relation is not q.f.p.p. definable by $R^{\neq 01}_{1/3}$ we also need the additional arguments $w_1, \ldots, w_{k-4}$, which do not have any complementary arguments.

The relations β^k and γ^k can be understood in a similar way. We then have the following straightforward Lemma which states that the weak partial co-clones of the relations α^k, β^k and γ^k, are proper subsets of $\langle R^{01}_{1/3} \rangle_{\nexists}$, $\langle R^{\neq 01}_{1/3} \rangle_{\nexists}$, and $\langle R^{\neq\neq 01}_{1/3} \rangle_{\nexists}$, respectively.

▶ **Lemma 12.** *$\langle R^{01}_{1/3} \rangle_{\nexists} \supset \langle \alpha^k \rangle_{\nexists}$, $\langle R^{\neq 01}_{1/3} \rangle_{\nexists} \supset \langle \beta^k \rangle_{\nexists}$, and $\langle R^{\neq\neq 01}_{1/3} \rangle_{\nexists} \supset \langle \gamma^k \rangle_{\nexists}$ for each $k \geq 5$.*

Proof. Let $k \geq 5$. We only consider the case $\langle R^{01}_{1/3} \rangle_{\nexists} \supset \langle \alpha^k \rangle_{\nexists}$ since the other cases are similar. We begin by proving that $\alpha^k \in \langle R^{01}_{1/3} \rangle_{\nexists}$ which implies that $\langle R^{01}_{1/3} \rangle_{\nexists} \supseteq \langle \alpha^k \rangle_{\nexists}$. The base case when $k = 5$ is simple:

$$\begin{aligned}
&\alpha^5(x_1, x_2, x_3, x_4, x_5, y_1, y_2, z_1, z_2, w_1, c_0, c_1) \equiv \\
&R^{01}_{1/3}(x_1, x_2, y_1, c_0, c_1) \land R^{01}_{1/3}(x_3, y_2, z_1, c_0, c_1) \land \\
&R^{01}_{1/3}(c_0, y_1, z_1, c_0, c_1) \land R^{01}_{1/3}(x_4, x_5, z_2, c_0, c_1) \land \\
&R^{01}_{1/3}(c_0, y_2, z_2, c_0, c_1) \land R^{01}_{1/3}(x_1, x_3, w_1, c_0, c_1).
\end{aligned}$$

For the inductive step assume that $\alpha^{k-1} \in \langle R^{01}_{1/3} \rangle_{\nexists}$. We can then implement α^k as follows.

$$\begin{aligned}
&\alpha^k(x_1, \ldots, x_k, y_1, y_2, \ldots, y_{k-4}, y_{k-3}, z_1, z_2, \ldots, z_{k-4}, z_{k-3}, w_1, \ldots, w_{k-5}, w_{k-4}, c_0, c_1) \equiv \\
&\quad \alpha^{k-1}(x_1, \ldots, x_{k-2}, y_{k-3}, y_1, \ldots, y_{k-4}, z_1, \ldots, z_{k-4}, w_1, \ldots, w_{k-5}, c_0, c_1) \land \\
&\quad R^{01}_{1/3}(z_{k-3}, x_{k-1}, x_k, c_0, c_1) \land R^{01}_{1/3}(c_0, y_{k-3}, z_{k-3}, c_0, c_1) \land R^{01}_{1/3}(x_1, x_{k-2}, w_{k-4}, c_0, c_1).
\end{aligned}$$

To prove the proper inclusion $\langle R^{01}_{1/3} \rangle_{\nexists} \supset \langle \alpha^k \rangle_{\nexists}$ we show that there exists $f \in \mathrm{pPol}(\alpha^k)$ such that $f \notin \mathrm{pPol}(R^{01}_{1/3})$. Consider the ternary partial function f defined such that $f(0,0,0) = 0$, $f(1,1,1) = 1$, and $f(0,0,1) = f(0,1,0) = f(1,0,0) = 0$. By Theorem 10 it follows that $f \notin \mathrm{pPol}(R^{01}_{1/3})$, but it is not difficult to verify that for any sequence of three tuples $t_1, t_2, t_3 \in \alpha^k$, the set $\mathit{ColsSet}(t_1, t_2, t_3)$ will either be of the form $\{(0,0,0), (1,1,1)\}$, or it will contain the complement of $(0,0,1)$, $(0,1,0)$, or $(1,0,0)$. Hence, $f(t_1, t_2, t_3)$ is either a projection or undefined, from which it follows that $f \in \mathrm{pPol}(\alpha^k)$. ◀

We now need to prove that it cannot be the case that $\langle \alpha^k \rangle_{\nexists} = \langle \alpha^{k'} \rangle_{\nexists}$ whenever $k \neq k'$, and similarly for the relations β^k and γ^k. Before proving this we need a slight generalisation of the concept of exact covers from Definition 7. A set $\{\omega_1, \ldots, \omega_{k'}\} \subseteq \mathbb{B}^n$ is a *exact k'-cover of $R_{1/k}$* if (1) $2 \leq k' \leq k$ and (2) for every $i \in \{1, \ldots, n\}$ it holds that $\omega_1[i] + \ldots + \omega_{k'}[i] = 1$. From Lagerkvist et al. [22] it follows that $t_1, \ldots, t_{k'} \in R_{1/k}$ if and only if $\mathit{ColsSet}(t_1, \ldots, t_{k'})$ is an exact k'-cover of $R_{1/k}$. This implies that $t_1, \ldots, t_{k'} \in \alpha^k$ if and only if $\{(t_1[1], \ldots, t_{k'}[1]), \ldots, (t_1[k], \ldots, t_{k'}[k])\}$ is an exact k'-cover of $R_{1/k}$.

▶ **Lemma 13.** *For each $k, k' \geq 5$ such that $k > k'$ there exists $f \in \mathrm{pPol}(\alpha^k)$, $f' \in \mathrm{pPol}(\beta^k)$, and $f'' \in \mathrm{pPol}(\gamma^k)$ such that $f \notin \mathrm{pPol}(\alpha^{k'})$, $f' \notin \mathrm{pPol}(\beta^{k'})$, and $f'' \notin \mathrm{pPol}(\gamma^{k'})$.*

Proof. We only consider the relations α^k and $\alpha^{k'}$ since the other cases are similar. We provide a partial function $f^{k'}$ such that $f^{k'} \in \mathrm{pPol}(\alpha^k)$ but $f^{k'} \notin \mathrm{pPol}(\alpha^{k'})$. Let $j = \mathrm{ar}(\alpha^{k'})$, let $\{t_1, \ldots, t_{k'}\} = \alpha^{k'}$, and define $f^{k'}$ such that $\mathrm{dom}(f^{k'}) = \mathit{ColsSet}(t_1, \ldots, t_{k'})$ and such that $f^{k'}(t_1, \ldots, t_{k'}) = (t_1[1], \ldots, t_1[j-3], c, 0, 1)$, where $c = t_1[1] \oplus t_1[k'-3]$. Note that, by definition, $f^{k'}(t_1, \ldots, t_{k'}) \notin \alpha^{k'}$ since any tuple $t \in \alpha^{k'}$ satisfies $t[1] + t[k'-3] + t[j-2] = 1$. Hence, $f^{k'} \notin \mathrm{pPol}(\alpha^{k'})$. Given a tuple $t \in \{0,1\}^k$ we let $\Sigma t = \Sigma_{i=1}^{k} t[i]$. Let $\mathbf{x} = (t_1[j-3], \ldots, t_{k'}[j-3])$, and note that due to the constraint $R^{01}_{1/3}(x_1, x_{k-2}, w_{k-4}, c_0, c_1)$ in Definition 11, $\Sigma\mathbf{x} = 2$, i.e., the sequence $\mathbf{x}$ contains exactly two zeroes. We now prove that $f^{k'} \in \mathrm{pPol}(\alpha^k)$. Let $u_1, \ldots, u_{k'} \in \alpha^k$ and assume, with the aim of reaching a contradiction, that $f^{k'}(u_1,\ldots,u_{k'}) \notin \alpha^k$. Since $f^{k'}(\mathbf{y}) = \pi_1^{k'}(\mathbf{y})$ for all $\mathbf{y} \in \mathrm{dom}(f^{k'}) \setminus \{\mathbf{x}\}$, it follows that $\mathbf{x}$ is included in the sequence $Cols(u_1,\ldots,u_{k'})$. There are now only three possible distinct cases to consider depending on where the sequence $\mathbf{x}$ occurs in $Cols(u_1,\ldots,u_{k'})$. We will show that for each of these cases $f^{k'}(u_1,\ldots,u_{k'})$ must be undefined, contradicting the assumption that $f^{k'}(u_1,\ldots,u_{k'}) \notin \alpha^k$. Let

$$(a_1,\ldots,a_k,b_1,\ldots,b_{k-3},c_1,\ldots,c_{k-3},d_1,\ldots,d_{k-4},\vec{0}^{k'},\vec{1}^{k'}) = Cols(u_1,\ldots,u_{k'}).$$

First assume that $\mathbf{x} = a_i$ for some $1 \leq i \leq k$. Since $\{a_1,\ldots,a_k\}$ must form an exact k'-cover and since $\Sigma\mathbf{x} = \Sigma a_i = k' - 2$ it follows that there exists a_j and $a_{j'}$ such that $\{a_i,a_j,a_{j'}\}$ is an exact k'-cover and that all other a_l sequences are equal to $\vec{0}^{k'}$. If $|\{a_i,a_j,a_{j'}\}| = 2$ then $\overline{a_j} = \overline{a_{j'}} = a_i$, which is a contradiction since $\overline{a_j} = \overline{\mathbf{x}} \notin \mathrm{dom}(f^{k'})$. Assume that $|\{a_i,a_j,a_{j'}\}| = 3$. This implies that $\Sigma a_j = \Sigma a_{j'} = 1$ and that there exists a tuple $u \in ColsSet(u_1,\ldots,u_{k'})$ such that either $u = \overline{a_i}, u = \overline{a_j}$, or $u = \overline{a_{j'}}$. In each of these cases it follows that $u \notin \mathrm{dom}(f)$, and we reach a contradiction.

Second, assume that $\mathbf{x} = b_i$ for some $i \in \{1,\ldots,k-3\}$. Then $c_i = \overline{\mathbf{x}}$. This is a contradiction since $\overline{\mathbf{x}} \notin \mathrm{dom}(f^{k'})$. The case when $\mathbf{x} = c_i$ for some $i \in \{1,\ldots,k-3\}$ is entirely analogous.

Third, assume that $\mathbf{x} = d_i$ for some $i \in \{1,\ldots,k-4\}$. Then, due to the constraint $R_{1/3}(x_1,x_{i-2},w_i)$ in Definition 11, it follows that $\Sigma a_1 = \Sigma a_{i-2} = 1$. Assume without loss of generality that $a_1[k'] = 1$. Now note that each constraint of the form $R_{1/3}(x_1,x_j,w_{j-2})$ for $j \in \{3,\ldots,k-2\}$ also implies that $\Sigma a_j \geq 1$. First, assume $\Sigma a_j = 1$ for $j \in \{1,\ldots,k-2\}$. Since $k' < k$ and since $\{a_1,\ldots,a_k\}$ must form an exact k'-cover, it follows that either $\Sigma a_{k-1} + \Sigma a_k = 1$ or that $\Sigma a_{k-1} + \Sigma a_k = 0$, and in both these cases $f^{k'}(u_1,\ldots,u_{k'})$ must be undefined. Assume that there exists some $a_{j'}$, $j' \in \{2,\ldots,i-1,i+1,\ldots,k\}$ such that $\Sigma a_{j'} > 1$. Since $\Sigma a_j \geq 1$ for each $j \in \{1,\ldots,k-2\}$ and since $\Sigma_{j=1}^{k} \Sigma a_j = k' < k$, it follows that $\Sigma a_{j'} = 2$, $k' = k-1$, and that $\Sigma a_j = 1$ for every $j \in \{1,\ldots,k-2\}$. This implies that $\Sigma a_{k-1} + \Sigma a_k = 1$, and, due to the constraint $R_{1/4}(x_{k-2},x_{k-1},x_k,z_{k-4})$, that $c_{k-4} = \overline{a_{k-2}}$. If $\Sigma a_{k-2} = 1$ then $\overline{a_{k-2}} \notin \mathrm{dom}(g)$. Hence, assume that $j' = k-2$ and that $\Sigma a_{k-2} = 2$. Due to the constraint $R_{1/3}(x_1,x_{k-2},w_{k-4})$ it follows that $\Sigma d_{k-4} = k-3$, and since $a_1[k'] = 1$, we have that $a_{k-2} \notin \mathrm{dom}(f^{k'})$ or that $d_{k-4} \notin \mathrm{dom}(f^{k'})$. ◀

Last, to get the inclusion structure in Figure 2, we need to prove that $\langle \alpha^k \rangle_{\nexists} \supset \langle R^{\neq 01}_{1/3} \rangle_{\nexists}$, $\langle \beta^k \rangle_{\nexists} \supset \langle R^{\neq\neq 01}_{1/3} \rangle_{\nexists}$, and $\langle \gamma^k \rangle_{\nexists} \supset \langle R^{\neq\neq\neq 01}_{1/3} \rangle_{\nexists}$.

▶ **Lemma 14.** *$\langle \alpha^k \rangle_{\nexists} \supset \langle R^{\neq 01}_{1/3} \rangle_{\nexists}$, $\langle \beta^k \rangle_{\nexists} \supset \langle R^{\neq\neq 01}_{1/3} \rangle_{\nexists}$, and $\langle \gamma^k \rangle_{\nexists} \supset \langle R^{\neq\neq\neq 01}_{1/3} \rangle_{\nexists}$, for each $k \geq 5$.*

Proof. We only consider α^k since the other cases are similar. To prove that $\langle \alpha^k \rangle_{\nexists} \supseteq \langle R^{\neq 01}_{1/3} \rangle_{\nexists}$, we use the q.f.p.p. definition

$$R^{\neq 01}_{1/3}(x_1,x_2,x_3,x_4,c_0,c_1) \equiv$$
$$\alpha^k(\underbrace{c_0,\ldots,c_0}_{k-3},x_1,x_2,x_3,\underbrace{c_1,\ldots,c_1}_{k-4},x_4,\underbrace{c_0,\ldots c_0}_{k-4},x_1,\underbrace{c_1,\ldots,c_1}_{k-5},x_4,c_0,c_1).$$

For the proper inclusion, simply note that the function f^k in the proof of Lemma 13 does not preserve α^k. An application of Theorem 10 shows that $f^k \in \mathrm{pPol}(R^{\neq 01}_{1/3})$. ◀

By combining Lemma 12, Lemma 13 and Lemma 14 we have thus proved the main result of the paper.

▶ **Theorem 15.** *The cardinalities of the sets $\{\langle \Gamma \rangle_{\nexists} \mid \langle R^{\neq 01}_{1/3} \rangle_{\nexists} \subset \langle \Gamma \rangle_{\nexists} \subset \langle R^{01}_{1/3} \rangle_{\nexists}\}$, $\{\langle \Gamma \rangle_{\nexists} \mid \langle R^{\neq\neq 01}_{1/3} \rangle_{\nexists} \subset \langle \Gamma \rangle_{\nexists} \subset \langle R^{\neq 01}_{1/3} \rangle_{\nexists}\}$, and $\{\langle \Gamma \rangle_{\nexists} \mid \langle R^{\neq\neq\neq 01}_{1/3} \rangle_{\nexists} \subset \langle \Gamma \rangle_{\nexists} \subset \langle R^{\neq\neq 01}_{1/3} \rangle_{\nexists}\}$ are at least countably infinite.*

5 Concluding Remarks and Future Research

We have studied the structure of NP-complete satisfiability problems whose complexity is not higher than $\mathrm{SAT}(R_{1/3})$. By using partial clone theory we have proven that one can find an infinite number of such satisfiability problems, and in the process we have also obtained a complete description of the partial polymorphisms of $R_{1/3}$. These results raise two questions that we deem particularly interesting for future research.

Algorithms based on partial polymorphisms. There exist many examples in the literature of polynomial-time algorithms based on properties of polymorphisms of constraint languages. For example, one can use polynomial-time algorithms based on Gaussian elimination to solve constraint satisfaction problems whenever the constraint language contains a so-called k-edge polymorphism [12]. By Theorem 4 we know that the partial polymorphisms of a constraint language correlates to the worst-case complexity of the corresponding satisfiability problem. Is it possible to exploit the information given by the partial polymorphisms to construct better exponential-time algorithms for satisfiability problems? In particular, can the classification in Theorem 9 be used to improve algorithms for 1-in-3-SAT? For a concrete example, consider the following strategy: it is known that the *inverse satisfiability problem* for $R_{1/3}$, $\text{Inv-SAT}(R_{1/3})$, is co-NP-complete [17]. In our terminology this problem can be stated as determining whether a given relation R is included in $\langle R_{1/3} \rangle_{\nexists}$, and can therefore be restated as whether $\mathrm{pPol}(R_{1/3}) \subseteq \mathrm{pPol}(R)$. Hence, to solve $\mathrm{SAT}(R_{1/3})$ we can utilize a Turing reduction to $\text{Inv-SAT}(R_{1/3})$, which in turn can be solved by enumerating the partial polymorphisms of $R_{1/3}$ and checking if they preserve R. Would it be possible to transform this rather implicit algorithm into an efficient algorithm for 1-in-3-SAT?

Uncountably many weak partial co-clones? We have proven that there exists at least a countably infinite number of weak partial co-clones below $\langle R_{1/3} \rangle_{\nexists}$. Is it possible to strengthen this even further and prove that there exists an uncountably infinite number of such weak partial co-clones? A starting point for proving this is to first show that the converse of Lemma 13 also holds, i.e., that $\langle \alpha^{k} \rangle_{\nexists}$ and $\langle \alpha^{k'} \rangle_{\nexists}$ are always incomparable whenever $k \neq k'$.

Does $\langle R_{1/3} \rangle_{\nexists}$ cover $\langle R^{01}_{1/3} \rangle_{\nexists}$? In this paper we restricted ourselves to study weak partial co-clones below $\langle R^{01}_{1/3} \rangle_{\nexists}$ since the two problems $\mathrm{SAT}(R_{1/3})$ and $\mathrm{SAT}(R^{01}_{1/3})$ have the same worst-case time complexity. From an algebraical point of view, however, it would be interesting to prove or disprove that $\langle R_{1/3} \rangle_{\nexists}$ covers $\langle R^{01}_{1/3} \rangle_{\nexists}$, since only a handful of such results are known in the literature [10]. This question might not be as easy as one might believe at a first glance, since it is e.g. known that there exist an uncountably infinite number of weak partial co-clones between $\langle \mathrm{OR} \rangle_{\nexists}$ and $\langle \mathrm{OR}^{01} \rangle_{\nexists}$, where $\mathrm{OR} = \{(0,1),(1,0),(1,1)\}$ and $\mathrm{OR}^{01} = \{(0,1,0,1),(1,0,0,1),(1,1,0,1)\}$ [29]. Hence, even though the relations $R_{1/3}$ and $R^{01}_{1/3}$ might appear to be almost identical, it might indeed be very hard to prove that $\langle R_{1/3} \rangle_{\nexists}$ covers $\langle R^{01}_{1/3} \rangle_{\nexists}$.

Acknowledgements. The first author has received funding from the DFG-funded project "Homogene Strukturen, Bedingungserfüllungsprobleme, und topologische Klone" (Project number 622397).

References

1 V. B. Alekseev and A. A. Voronenko. On some closed classes in partial two-valued logic. *Discrete Mathematics and Applications*, 4(5):401–419, 1994. `doi:10.1515/dma.1994.4.5.401`.

2 L. Barto. Constraint satisfaction problem and universal algebra. *ACM SIGLOG News*, 1(2):14–24, October 2014.

3 M. Behrisch, M. Hermann, S. Mengel, and G. Salzer. Give me another one! In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC-2015)*, pages 664–676, 2015.

4 M. Behrisch, M. Hermann, S. Mengel, and G. Salzer. As close as it gets. In *Proceedings of the 10th International Workshop on Algorithms and Computation (WALCOM-2016)*, pages 222–235, 2016.

5 V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. I. *Cybernetics*, 5:243–252, 1969.

6 V. G. Bodnarchuk, L. A. Kaluzhnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras. II. *Cybernetics*, 5:531–539, 1969.

7 F. Börner, L. Haddad, and R. Pöschel. Minimal partial clones. *Bull. Austral. Math. Soc.*, 44:405–415, 1991.

8 A. Bulatov and A. Hedayaty. Counting problems and clones of functions. *Multiple-Valued Logic and Soft Computing*, 18(2):117–138, 2012.

9 D. Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27(1):95–100, 1968.

10 L. Haddad and K. Schölzel. Countable intervals of partial clones. In *Proceedings of the 44th IEEE International Symposium on Multiple-Valued Logic, (ISMVL-2010)*, pages 155–160. IEEE Computer Society, 2014.

11 T. Hertli. 3-SAT faster and simpler – unique-SAT bounds for PPSZ hold in general. *SIAM Journal on Computing*, 43(2):718–729, 2014. `doi:10.1137/120868177`.

12 P. Idziak, P. Marković, R. McKenzie, M. Valeriote, and R. Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM Journal on Computing*, 39(7):3023–3037, June 2010.

13 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

14 P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.

15 P. Jonsson, V. Lagerkvist, G. Nordh, and B. Zanuttini. Complexity of SAT problems, clone theory and the exponential time hypothesis. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2013)*, pages 1264–1277, 2013. URL: `http://knowledgecenter.siam.org/0236-000094/`.

16 P. Jonsson, V. Lagerkvist, J. Schmidt, and H. Uppman. Relating the time complexity of optimization problems in light of the exponential-time hypothesis. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS-14)*, pages 408–419, Berlin, Heidelberg, 2014. Springer-Verlag.

17 D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM Journal on Computing*, 28:152–163, 1998.

18 V. Lagerkvist. Weak bases of Boolean co-clones. *Information Processing Letters*, 114(9):462–468, 2014.

19 V. Lagerkvist. *Mathematical Foundations of Computer Science 2015: 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, chapter Precise Upper and Lower Bounds for the Monotone Constraint Satisfaction Problem, pages 357–368. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

20 V. Lagerkvist. *Strong Partial Clones and the Complexity of Constraint Satisfaction Problems: Limitations and Applications.* PhD thesis, Linköping University, The Institute of Technology, 2016.

21 V. Lagerkvist and M. Wahlström. The power of primitive positive definitions with polynomially many variables. *To appear in Journal of Logic and Computation*, 2016.

22 V. Lagerkvist, M. Wahlström, and B. Zanuttini. Bounded bases of strong partial clones. In *Proceedings of the 45th International Symposium on Multiple-Valued Logic (ISMVL-2015)*, pages 189–194, 2015.

23 D. Lau. *Function Algebras on Finite Sets: Basic Course on Many-Valued Logic and Clone Theory (Springer Monographs in Mathematics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

24 B.A. Romov. The algebras of partial functions and their invariants. *Cybernetics*, 17(2):157–167, 1981.

25 T. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory Of Computing (STOC-78)*, pages 216–226. ACM Press, 1978.

26 H. Schnoor and I. Schnoor. Partial polymorphisms and constraint satisfaction problems. In N. Creignou, P. G. Kolaitis, and H. Vollmer, editors, *Complexity of Constraints*, volume 5250 of *Lecture Notes in Computer Science*, pages 229–254. Springer Berlin Heidelberg, 2008.

27 I. Schnoor. *The weak base method for constraint satisfaction.* PhD thesis, Gottfried Wilhelm Leibniz Universität, Hannover, Germany, 2008. URL: `http://edok01.tib.uni-hannover.de/edoks/e01dh08/559615132.pdf`.

28 K. Schölzel. *Clones of partial functions on finite sets.* PhD thesis, Universität Rostock, 2010.

29 K. Schölzel. Dichotomy on intervals of strong partial boolean clones. *Algebra Universalis*, 73(3-4):347–368, 2015.

30 M. Wahlström. *Algorithms, measures and upper bounds for satisfiability and related problems.* PhD thesis, Linköping University, TCSLAB – Theoretical Computer Science Laboratory, The Institute of Technology, 2007.

Uniformization Problems for Tree-Automatic Relations and Top-Down Tree Transducers*

Christof Löding[1] **and Sarah Winter**[2]

1 **Lehrstuhl für Informatik 7, RWTH Aachen University, Aachen, Germany**
2 **Lehrstuhl für Informatik 7, RWTH Aachen University, Aachen, Germany**

Abstract

For a given binary relation of finite trees, we consider the synthesis problem of deciding whether there is a deterministic top-down tree transducer that uniformizes the relation, and constructing such a transducer if it exists. A uniformization of a relation is a function that is contained in the relation and has the same domain as the relation. It is known that this problem is decidable if the relation is a deterministic top-down tree-automatic relation. We show that it becomes undecidable for general tree-automatic relations (specified by non-deterministic top-down tree automata). We also exhibit two cases for which the problem remains decidable. If we restrict the transducers to be path-preserving, which is a subclass of linear transducers, then the synthesis problem is decidable for general tree-automatic relations. If we consider relations that are finite unions of deterministic top-down tree-automatic relations, then the problem is decidable for synchronous transducers, which produce exactly one output symbol in each step (but can be non-linear).

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases tree transducers, tree automatic relation, uniformization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.65

1 Introduction

A uniformization of a (binary) relation is a function that selects for each element in the domain of the relation a unique image that is in relation with this element. Originally, uniformization has been studied in set theory, where the complexity of a class of definable relations is related with the complexity of uniformizations for these relations (see [18] for results of this kind). The basic uniformization question for a class $\mathcal{C}$ of relations is whether each relation from $\mathcal{C}$ has a uniformization in $\mathcal{C}$.

Automata provide a natural framework for defining relations (over words or trees), and uniformization problems in this setting have been studied since the early days of automata theory. Word relations defined by asynchronous finite automata [8], also called rational relations, were first shown to have rational uniformizations in [13, Theorem 3] (with many alternative and simplified proofs following later). For relations of infinite words that are accepted by synchronous finite automata, or equivalently definable in monadic second-order logic (MSO) over the structure consisting of natural numbers equipped with the successor relation, the uniformization property is shown in [19]. Over infinite trees, the uniformization property fails for MSO definable relations (corresponding to synchronous tree automata) [10, 2], while it has been shown recently that uniformization is possible for synchronous

* This work was supported by the DFG grant "Transducer Synthesis from Automaton Definable Specifications" (LO 1174/3-1)

relations over finite trees [14, 4]. These relations defined by synchronous automata are usually referred to as automatic, ω-automatic, tree-automatic and ω-tree-automatic relations (for finite words, infinite words, finite trees, and infinite trees, respectively).

In a more algorithmic setting, uniformization is often referred to as synthesis: The relation is viewed as a specification between inputs and outputs, and the function is supposed to be realized by a device that produces the output while processing the input. This means, that the class for the uniformizations is usually different from the class of the specifications, and the problem of interest is now the decision problem whether a given relation admits a uniformization in the desired class.

The classical setting, originating from Church's synthesis problem [3], is the one of infinite words. The specification is given by an ω-automatic relation (orginally in MSO), and the question is whether it can be uniformized by a synchronous sequential transducer that produces, letter by letter, an infinite output word while reading an infinite input word. The seminal paper of Büchi and Landweber [1] shows the decidability of this problem, and has been extended later to uniformizations by asynchronous sequential transducers [12, 11]. A detailed study of the synthesis of sequential transducers for asynchronous automata on finite words is provided in [6].

Our aim is to study these uniformization questions for relations over finite trees. Tree automata are used in many fields, for example as a tool for analyzing and manipulating rewrite systems or XML Schema languages (see [7]). Tree transformations that are realized by finite tree transducers thus become interesting in the setting of translations from one document scheme into another [17]. As class for the uniformizations we consider deterministic top-down tree transducers (D↓TTs), which are a natural extension of sequential transducers on words. A first result in this setting was obtained in [16], where we show that it is decidable whether a tree-automatic relation that is defined by a deterministic top-down tree automaton (D↓TA) can be uniformized by a D↓TT. A representation of the specification by a deterministic automaton model is essential in many synthesis algorithms for automata. A standard approach is to build a game in which the two players produce input and output. The aim of the output player is to ensure that the pair of input and output produced along a play satisfies the specification. This property is ensured in the game by simulating a deterministic automaton for the specification on the moves of the players. A winning strategy for the output player then corresponds to a uniformizer.

In this paper, we show that the synthesis problem for D↓TT from nondeterministic tree-automatic relations is indeed undecidable, showing that the nondeterminism does not only destroy the game theoretic approach (as sketched above) but makes the problem intractable in general. On the positive side, we prove two decidability results for restricted classes of uniformizers and specifications:

1. For nondeterministic tree-automatic relations uniformization by path-preserving D↓TTs is decidable. Intuitively, we call a D↓TT path-preserving if every node of the output tree is produced from a node of the input tree that is above or below the output node (this implies that each path-preserving D↓TT is in particular linear). For this class of uniformizers we can adapt the game theoretic approach by using guidable automata [5, 15] instead of deterministic automata for the specification.
2. If we restrict the specifications to unions of D↓TAs with disjoint domain, we obtain decidability for uniformizations by synchronous D↓TTs. We call a D↓TT synchronous if it produces one output symbol in each transition (but the transitions can be non-linear). While this is a rather specific result, it is the first decidability result for synthesis of transducers in which in the synthesized transducer may need to be non-linear.

The paper is structured as follows. In Section 2 we fix some basic definitions and terminology. In Section 3 we show undecidability for synthesis of D↓TTs from tree-automatic specifications, and the decidability results are presented in Section 4.

2 Preliminaries

Words and trees. The set of natural numbers containing zero is denoted by $\mathbb{N}$. For a set S, the powerset of S is denoted by 2^S. An *alphabet* Σ is a finite non-empty set of letters. A finite *word* is a finite sequence of letters. The set of all finite words over Σ is denoted by Σ^*. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, the empty word is denoted by ε. For $w = a_1 \dots a_n \in \Sigma^*$ for some $n \in \mathbb{N}$ and $a_1, \dots, a_n \in \Sigma$, let $w[i]$ denote the ith letter of w, i.e., $w[i] = a_i$. Furthermore, let $w[i,j]$ denote the infix from the ith to the jth letter of w, i.e., $w[i,j] = a_i \dots a_j$. We write $u \sqsubseteq w$ if there is some v such that $w = uv$ for $u, v \in \Sigma^*$. A subset $L \subseteq \Sigma^*$ is called *language* over Σ.

A *ranked alphabet* Σ is an alphabet where each letter $f \in \Sigma$ has a rank $rk(f) \in \mathbb{N}$. The set of letters of rank i is denoted by Σ_i. A tree domain *dom* is a non-empty finite subset of $(\mathbb{N} \setminus \{0\})^*$ such that *dom* is prefix-closed and for each $u \in (\mathbb{N} \setminus \{0\})^*$ and $i \in \mathbb{N} \setminus \{0\}$ if $ui \in dom$, then $uj \in dom$ for all $1 \leq j < i$. We speak of ui as successor of u for each $u \in dom$ and $i \in \mathbb{N} \setminus \{0\}$.

A (finite Σ-labeled) *tree* is a pair $t = (dom_t, val_t)$ with a mapping $val_t : dom_t \to \Sigma$ such that for each node $u \in dom_t$ the number of successors of u is a rank of $val_t(u)$. The height h of a tree t is the length of its longest path, i.e., $h(t) = max\{|u| \mid u \in dom_t\}$. The set of all Σ-labeled trees is denoted by T_Σ. A subset $T \subseteq T_\Sigma$ is called *tree language* over Σ.

A *subtree* $t|_u$ of a tree t at node u is defined by $dom_{t|_u} = \{v \in \mathbb{N}^* \mid uv \in dom_t\}$ and $val_{t|_u}(v) = val_t(uv)$ for all $v \in dom_{t|_u}$. In order to formalize concatenation of trees, we introduce the notion of special trees. A *special tree* over Σ is a tree over $\Sigma \uplus \{\circ\}$ such that $\circ$ occurs exactly once at a leaf. Given $t \in T_\Sigma$ and $u \in dom_t$, we write $t[\circ/u]$ for the special tree that is obtained by deleting the subtree at u and replacing it by $\circ$. Let S_Σ be the set of special trees over Σ. For $t \in S_\Sigma$ and $s \in T_\Sigma$ or $s \in S_\Sigma$ let the *concatenation* $t \cdot s$ be the tree that is obtained from t by replacing $\circ$ with s.

Let X_n be a set of n variables $\{x_1, \dots, x_n\}$ and Σ be a ranked alphabet. We denote by $T_\Sigma(X_n)$ the set of all trees over Σ which additionally can have variables from X_n at their leaves. We define X_0 to be the empty set, the set $T_\Sigma(\emptyset)$ is equal to T_Σ. Let $X = \bigcup_{n>0} X_n$. A tree from $T_\Sigma(X)$ is called *linear* if each variable occurs at most once. For $t \in T_\Sigma(X_n)$ let $t[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]$ be the tree that is obtained by substituting each occurrence of $x_i \in X_n$ by $t_i \in T_\Sigma(X)$ for every $1 \leq i \leq n$.

A tree from $T_\Sigma(X_n)$ such that all variables from X_n occur exactly once and in the order $x_1, \dots, x_n$ when reading the leaf nodes from left to right, is called *n-context* over Σ. Given an n-context, the node labeled by x_i is referred to as ith hole for every $1 \leq i \leq n$. A special tree can be seen as a 1-context, a tree without variables can be seen a 0-context. If C is an n-context and $t_1, \dots, t_n \in T_\Sigma(X)$ we write $C[t_1, \dots, t_n]$ instead of $C[x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n]$.

Tree automata. We fix our notations. For a detailed introduction to tree automata see e.g. [9] or [7]. Let $\Sigma = \bigcup_{i=0}^m \Sigma_i$ be a ranked alphabet. A *non-deterministic top-down tree automaton* (an N↓TA) over Σ is of the form $\mathcal{A} = (Q, \Sigma, Q_0, \Delta)$ consisting of a finite set of states Q, a set $Q_0 \subseteq Q$ of initial states, and $\Delta \subseteq \bigcup_{i=0}^m (Q \times \Sigma_i \times Q^i)$ is the transition relation. For $i = 0$, we identify $Q \times \Sigma_i \times Q^i$ with $Q \times \Sigma_0$. Let t be a tree and $\mathcal{A}$ be an N↓TA, a *run* of $\mathcal{A}$ on t is a mapping $\rho : dom_t \to Q$ compatible with Δ, i.e., $\rho(\varepsilon) \in Q_0$ and for each node $u \in dom_t$ with $i \geq 0$ successors $(\rho(u), val_t(u), \rho(u1), \ldots, \rho(ui)) \in \Delta$. A tree $t \in T_\Sigma$ is *accepted* if, and only if, there is a run of $\mathcal{A}$ on t. The tree language *recognized* by $\mathcal{A}$ is $T(\mathcal{A}) = \{t \in T_\Sigma \mid \mathcal{A} \text{ accepts } t\}$. A tree language $T \subseteq T_\Sigma$ is called *regular* if T is recognizable by a non-deterministic top-down tree automaton.

A top-down tree automaton $\mathcal{A} = (Q, \Sigma, Q_0, \Delta)$ is *deterministic* (a D↓TA) if the set Q_0 is a singleton set and for each $f \in \Sigma_i$ and each $q \in Q$ there is at most one transition $(q, f, q_1, \ldots, q_i) \in \Delta$. However, non-deterministic and deterministic top-down automata are not equally expressive. It is effectively decidable whether a regular tree language is top-down deterministic [9].

In Section 4.1 we use *guidable tree automata* [15]. The concept of guidable tree automata is that another tree automaton can act as a guide, meaning that a tree automaton $\mathcal{B}$ can guide a tree automaton $\mathcal{A}$ if an accepting run of $\mathcal{B}$ on a tree t can be translated deterministically into an accepting run of $\mathcal{A}$ on t.

Formally, an N↓TA $\mathcal{A}$ can be *guided by* an N↓TA $\mathcal{B}$ if there is a mapping $g : Q_\mathcal{A} \times \Delta_\mathcal{B} \to \Delta_\mathcal{A}$ such that $g(q, (p, a, p_1, \ldots, p_i)) = (q, a, q_1, \ldots, q_i)$ for some $q_1, \ldots, q_i \in Q_\mathcal{A}$, and for every accepting run ρ of $\mathcal{B}$ over a tree t, $g(\rho)$ is an accepting run of $\mathcal{A}$ over t, where $g(\rho) = \rho'$ is the unique run such that $\rho'(\varepsilon) = q_0^\mathcal{A}$, and for all $u \in dom_t : (\rho'(u), val_t(u), \rho'(u1), \ldots, \rho'(ui)) = g(\rho'(u), (\rho(u), val_t(u), \rho(u1), \ldots, \rho(ui)))$. An N↓TA $\mathcal{A}$ is called *guidable* if it can be guided by every N↓TA $\mathcal{B}$ such that $T(\mathcal{B}) \subseteq T(\mathcal{A})$.

Tree-automatic relations are defined by using tree automata over a product alphabet. For nodes that belong only to one of the trees one uses a padding symbol. Formally, let Σ, Γ be ranked alphabets and let $\Sigma_\perp = \Sigma \uplus \{\perp\}$, $\Gamma_\perp = \Gamma \uplus \{\perp\}$, where $\perp$ is a new symbol with rank 0. For an i-ary symbol $f \in \Sigma_\perp$ and a j-ary symbol $g \in \Gamma_\perp$, let $rk((f, g)) = max\{i, j\}$. The *convolution* of (t_1, t_2) with $t_1 \in T_\Sigma$, $t_2 \in T_\Gamma$ is the $\Sigma_\perp \times \Gamma_\perp$-labeled tree $t = t_1 \otimes t_2$ defined by $dom_t = dom_{t_1} \cup dom_{t_2}$, and $val_t(u) = (val^\perp_{t_1}(u), val^\perp_{t_2}(u))$ for all $u \in dom_t$, where $val^\perp_{t_i}(u) = val_{t_i}(u)$ if $u \in dom_{t_i}$ and $val^\perp_{t_i}(u) = \perp$ otherwise for $i \in \{1, 2\}$. As a special case, given $t \in T_\Sigma$, we define $t \otimes \perp$ to be the tree with $dom_{t \otimes \perp} = dom_t$ and $val_{t \otimes \perp}(u) = (val_t(u), \perp)$ for all $u \in dom_t$. Analogously, we define $\perp \otimes t$. We define the *convolution of a tree relation* $R \subseteq T_\Sigma \times T_\Gamma$ to be the tree language $T_R := \{t_1 \otimes t_2 \mid (t_1, t_2) \in R\}$.

We call a (binary) relation R *tree-automatic* if there exists a regular tree language T such that $T = T_R$. For ease of presentation, we say a tree automaton $\mathcal{A}$ recognizes R if it recognizes the convolution T_R and denote by $R(\mathcal{A})$ the induced relation R.

A *uniformization* of a relation $R \subseteq X \times Y$ is a function $f_R : X \to Y$ such that $(x, f_R(x)) \in R$ for all $x \in dom(R)$. We are interested in uniformizations of tree-automatic relations by deterministic top-down tree transducers.

Tree transducers. We consider top-down tree transducers, which read the tree from the root to the leaves and produce finite output trees in each step that are attached to the already produced output (see [7] for an introduction to tree transducers).

A *top-down tree transducer* (a ↓TT) is of the form $\mathcal{T} = (Q, \Sigma, \Gamma, q_0, \Delta)$ consisting of a finite set of states Q, a finite input alphabet Σ, a finite output alphabet Γ, an initial state $q_0 \in Q$, and Δ is a finite set of transition rules of the form $q(f(x_1, \ldots, x_i)) \to w[q_1(x_{j_1}), \ldots, q_n(x_{j_n})]$, or $q(x_1) \to w[q_1(x_1), \ldots, q_n(x_1)]$($\varepsilon$-transition), where $f \in \Sigma_i$, w is an n-context over Γ, $q, q_1, \ldots, q_n \in Q$ and variables $x_{j_1}, \ldots, x_{j_n} \in X_i$. A *deterministic* top-down tree transducer (a D↓TT) has no ε-transitions and no two rules with the same left-hand side.

A *configuration* of a top-down tree transducer is a triple $c = (t, t', \varphi)$ of an input tree $t \in T_\Sigma$, an output tree $t' \in T_{\Gamma \cup Q}$ and a function $\varphi : D_{t'} \to dom_t$, where

- $val_{t'}(u) \in \Gamma_i$ for each $u \in dom_{t'}$ with $i > 0$ successors, and

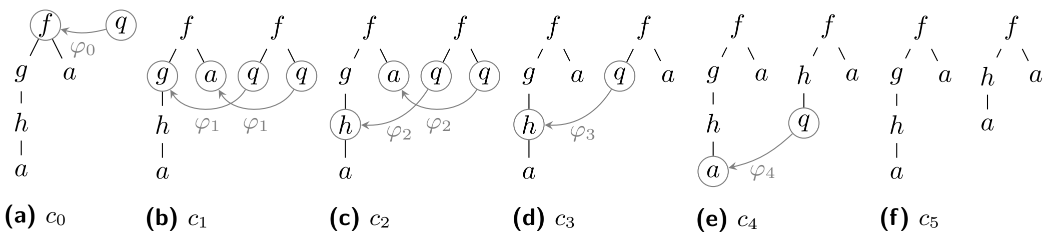


Figure 1 The configuration sequence c_0 to c_5 of $\mathcal{T}$ on t from Example 1.

- $val_{t'}(u) \in \Gamma_0$ or $val_{t'}(u) \in Q$ for each leaf $u \in dom_{t'}$, and
- $D_{t'} \subseteq dom_{t'}$ with $D_{t'} = \{u \in dom_{t'} \mid val_{t'}(u) \in Q\}$, i.e., φ maps every node from the output tree t' that has a state-label to a node of the input tree t.

Let $c_1 = (t, t_1, \varphi_1)$ and $c_2 = (t, t_2, \varphi_2)$ be configurations of a top-down tree transducer over the same input tree. We define a *successor relation* $\rightarrow_\mathcal{T}$ on configurations as usual by applying one rule. Figure 1 illustrates a configuration sequence explained in Example 1 below. Formally, for the application of a non-ε-rule, we define $c_1 \rightarrow_\mathcal{T} c_2 :\Leftrightarrow$

- There is a state-labeled node $u \in D_{t'}$ of the output tree t_1 that is mapped to a node $v \in dom_t$ of the input tree t, i.e., $\varphi_1(u) = v$, and
- there is a rule $val_{t_1}(u)\,(val_t(v)(x_1, \dots, x_i)) \rightarrow w[q_1(x_{j_1}), \dots, q_n(x_{j_n})] \in \Delta$ such that the output tree is correctly updated, i.e., $t_2 = t_1[\circ/u] \cdot w[q_1, \dots, q_n]$, and
- the mapping φ_2 is correctly updated, i.e., $\varphi_2(u') = \varphi_1(u')$ if $u' \in D_{t_1} \setminus \{u\}$ and $\varphi_2(u') = v.j_i$ if $u' = u.u_i$ with u_i is the ith hole in w.

Furthermore, let $\rightarrow^*_\mathcal{T}$ be the reflexive and transitive closure of $\rightarrow_\mathcal{T}$ and $\rightarrow^n_\mathcal{T}$ the reachability relation for $\rightarrow_\mathcal{T}$ in n steps. From here on, let φ_0 always denote the mapping $\varphi_0(\varepsilon) = \varepsilon$. A configuration (t, q_0, φ_0) is called *initial configuration* of $\mathcal{T}$ on t. A configuration $c = (t, t', \varphi)$ is said to be *reachable* in a computation of $\mathcal{T}$ on t, if $c_0 \rightarrow^*_\mathcal{T} c$, where c_0 is the initial configuration of $\mathcal{T}$ on t. The relation $R(\mathcal{T})$ induced by a top-down tree transducer $\mathcal{T}$ is $R(\mathcal{T}) = \{(t, t') \in T_\Sigma \times T_\Gamma \mid (t, q_0, \varphi_0) \rightarrow^*_\mathcal{T} (t, t', \varphi)\}$. For a (special) tree $t \in T_\Sigma$ or $t \in S_\Sigma$ let $\mathcal{T}(t) \subseteq T_{\Gamma \cup Q}$ be the set of *final transformed outputs* of a computation of $\mathcal{T}$ on t, that is the set $\{t' \mid (t, q_0, \varphi_0) \rightarrow^*_\mathcal{T} (t, t', \varphi)$ s.t. there is no successor configuration of $(t, t', \varphi)\}$. Note, we explicitly do not require that the final transformed output is a tree over Γ. In the special case that $\mathcal{T}(t)$ is a singleton set $\{t'\}$, we also write $\mathcal{T}(t) = t'$. The class of relations definable by ↓TTs is called the class of *top-down tree transformations*.

▶ **Example 1.** Let Σ be a ranked alphabet given by $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g, h\}$, and $\Sigma_0 = \{a\}$. Consider the ↓TT $\mathcal{T}$ given by $(\{q\}, \Sigma, \Sigma, \{q\}, \Delta)$ with $\Delta = \{\ q(a) \rightarrow a,\ q(g(x_1)) \rightarrow q(x_1),\ q(h(x_1)) \rightarrow h(q(x_1)),\ q(f(x_1, x_2)) \rightarrow f(q(x_1), q(x_2))\ \}$. For each $t \in T_\Sigma$ the transducer deletes all occurrences of g in t. Consider $t := f(g(h(a)), a)$. A possible sequence of configurations of $\mathcal{T}$ on t is $c_0 \rightarrow^5_\mathcal{T} c_5$ such that $c_0 := (t, q, \varphi_0)$ with $\varphi_0(\varepsilon) = \varepsilon$, $c_1 := (t, f(q, q), \varphi_1)$ with $\varphi_1(1) = 1$, $\varphi_1(2) = 2$, $c_2 := (t, f(q, q), \varphi_2)$ with $\varphi_2(1) = 11$, $\varphi_2(2) = 2$, $c_3 := (t, f(q, a), \varphi_3)$ with $\varphi_3(1) = 11$, $c_4 := (t, f(h(q), a), \varphi_4)$ with $\varphi_4(11) = 111$, and $c_5 := (t, f(h(a), a), \varphi_5)$. A visualization of this sequence is shown in Figure 1.

We consider two restricted types of top-down tree transducers. The first type are *transducers with bounded (output) delay*. Intuitively, delay occurs in a computation of a transducer if there is a difference between the number of produced output symbols and read input symbols. If the output is behind this is called output delay. More formally, in a configuration (t, t', φ) occurs *delay* d w.r.t. a node $u \in D_{t'}$ if the absolute value of $|\varphi(u)| - |u|$

equals d. We speak of *output delay* if $|\varphi(u)| - |u|$ is a positive integer. We say the *delay* (resp. *output delay*) in a $\downarrow$TT $\mathcal{T}$ is *bounded* by k, if there is a $k \in \mathbb{N}$ such that for every reachable configuration c of $\mathcal{T}$ the maximal delay (resp. output delay) that occurs in c is at most k. We speak of *synchronous* $\downarrow$TTs if the delay is bounded by 0. Consider $\mathcal{T}$ from Example 1 and the configuration sequence of $\mathcal{T}$ given in Example 1. In c_2 occurs output delay 1 resp. 0 w.r.t. node 1 resp. 2 of the output tree. It is easy to see that the transducer has unbounded output delay, because it deletes all occurrences of g in an input tree.

The second restricted type of top-down tree transducer concerns the ability to copy and swap subtrees. A $\downarrow$TT is *linear* if all the trees in the transitions are linear. In Section 4.1, we consider a special case of linear $\downarrow$TTs called *path-preserving*. Intuitively, a $\downarrow$TT is said to be path-preserving if in every computation the read input and correspondingly produced output are always on the same path, i.e., every node of the output tree is produced from a node of the input tree that is above or below the output node. More formally, in every reachable configuration (t, t', φ) of the transducer it holds either $u \sqsubseteq \varphi(u)$ or $\varphi(u) \sqsubseteq u$ for every node $u \in D_{t'}$. We refer to this kind of $\downarrow$TTs as P$\downarrow$TTs for short.

3 Undecidability Results

▶ **Theorem 2.** *It is undecidable whether a given tree-automatic relation has a uniformization by a deterministic top-down tree transducer.*

Proof Sketch. We give a reduction from the halting problem for Turing machines (TM). Given a TM M, our goal is to describe a tree-automatic specification R_M which can only be realized by a deterministic top-down tree transducer if M does not halt on the empty input tape. In order to save space, we draw trees from left to right rather than from top to bottom. For explaining the idea, we provide for a given Turing machine a specification S and a uniformizer θ and a D$\downarrow$TT-realizable transformation θ such that θ uniformizes S if, and only if, M does not halt on the empty tape. For the full proof, the specification and the uniformizer have to be adapted such that θ becomes the only candidate for uniformizer of S, which then implies the undecidability of the existence of a uniformizer.

In the following, we explain the simple versions of S and θ. Let Q_M denote the state set of M, q_0 denote the initial state of M, and Γ_M denote the tape alphabet of M. We can represent a configuration c of M, as a unary tree, i.e., as a string, of the form $u_1 - \cdots - u_k - q - v_1 \cdots - u_\ell$, where $u_1, \ldots, u_k, v_1, \ldots, v_\ell \in \Gamma_M$, $u_1 \ldots u_k v_1 \ldots v_\ell$ is the content of the tape of M, $q \in Q_M$ is the current control state of M, and the head of M is on v_1.

We start with the first step. Concerning the specification S, we are interested in pairs (t, t') of trees over $Q_M \cup \Gamma_M \cup \{f, a\}$ which have the form

$$\left(\begin{array}{ccccccc} f & - & f & - \cdots - & f & - & a \\ | & & | & & | & & \\ c_0 & & c_1 & & c_n & & \end{array}\ ,\ \begin{array}{ccccccc} f & - & f & - \cdots - & f & - & a \\ | & & | & & | & & \\ k_1 & & k_2 & & k_m & & \end{array}\right),$$

where $m \geq n$, each c_i (resp. k_i) is a configuration of M, c_0 is the initial configuration of M on the empty tape and c_n is a halting configuration of M. Note that the numbering of the c_i starts with 0 and the numbering of the k_i with 1, this is intended. Such a pair of trees is part of the specification S if it additionally satisfies the following: There is an $i \in \{0, \ldots, n-1\}$ such that $succ(c_i) \neq k_{i+1}$, where $succ(c_i)$ is the successor configuration of c_i.

The specification S is tree-automatic. Note that for a pair (t, t') of the correct form, the configurations c_i and k_{i+1} overlap for each $i \in \{0, \ldots, n-1\}$ in $t \otimes t'$. A tree-automaton can guess a branch and verify that $succ(c_i) \neq k_{i+1}$ holds.

Now, we consider the function $\theta\colon dom(S) \to T_{Q_M \cup \Gamma_M \cup \{f,a\}}$ defined by

$$\begin{array}{ccccccc} f & - & f & - \cdots - & f & - & a \\ | & & | & & | & & \\ c_0 & & c_1 & & c_n & & \end{array} \quad \mapsto \quad \begin{array}{ccccccc} f & - & f & - \cdots - & f & - & a \\ | & & | & & | & & \\ c_1 & & c_2 & & c_n & & \end{array} .$$

Assuming that a transducer is only given input trees that have the desired form, this function is realizable by a deterministic top-down tree transducer, e.g., by some transducer that produces no output in the first step, continues at the right child and then simply copies the rest of the input tree.

Assume that M does not halt on the empty input tape and consider an input tree $t \in dom(S)$, then there are configurations c_i and c_{i+1} such that c_{i+1} is not the successor configuration of c_i. The transformation θ yields $c_{i+1} = k_{i+1}$, it follows that $succ(c_i) \neq k_{i+1}$, i.e., $(t, \theta(t)) \in S$. Conversely, assume that M halts on the empty input tape. Consider an input tree $t \in dom(S)$ such that $c_0 c_1 \cdots c_n$ is the halting configuration sequence. It follows that $k_{i+1} = succ(c_i) = c_{i+1}$ for all $i \in \{0, \ldots, n-1\}$, i.e., $(t, \theta(t)) \notin S$. Clearly, S is uniformized by θ if, and only if, M does not halt on the empty input tape.

However, the specification S does not suffice to enforce that this kind of transformation is the only possible uniformizer. This can be achieved by extending the alphabet and the specification. ◀

From the undecidability proof one can derive that the uniformization problems remain undecidable if we restrict the D↓TTs, as stated in the following two theorems. Together with the decidability result from Section 4.1 this gives an almost complete picture of the frontier between decidability and undecidability (for the case of all tree-automatic relations as specifications, and subclasses of D↓TTs as uniformizers).

▶ **Theorem 3.** *It is undecidable whether a given tree-automatic relation has a uniformization by a linear deterministic top-down tree transducer with delay bounded by 1.*

▶ **Theorem 4.** *It is undecidable whether a given tree-automatic relation has a uniformization by a synchronous deterministic top-down tree transducer.*

4 Decidability Results

In the previous section we have seen that the uniformization problem for general tree-automatic specifications is undecidable. In order to regain decidability of the uniformization problem for non-deterministic top-down specifications we present two approaches. In Section 4.1, we consider general non-deterministic top-down specifications and restrict the uniformizer, whereas in Section 4.2 we consider a restricted class of non-deterministic top-down specifications and ask whether there is a synchronous uniformizer.

4.1 Path-preserving uniformization

In this section, we consider general non-deterministic tree relations and restrict the uniformizer; we are looking for a uniformization by a deterministic path-preserving top-down transducer. We solve the following uniformization problem.

▶ **Theorem 5.** *It is decidable whether a given tree-automatic relation has a uniformization by a deterministic path-preserving top-down tree transducer.*

In the following we show that deciding whether a general non-deterministic top-down tree relation has a path-preserving uniformization reduces to deciding the winner in a safety game between two players. We show that the use of guidable tree automata [15] for the specifications makes it feasible to adapt a decision procedure presented in [16], where the uniformization problem for deterministic top-down tree relations was reduced to deciding the winner in a safety game.

Before we present the decision procedure, we need to fix some notations. Given $\Sigma = \bigcup_{i=0}^{m} \Sigma_i$, let $\mathrm{dir}_\Sigma = \{1, \dots, m\}$ be the set of directions compatible with Σ. For $\Sigma = \bigcup_{i=0}^{m} \Sigma_i$, the set Path_Σ of labeled paths over Σ is defined inductively by:

- ε is a labeled input path and each $f \in \Sigma$ is a labeled input path,
- given a labeled input path $\pi = x \cdot f$ with $f \in \Sigma_i$ $(i > 0)$ over Σ, then $\pi \cdot jg$ with $j \in \{1, \dots, i\}$ and $g \in \Sigma$ is a labeled input path.

For $\pi \in \mathrm{Path}_\Sigma$, we define the path $path(\pi) \in \mathrm{dir}_\Sigma^*$ and the word $labels(\pi) \in \Sigma^*$ induced by π inductively by:

- if $\pi = \varepsilon$ or $\pi = f$, then $path(\varepsilon) = path(f) = \varepsilon$, $labels(\varepsilon) = \varepsilon$ and $labels(f) = f$,
- if $\pi = x \cdot jf$ with $j \in \mathrm{dir}_\Sigma$, $f \in \Sigma$, then $path(\pi) = path(x) \cdot j$, $labels(\pi) = labels(x) \cdot f$.

The length $|| \; ||$ of a labeled path over Σ is the length of the word induced by its path, i.e., $||\pi|| = |labels(\pi)|$.

For $\pi \in \mathrm{Path}_\Sigma$ let $T_\Sigma^\pi := \{t \in T_\Sigma \mid val_t\big(path(\pi)[1, (i{-}1)]\big) = labels(\pi)[i]$ for $1 \leq i \leq ||\pi||\}$ be the set of trees t over Σ such that π is a prefix of a labeled path through t. For a tree-automatic relation $R \subseteq T_\Sigma \times T_\Gamma$ recognized by an N↓TA $\mathcal{A}$, $\pi \in \mathrm{Path}_\Sigma$ and $q \in Q_\mathcal{A}$ let $R^\pi := \{(t, t') \in R \mid t \in T_\Sigma^\pi\}$ and $R_q^\pi := \{(t, t') \in R(\mathcal{A}_q) \mid t \in T_\Sigma^\pi\}$.

Since we consider labeled paths through trees, it is convenient to define the notion of convolution also for labeled paths. For a labeled path $x \in \mathrm{Path}_\Sigma$ with $||x|| > 0$, let $dom_x := \{u \in \mathrm{dir}_\Sigma^* \mid u \sqsubseteq path(x)\}$ and $val_x : dom_x \to \Sigma$, where $val_x(u) = labels(x)[i]$ if $u \in dom_x$ with $|u| = i + 1$. Let $x \in \mathrm{Path}_\Sigma$, $y \in \mathrm{Path}_\Gamma$ with $path(y) \sqsubseteq path(x)$ or $path(x) \sqsubseteq path(y)$, then the *convolution* of x and y is $x \otimes y$ defined by $dom_{x \otimes y} = dom_x \cup dom_y$, and $val_{x \otimes y}(u) = (val_x^\perp(u), val_y^\perp(u))$ for all $u \in dom_{x \otimes y}$, where $val_x^\perp(u) = val_x(u)$ if $u \in dom_x$ and $val_x^\perp(u) = \perp$ otherwise, analogously defined for $val_y^\perp(u)$.

Furthermore, it is useful to relax the notion of runs to labeled paths. Let $x \in \mathrm{Path}_\Sigma$, $y \in \mathrm{Path}_\Gamma$ such that $x \otimes y$ is defined, i.e., $path(y) \sqsubseteq path(x)$ or $path(x) \sqsubseteq path(y)$. We define the run of $\mathcal{A}$ on $x \otimes y$ such that it maps all nodes from $dom_{x \otimes y}$ as well as all nodes that are a direct successor of a node from $dom_{x \otimes y}$ to a state of $\mathcal{A}$. Formally, let the (partial) *run* of $\mathcal{A}$ on $x \otimes y$ be the partial function $\rho : \mathrm{dir}_\Sigma^* \to Q_\mathcal{A}$ such that $\rho(\varepsilon) = q_0^\mathcal{A}$, and for each $u \in dom_{x \otimes y}$: if $q := \rho(u)$ is defined and there is a transition $(q, val_{x \otimes y}(u), q_1, \dots, q_i) \in \Delta_\mathcal{A}$, then $\rho(u.j) = q_j$ for all $j \in \{1, \dots, i\}$. Let $path(x \otimes y) = v$ and $i \in \mathrm{dir}_\Sigma$. Shorthand, we write $\mathcal{A} : q_0^\mathcal{A} \xrightarrow{x \otimes y}_i q$, if $q := \rho(vi)$ is defined. We write $\mathcal{A} : q_0^\mathcal{A} \xrightarrow{x \otimes y} Acc$ if $rk(val_{x \otimes y}(v)) = 0$ and $(\rho(v), val_{x \otimes y}(v)) \in \Delta_\mathcal{A}$ to indicate that the (partial) run ρ of $\mathcal{A}$ on $x \otimes y$ is accepting.

We explicitly state a simple lemma that is used in several places.

▶ **Lemma 6** ([16]). *Given a ↓TA $\mathcal{A}$ and a state q of $\mathcal{A}$, the following properties are decidable:*

1. $\forall t \in T_\Sigma : t \otimes \perp \in T(\mathcal{A}_q)$,
2. $\exists t' \in T_\Gamma : \perp \otimes t' \in T(\mathcal{A}_q)$,
3. $\exists t' \in T_\Gamma \, \forall t \in T_\Sigma : t \otimes t' \in T(\mathcal{A}_q)$.

Towards the decision procedure, we consider the relationship between the delay that a transducer introduces and uniformizability. Intuitively, if a specification is uniformized by a transducer such that the uniformizer introduces long delays between outputs, then only one path in an input tree is relevant in order to determine an output tree. We express this property by introducing the term path-recognizable function, meaning that there is a D↓TT that first deterministically reads a path from the root to a leaf in an input tree and then outputs a matching output tree. Note that such a uniformizer is always path-preserving.

Formally, we say a relation $R \subseteq T_\Sigma \times T_\Gamma$ is *uniformizable by a path-recognizable function*, if there exists a D↓TT $\mathcal{T}$ that uniformizes R such that $\Delta_\mathcal{T}$ only contains transitions of the following form:

1. $q(f(x_1,\dots,x_i)) \to q'(x_j)$, or
2. $q(a) \to t$,

where $f \in \Sigma_i$, $i > 0$, $a \in \Sigma_0$, $q, q' \in Q_\mathcal{T}$ and $j \in \{1,\dots,i\}$ and $t \in T_\Gamma$.

This notion was introduced in [16], where it was shown to be decidable whether a top-down deterministic relation can be uniformized by a path-recognizable function. Using guidable automata, the result carries over to general tree-automatic relations.

▶ **Theorem 7.** *It is decidable whether a given tree-automatic relation can be uniformized by a path-recognizable function.*

Given a specification, we can show that there exists a computable bound with the following property: If it is necessary for a D↓PTT to introduce delay that exceeds the bound in order to satisfy the specification, then either the remaining specification has a simple uniformization by a path-recognizable function, which is decidable by the above theorem, or is not D↓PTT-uniformizable.

Towards the definition of the game, we need one more notion. We introduce a relation that contains state transformations of a given specification automaton that a labeled path together with some output sequence on this path induces. However, we are only interested in the result of a state transformation if it suffices for a uniformizer to read this labeled path segment in an input tree to (partially) determine a matching output tree. Formally, let $x \in \mathrm{Path}_\Sigma$, $y \in \mathrm{Path}_\Gamma$ and $i \in \mathrm{dir}_\Sigma$ such that $x \otimes y$ is defined. We define the relation $\tau_{xi,y} \subseteq Q_\mathcal{A} \times Q_\mathcal{A}$ such that $(q,q') \in \tau_{xi,y}$ if there is a partial run ρ_q of $\mathcal{A}_q$ on $x \otimes y$ with $\mathcal{A}_q : q \xrightarrow{x\otimes y}_i q'$ and for each uj with $u \in dom_{x\otimes y}, uj \not\sqsubseteq path(x \otimes y)i$, and $j \in \{1,\dots,rk\big((val_x^\perp(u), val_y^\perp(u))\big)\}$ holds

- if $r := \rho_q(uj)$ and $j \leq rk(val_x^\perp(u)), rk(val_y^\perp(u))$, then there exists $t' \in T_\Gamma$ such that $t \otimes t' \in T(\mathcal{A}_r)$ for all $t \in T_\Sigma$, and
- if $r := \rho_q(uj)$ and $rk(val_y^\perp(u)) < j \leq rk(val_x^\perp(u))$, then $t \otimes \perp \in T(\mathcal{A}_r)$ for all $t \in T_\Sigma$, and
- if $r := \rho_q(uj)$ and $rk(val_x^\perp(u)) < j \leq rk(val_y^\perp(u))$, then there exists $t' \in T_\Gamma$ such that $\perp \otimes t' \in T(\mathcal{A}_r)$.

Lemma 6 implies that it is decidable whether $(q,q') \in \tau_{xi,y}$. Basically, if q is in the domain of $\tau_{xi,y}$, then there exists a fixed (partial) output tree $s' \in S_\Gamma^{yio}$ such that for each input tree $t \in T_\Sigma^x \cap dom(T(\mathcal{A}_q))$ there exists some $t' \in T_\Gamma$ such that $t \otimes (s' \cdot t') \in T(\mathcal{A}_q)$.

Now, we are ready to show that the uniformization problem posed in this section reduces to deciding the winner in a safety game, provided that the specification is given by a guidable automaton. The game is played between In and Out on a game graph parameterized by k, where In can follow any path from the root to a leaf in an input tree such that In plays one input symbol at a time. Out can either react with an output symbol, or delay the output a bounded number of times (at most $2k$ times) and react with a direction in which In should continue with his input sequence. As stated after Theorem 7, when the output delay increases to a computable bound, then uniformization is either impossible or can be realized by a path-recognizable function (Out then wins automatically, see o4. in the construction below). To make the decision procedure sound, the parameter k has to be chosen as this bound.

Given a tree-automatic relation $R \subseteq T_\Sigma \times T_\Gamma$, we assume its domain to be deterministic, otherwise no deterministic ↓TT can recognize the domain. Let R be recognized by a guidable

N↓TA $\mathcal{A}$ and let $dom(R)$ be recognized by a D↓TA $\mathcal{B}$. Formally, the game graph $G^k_{\mathcal{A},\mathcal{B}}$ is constructed as follows.

- $V_{\mathsf{In}} = \{(p, q, \pi j) \in Q_{\mathcal{B}} \times Q_{\mathcal{A}} \times \text{Path}_\Sigma \cdot \text{dir}_\Sigma \mid \|\pi\| \leq 2k+1, \pi \in \text{Path}_\Sigma, j \in \text{dir}_\Sigma\} \cup 2^{Q_{\mathcal{B}} \times Q_{\mathcal{A}}}$ is the set of vertices of player In including the initial vertex $\{(q_0^{\mathcal{B}}, q_0^{\mathcal{A}})\}$.
- $V_{\mathsf{Out}} = \{(p, q, \pi) \in Q_{\mathcal{B}} \times Q_{\mathcal{A}} \times \text{Path}_\Sigma \mid \|\pi\| \leq 2k+1\}$ is the set of vertices of player Out.
- From a vertex of In the following moves are possible:
  i1. $(p, q, \pi j) \to (p, q, \pi j f)$ for each $f \in \Sigma$ such that $\mathcal{B}\colon p \xrightarrow{\pi}_j p'$ and there exists $(p', f, p_1, \ldots, p_i) \in \Delta_{\mathcal{B}}$ if $\|\pi\| < 2k+1$ (delay; In chooses the next input symbol)
  i2. $\{(p_1, q_1), \ldots, (p_n, q_n)\} \to (p_j, q_j, f)$ for each $f \in \Sigma$ such that there is $(p_j, f, p_j^1, \ldots, p_j^i) \in \Delta_{\mathcal{B}}$ and each $j \in \{1, \ldots, n\}$ (no delay; In chooses the next direction and input symbol)
- From a vertex of Out the following moves are possible:
  o1. $(p, q, f) \xrightarrow{r} \{(p_1, q_1), \ldots, (p_i, q_i)\}$ if there is $r = (q, (f, g), q_1, \ldots, q_n) \in \Delta_{\mathcal{A}}$, $(p, f, p_1, \ldots, p_i) \in \Delta_{\mathcal{B}}$, $f \in \Sigma$ is i-ary, $g \in \Sigma_\perp$ is j-ary, $n = max\{i, j\}$, and if $j > i$ there exist trees $t_{i+1}, \ldots, t_j \in T_\Gamma$ such that $\perp \otimes t_\ell \in T(\mathcal{A}_{q_l})$ for all $i < \ell \leq j$.
  (no delay; Out applies a transition; Out can pick output trees for all directions where the input has ended; In can continue from the other directions)
  Note, if $f \in \Sigma_0$, i.e., the input symbol is a leaf, then the next reached vertex is $\emptyset \in V_{\mathsf{In}}$, which is a terminal vertex.
  o2. $(p, q, fj\pi) \xrightarrow{r} (p_j, q_j, \pi)$ if there is $r = (q, (f, g), q_1, \ldots, q_n) \in \Delta_{\mathcal{A}}$ such that $(q, q_j) \in \tau_{fj,g}$ and $(p, f, p_1, \ldots, p_i) \in \Delta_{\mathcal{B}}$.
  (delay; Out applies a transition, removes the leftmost input symbol and advances in direction of the labeled path ahead; Out can pick output trees for all divergent directions)
  o3. $(p, q, \pi j f) \to (p, q, \pi j f j')$ for each $j' \in \{1, \ldots, i\}$ for $f \in \Sigma_i$ if $\|\pi j f\| < k+1$
  (Out delays and chooses a direction from where In should continue)
  o4. $(p, q, \pi) \to (p, q, \pi)$ if R_q^π is uniformizable by a path-recognizable function.
  (Out stays in this vertex and wins)

Note that the game graph can effectively be constructed, because Lemma 6 and Theorem 7 imply that it is decidable whether the edge constraints are satisfied.

The desired winning condition expresses that player Out loses the game if the input can be extended, but no valid output can be produced. This is represented in the game graph by a set of bad vertices B that contains all vertices of Out with no outgoing edges. If one of these vertices is reached during a play, Out loses the game. Thus, we define $\mathcal{G}^k_{\mathcal{A},\mathcal{B}} = (G^k_{\mathcal{A},\mathcal{B}}, V \setminus B)$ as safety game for Out.

The following two lemmata show that from the existence of a winning strategy a top-down tree transducer that uniformizer the relation can be obtained and vice versa.

▶ **Lemma 8.** *Given k, if Out has a winning strategy in $\mathcal{G}^k_{\mathcal{A},\mathcal{B}}$, then R is D↓PTT-uniformizable.*

The key idea in order to lift the proof in [16] from deterministic to general non-deterministic specifications is, given a guidable automaton for the specification, to turn a uniformizer into a guide for the specification automaton in order to construct a winning strategy.

▶ **Lemma 9.** *If R is D↓PTT-uniformizable, then Out has a winning strategy in $\mathcal{G}^k_{\mathcal{A},\mathcal{B}}$, where k is a number effectively computable from $\mathcal{A}$.*

As a consequence of Lemmata 8 and 9 and the fact that a winning strategy for Out in $\mathcal{G}^k_{\mathcal{A},\mathcal{B}}$ can effectively be computed, together with the fact that for each tree-automatic relation a guidable N↓TA can effectively be constructed, see [15], we immediately obtain Theorem 5.

4.2 Union of top-down deterministic specifications

In this section, we assume that $R \subseteq T_\Sigma \times T_\Gamma$ is given as the union $\bigcup_{i=1}^{n} R_i$ of n relations with pairwise disjoint domains, where each R_i is recognized by a D↓TA $\mathcal{A}_i$ and its domain is recognized by a D↓TA $\mathcal{B}_i$. Furthermore, we assume that the domain of the relation is D↓TA-recognizable, otherwise there exists no uniformization by a deterministic top-down tree transducer.

▶ **Example 10.** Let Σ be an input alphabet given by $\Sigma_1 = \{h\}$ and $\Sigma_0 = \{c, d\}$ and let Γ be an output alphabet given by $\Gamma_2 = \{f\}$, $\Gamma_1 = \{h\}$ and $\Gamma_0 = \{c, d\}$. We consider the relation $R \subseteq T_\Sigma \times T_\Gamma$ defined by $\{(h(t), f(t, t')) \mid t, t' \in T_\Sigma$ such that t and t' have the same leaf symbol$\}$. This specification can be obtained by the union of two deterministic top-down specifications, one specification for each leaf symbol. Clearly, a deterministic top-down transducer can realize the specification by producing $f(t, t)$ for a unary input tree $h(t)$, e.g., by starting with $q_0(h(x_1)) \to f(q(x_1), q(x_1))$. However, there is no linear synchronous uniformizer for R, because in the first step a linear D↓TT would have to pick for either the right or the left subtree an output tree with a fixed leaf symbol. As the actual leaf symbol of the input tree is yet unknown it is not possible to fix a correct output tree.

We provide a decision procedure for the following problem.

▶ **Theorem 11.** *It is decidable whether the union of D↓TA-recognizable relations with pairwise disjoint D↓TA-recognizable domains has a uniformization by a synchronous deterministic top-down tree transducer.*

We show that the existence of a synchronous uniformizer for such a relation is a regular property over infinite trees that can be checked by a parity tree automaton. For an introduction to parity tree automata, see e.g. [20]. We define a regular infinite tree, given as the unfolding of a finite graph, such that each vertex of the infinite tree represents a node in an input tree together with a set of output nodes produced from this input node. Since the uniformizer might be non-linear, output at different positions in the output tree can depend on the same position in the input tree. Our construction bounds the number of required output choices by making the choice only depending on the state transformation that the current output sequences together with the input sequence induces.

Before we formally define the finite graph, we describe its components. Recall, $R = \bigcup_{i=1}^{n} R_i$, where R_i is recognized by a D↓TA $\mathcal{A}_i$ and $dom(R)$ is recognized by a D↓TA $\mathcal{D}$. The graph keeps track of the state of $\mathcal{D}$ on the input, and the states of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ on the produced output. For the latter we use vectors with n elements. We define a function λ_ℓ that returns the ℓth element of a vector, for each $1 \leq \ell \leq n$. Let L denote such a vector, then $\lambda_\ell(L)$ stores the information w.r.t. $\mathcal{A}_\ell$. We model that read input and produced output can be on the same or on divergent paths as follows: In case that input and output are on the same path, $\lambda_\ell(L)$ is the state of $\mathcal{A}_\ell$ on the combined input sequence and output sequence. In case that the output is mapped to a divergent path, $\lambda_\ell(L)$ is a set of states of $\mathcal{A}_\ell$ that is obtained by combining all possible input sequences with the produced output sequence. Now we are ready to formally define the graph $\mathcal{G}$:

- From a vertex v of the form $(p, \{L_1, \ldots, L_m\})$, where p is a state of $\mathcal{D}$ and each L_j is a vector of states resp. sets of states over $\mathcal{A}_1, \ldots, \mathcal{A}_n$, the following edges exist:
  - $v \to (v, f)$ if there is $(p, f, p_1 \ldots, p_i) \in \Delta_\mathcal{D}$ (edges for every possible input symbol)
- An edge $((p, \{L_1, \ldots, L_m\}), f) \xrightarrow{o_1, \ldots, o_m} [(p_1, Q_1), \ldots, (p_i, Q_i)]$ defining output choices $o_1, \ldots, o_m$ exists if $(p, f, p_1 \ldots, p_i) \in \Delta_\mathcal{D}$ and the following conditions hold:
  - $o_j \in \Gamma(X_i)$ for each $1 \leq j \leq m$, and
    (for each L_j an output o_j consisting of one symbol and directions to continue is chosen)

- the set Q_d is constructed as follows for each $1 \leq d \leq i$:
  if for output $o_j = g(x_{j_1}, \dots, x_{j_r})$ there is $k \in \{1, \dots, r\}$ with $j_k = d$,
  (the kth child of the output o_j depends on the dth child of the input)
  we add a vector V_k to Q_d, where the component $\lambda_\ell(V_k)$ referring to $\mathcal{A}_\ell$ is build up from $\lambda_\ell(L_j)$ and o_j as follows:
  * if $\lambda_\ell(L_j) \in Q_{\mathcal{A}_\ell}$, say $q \in Q_{\mathcal{A}_\ell}$, (input and output are at the same position)
    and there is $(q, (f, g), q_1, \dots, q_{max\{rk(f),rk(g)\}}) \in \Delta_{\mathcal{A}_\ell}$,
    then $\lambda_\ell(V_k) = \begin{cases} q_k & \text{if } d = k, \quad \text{(input and output continue in the same direction)} \\ \{q_k\} & \text{otherwise. (input and output continue in divergent directions)} \end{cases}$
    (the corresponding transition in $\mathcal{A}_\ell$ is applied)
  * if $\lambda_\ell(L_j) \in 2^{Q_{\mathcal{A}_\ell}}$, (input and output are on divergent paths)
    then set $\lambda_\ell(V_k)$ to $\emptyset$ and for each $q \in \lambda_\ell(L_j)$ and each $f' \in \Sigma$ such that there is $(q, (f', g), q_1, \dots, q_{max\{rk(f'),rk(g)\}}) \in \Delta_{\mathcal{A}_\ell}$, add q_k to $\lambda_\ell(V_k)$.
    (all possibly reachable states in $\mathcal{A}_\ell$ are collected)
- From $[v_1, \dots, v_i]$ an edge to v_j exists for all $1 \leq j \leq i$. (edges to all directions)
- The initial vertex is $(p_0, \{L\})$, where $L = [q_0^{\mathcal{A}_1}, \dots, q_0^{\mathcal{A}_n}]$ and p_0 is the initial state of $\mathcal{D}$.

Now that we have defined $\mathcal{G}$, we consider the unfolding $\mathcal{H}$ of $\mathcal{G}$ which is a regular infinite tree. Consequently, each vertex of $\mathcal{H}$ is associated with a labeled path, interpreted as an input sequence π, and additionally it is associated with a bounded number of labeled paths, interpreted as output sequences produced by a transducer while reading the input sequence π. Note that different vertices of $\mathcal{H}$ may represent the same input sequence, but differ in the associated output sequences. This is a regular infinite tree that has the desired property, namely, each input sequence together with a (sufficiently large) number of possible output sequences is represented in the tree.

Our goal is to construct a parity tree automaton, whose tree language is non-empty iff R has a uniformization by a synchronous deterministic top-down tree transducer. The idea is to annotate $\mathcal{H}$ with an output strategy σ. The strategy selects for each node of the form (v, f) with $f \in \Sigma$ one child, i.e., σ fixes an output choice. Let $\mathcal{H}^\frown\sigma$ denote the tree $\mathcal{H}$ with annotations encoding σ. Given $\mathcal{H}^\frown\sigma$ and some input tree $t \in dom(R)$, the output choices defined by σ identify a unique output tree that a D↓TT can produce while reading t. For an input tree t, let $\sigma(t)$ denote the corresponding output tree. The strategy σ corresponds to a uniformizer if for all $t \in dom(R)$ holds that $(t, \sigma(t)) \in R$. The following lemma shows that the set of trees $\mathcal{H}^\frown\sigma$ such that σ corresponds to a uniformizer is a regular set of trees.

▶ **Lemma 12.** *There exists a parity tree automaton $\mathcal{C}$ that accepts exactly those trees $\mathcal{H}^\frown\sigma$ such that $(t, \sigma(t)) \in R$ for all $t \in dom(R)$.*

The next lemma shows that the uniformization problem posed in this section reduces to deciding the emptiness problem for $\mathcal{C}$. It directly implies Theorem 11 because emptiness of parity tree automata is decidable (see [20]).

▶ **Lemma 13.** *The tree language $T(\mathcal{C})$ is non-empty if, and only if, R has a uniformization by a synchronous deterministic top-down tree transducer.*

5 Conclusion

We have considered uniformization of tree-automatic relations by D↓TTs. Using the subclasses of bounded-delay, linear, and path-preserving D↓TTs, we have obtained an almost complete picture of the frontier between decidability and undecidability. We have also presented a class of tree-automatic relations for which the uniformization problem is decidable but requires, in general, non-linear uniformizers.

As further research questions it would be interesting to extend the class of specifications beyond those of tree-automatic relations. In [6] decidability results for word transformations have been obtained for deterministic rational relations, and for uniformization questions in which the delay of the uniformizer is related to the one of the specification. We plan to study extensions of these ideas from words to trees.

References

1 J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. doi:10.1090/S0002-9947-1969-0280205-0.

2 Aranud Carayol, Christof Löding, Damian Niwiński, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Central European Journal of Mathematics*, 8(4):662–682, 2010.

3 Alonzo Church. Logic, arithmetic and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.

4 Thomas Colcombet and Christof Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3(2):1–36, 2007. doi:10.2168/LMCS-3(2:4)2007.

5 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2008.

6 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *International Colloquium on Automata, Languages and Programming, ICALP 2016*, 2016. to appear, full version on http://arxiv.org/abs/1602.08565.

7 Hu. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007. Release October, 12th 2007. URL: http://www.grappa.univ-lille3.fr/tata.

8 C. C. Elgot and J. E. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Dev.*, 9(1):47–68, 1965.

9 Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

10 Yuri Gurevich and Saharon Shelah. Rabin's uniformization problem. *J. Symb. Log.*, 48(4):1105–1119, 1983.

11 Michael Holtmann, Łukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. In *Foundations of Software Science and Computational Structures*, volume 6014 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2010. doi:10.1007/978-3-642-12032-9_18.

12 Frederick A. Hosch and Lawrence H. Landweber. Finite delay solutions for sequential conditions. In *ICALP*, pages 45–60, 1972.

13 Kojiro Kobayashi. Classification of formal languages by functional binary transductions. *Information and Control*, 15(1):95–109, 1969.

14 Dietrich Kuske and Thomas Weidner. Size and computation of injective tree automatic presentations. In *Mathematical Foundations of Computer Science 2011 – 36th International Symposium, MFCS 2011, Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 2011.

15 C. Löding. Logic and automata over infinite trees, 2009. Habilitation Thesis, RWTH Aachen, Germany.

16 Christof Löding and Sarah Winter. Synthesis of deterministic top-down tree transducers from automatic tree relations. In *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014.*, volume 161 of *EPTCS*, pages 88–101, 2014. `doi:10.4204/EPTCS.161`.

17 T. Milo, D. Suciu, and V. Vianu. Typechecking for xml transformers. *J. Comput. Syst. Sci.*, 66(1):66–97, 2003. `doi:10.1016/S0022-0000(02)00030-2`.

18 Yiannis Nichola Moschovakis. *Descriptive Set Theory*, volume 100 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, Amsterdam, New York, Oxford, 1980.

19 Dirk Siefkes. The recursive sets in certain monadic second order fragments of arithmetic. *Arch. für mat. Logik und Grundlagenforschung*, 17:71–80, 1975.

20 Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 133–192. Elsevier Science Publishers, Amsterdam, 1990.

Two-Variable Logic over Countable Linear Orderings

Amaldev Manuel[1] **and A. V. Sreejith**[2]

1 **Chennai Mathematical Institute (CMI), Chennai, India**
amal@cmi.ac.in

2 **Chennai Mathematical Institute (CMI), Chennai, India**
sreejithav@cmi.ac.in

Abstract

We study the class of languages of finitely-labelled countable linear orderings definable in two-variable first-order logic. We give a number of characterisations, in particular an algebraic one in terms of circle monoids, using equations. This generalises the corresponding characterisation, namely variety DA, over finite words to the countable case. A corollary is that the membership in this class is decidable: for instance given an MSO formula it is possible to check if there is an equivalent two-variable logic formula over countable linear orderings. In addition, we prove that the satisfiability problems for two-variable logic over arbitrary, countable, and scattered linear orderings are NEXPTIME-complete.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases $\circ$-monoids, countable linear orderings, FO^2

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.66

1 Introduction

Countable linear orderings are linear orderings over countable domains. They are of primary interest in the context of satisfiability of logics due to a result of Shelah [24]: the satisfiability problem of monadic second-order (MSO) logic is undecidable over arbitrary linear orderings, and in particular over the Reals. But by Rabin's theorem [18] the problem remains decidable when considered over countable linear orderings. Thus the class of countable linear orderings sets a natural limit to the decidability of satisfiability problem for MSO over linear orderings. This is in sharp contrast with first-order (FO) logic, that has the corresponding question decidable over arbitrary linear orderings. A second and perhaps more important reason why the class of countable linear orderings are interesting is the logic-algebra connection on its subclasses – MSO definable languages over finite words (*resp.* ω-words) are precisely the class of languages definable by finite monoids (*resp.* ω-semigroups, equivalently Wilke algebras) – extends to countable linear orderings: the result due to Carton-Colcombet-Puppis [4] states that MSO definable languages of countable linear orderings are precisely the class of languages of countable linear orderings recognisable by $\circ$-monoids (recalled in the next section).

The principal import of such a connection is well displayed by the seminal theorem of Schützenberger [21]: over finite words, FO definable languages are precisely the languages recognisable by aperiodic finite monoids, in particular the syntactic monoids of FO definable languages are aperiodic. This immediately yields the decidability of membership in the class of FO definable languages: compute the syntactic monoid of the given language and check if it is aperiodic. Since the time of Schützenberger numerous logics have been characterised algebraically, over finite words, ω-words etc.

However, unlike finite words or ω-words, characterising a logic over countable linear orderings has the following added advantage: An algebraic, in particular decidable, characterisation of a class of languages of countable linear orderings (for instance languages definable by FO) in terms of $\circ$-monoids, immediately provides decidable characterisations over restricted classes of countable linear orderings that are equationally definable (for instance finite words, ω-words, bi-infinite words, rationals etc.). In that sense, characterising a logic algebraically over the class of countable linear orderings in *one shot* characterises it over all equationally definable subclasses.

An elaborate study over a variety of sublogics over countable linear orderings was done in [6] where FO, FO[cut], WMSO, WMSO[cut], MSO[ordinals], MSO[scattered] etc. were characterised algebraically. These characterisations show that WMSO with "cut" quantifiers are equivalent to those with "ordinal" quantifiers, whereas the rest of the logics are expressively different from each other. The study also gives decidability of membership for all these logics.

As a continuation, in this work we consider the class of languages of countable linear orderings that are definable in two-variable first-order logic (FO^2). Two-variable FO is the fragment of FO with at most two variables x, y. While over abritrary structures FO has an undecidable satisfiable problem, FO^2 has a decidable, low complexity satisfiability problem. Yet FO^2 is expressive enough to contain modal logics. This feature of FO^2 has been thoroughly studied and the decidability of satisfiability has been extended to special classes of structures as well as particular vocabularies. FO^2 has been of significant interest over words (and ω-words) as well. Over finite words, FO^2 definable languages have numerous characterisations [26, 25]: they are precisely the class of languages (1) definable in unary LTL [26, 8], (2) recognisable by 2-way partially ordered DFA [22], (3) definable by turtle expressions [27], and (4) whose syntactic monoids are in the variety DA [26] (a finite monoid is in DA if it is aperiodic and all its regular D-classes are subsemigroups) etc. The last characterisation also gives a decision procedure for membership in the class. Not only that FO^2 languages have numerous characterisations, they also have a rich structure inside them [17]– they form an infinite hierarchy under quantifier alternations that is also decidable as shown recently [12].

Though FO^2 is well understood algebraically over finite words, its algebraic characterisation over countable orderings, in particular over infinite ones, is not immediate. This is because even with two variables one can express a variety of "infinitary" conditions: clearly with two variables we can express that letter a has a minimum occurrence (for instance by the formula $\varphi_1 = \exists y \forall x\, (a(x) \wedge a(y) \wedge x \geq y))$, as well as its negation, that is there is an infinite descending chain of a's. Consider the following formula φ_2 that says that if an a-position has an a-position before it, then it has two a-positions before it.

$$\varphi_2 = \forall x\, (a(x) \wedge \exists y\ (a(y) \wedge x > y) \rightarrow \exists y\, (a(y) \wedge x > y \wedge \exists x\, (a(x) \wedge y > x)))$$

The word aa, as well as a^{ω^*} (the ordering $(\mathbb{Z}^-, <)$ labelled with a) does not satisfy $\varphi_1 \wedge \varphi_2$ while the words a and aa^{ω^*} satisfy $\varphi_1 \wedge \varphi_2$. Thus, as $\varphi_1 \wedge \varphi_2$ exemplifies, with two variables one can stipulate both a minimum occurrence as well as existence of a descending chain of a letter. Therefore for the algebraic characterisation of FO^2 one has to make an intricate analysis of whether the letters appear as a minimum or as an infinite chain at different factors of the word.

In the rest of the section, we mention works that are related to the present paper and our contributions.

Related Work

Algebraic characterisations, in particular for FO, for scattered linear orderings are given in [1, 2, 5]. The connection between MSO over countable linear orderings and $\circ$-monoids was proved in [4]. It showed that MSO is equivalent to $\circ$-monoids. This gives an alternate proof of decidability of MSO over countable linear orderings. Moreover it showed that MSO collapses to the second level of the quantifier alternation hierarchy. An algebraic classification of MSO under various forms of set quantifications, in particular corresponding to the sublogics FO, FO[cut], WMSO, WMSO[cut], MSO[ordinals], MSO[scattered], was done in [6].

The literature on FO^2 over arbitrary structures is extensive and we don't mention it here. FO^2 over finite words as well as ω-words has been studied extensively [22, 26, 8, 25, 27, 12, 17]. A survey of various characterisations of FO^2 is given in [25]. The quantifier alternation hierarchy on FO^2 was proved in [11] and the decidability of the hierarchy was shown in [12].

Satisfiability of FO^2 over arbitrary structures were shown to be NEXPTIME-complete in [10]. The corresponding results (also NEXPTIME-complete) was shown for ω-words in [8], and for ordinals in [15]. More recently the satisfiability problem was studied for words with additional linear orderings/preorderings [3, 23, 14, 13].

Satisfiability of LTL over countable linear orderings is PSPACE-complete [7, 19].

Contributions

We study the two variable fragment of first order logic over countable linear orderings and give a number of different characterisations. The simplest characterisation is in terms of temporal logic (TL): FO^2 is equivalent to TL with only the modalities *Future* ($\mathtt{F}$) and *Past* ($\mathtt{P}$). Our major contribution is an algebraic characterisation for FO^2. We show that it corresponds to a subclass of $\circ$-monoids and give two algebraic characterisations for this subclass: (1) by equations, and (2) as the class of $\circ$-monoids that are aperiodic and whose regular $\mathcal{J}$ classes are sub $\circ$-monoids. It follows that the membership in the class is decidable.

Next we study the satisfiability problem for FO^2 over countable linear orderings. The models of FO^2 formulas could be infinite, but we show that a satisfiable formula always admits a scattered model that has a finite representation of small (exponential in the size of the formula) size. Thus we prove that the satisfiability of FO^2 over countable linear orderings is NEXPTIME-complete. From this we also deduce that the satisfiability problems for FO^2 over arbitrary and scattered orderings are NEXPTIME-complete.

Structure of the paper

In Section 2, we introduce words over countable linear orderings, two-variable first-order logic, and the algebra required to characterise FO^2, namely $\circ$-monoids. In Section 3 we prove our main result (Theorem 8) which characterises FO^2. Section 4 deals with the satisfiability of FO^2 over countable linear orderings. Finally we conclude our results in Section 5.

2 Preliminaries

In this section we recall the basic facts about (countable) linear orderings, $\circ$-monoids, logics and related notions.

Words over countable linear orderings. A *linear ordering* $\alpha = (Z, <)$ is a set Z equipped with a total order $<$. For $X, Y \subseteq Z$ we write $X < Y$ if $x < y$ for each x in X and y in Y. In particular $\emptyset < X < \emptyset$ for any set X. Also if $X < Y$, $Y < Z$ and Y is nonempty, then

$X < Z$. A *cut* of the linear ordering α is a pair (Z_1, Z_2) such that $Z = Z_1 \cup Z_2$ and $Z_1 < Z_2$. The set of all cuts are linearly ordered and has the least upper bound property [2]. A set L is a *prefix* of X if $X = L \cup K$ and $L < K$ for some $K \subseteq X$. Similarly if $X = L \cup K$ and $L < K$, then K is a *suffix* of X. Element $z \in Z$ is an *upperbound* (*resp. lowerbound*) of a set $X \subseteq Z$ if $x \leq z$ (*resp.* $z \leq x$) for each x in X. A set X is *right-open* (*resp. left-open*) if it has no maximum element (*resp.* minimum element). Nonempty suffixes of right-open sets are right-open and nonempty prefixes of left-open sets are left-open. The set X is *dense* if between any two elements in the set there is another element; set X is *scattered* if it has no dense subsets. An ordering is a *countable* (scattered) linear ordering if the set Z is countable (scattered). See [20] for further details.

For a finite alphabet A and a linear ordering $\alpha = (Z, <)$, we define a *word* $w : \alpha \to A$ to be a mapping from the set Z to A. We call α the *domain* of w, $dom(w)$. For a word w, we say a point/position x to denote an element $x \in dom(w)$. The notation $w[x]$ denote the letter at the x^{th} position in w. A word has a minimal (respectively maximal) element if its domain has a minimal (maximal) element. The word u is a suffix (prefix) of w if $dom(u)$ is a suffix (prefix) of $dom(w)$. If u and v are words, then uv denotes the unique word w such that $(dom(u), dom(v))$ is a cut of $dom(w)$. This operation is naturally extended to a set of words $\{w_i\}_\alpha$ indexed by a linear ordering α as $\prod_{i\in\alpha} w_i$ (see [6] for more details). For a set $S \subseteq A$, and a word w, we denote the restriction of w to the positions labelled by S as $w_{|S}$. That is $w_{|S} = \{i \in dom(w) \mid w[i] \in S\}$.

The following words are of special interest. ϵ stands for the empty word (the word over an empty domain). The word $\{a\}^\omega$ (denoted in short as a^ω) denotes the word over the domain $(\mathbb{N}, <)$ such that every position is mapped to the letter a. Similarly a^{ω^*} denotes the word over the domain $(\mathbb{N}^-, <)$ where every position is mapped to letter a. A *perfect shuffle* over a nonempty set $S \subseteq A$ of letters, denoted by S^η, is the word over domain $(\mathbb{Q}, <)$ such that any nonempty open interval contains each of the letters in S. This is a unique word (up to isomorphism) (see [4]) and is an example of a dense word, i.e. a word whose domain is dense.

For an alphabet A, the set of all words over nonempty countable domains is denoted by A°. For a word w, we define *alphabet*(w) to be the set of all letters in w. A *language* over the alphabet A is a subset of A°. The language $\{a\}^\infty \subseteq \{a\}^\circ$ (or written as a^∞) denotes all words which are right open. Similarly for a set $S \subseteq A$, the language S^∞ is the set of all words whose letters come only from S and any letter from S can be seen arbitrarily towards the right. The sets $a^{-\infty}$ and $S^{-\infty}$ are defined analogously.

Circle monoids and algebras. A $\circ$-*semigroup* $\mathbf{M} = (M, \pi)$ consists of a set M with an operation $\pi : M^\circ \to M$ which satisfies the following two properties (1) $\pi(a) = a$ for all $a \in M$, (2) *generalised associativity property* – that is $\pi\big(\prod_{i\in\alpha} u_i\big) = \pi\big(\prod_{i\in\alpha} \pi(u_i)\big)$ for every countable linear ordering α. If $\mathbf{M}$ has an identity element, then it is called a $\circ$-*monoid*. An element $e \in \mathbf{M}$ is an *idempotent* if $\pi(ee) = e$.

For the rest of the paper, we assume that the monoid $\mathbf{M}$ is finite, that is M is a finite set. The product π is over countable linear orderings and hence it is not possible to finitely represent π. Fortunately, we are able to represent this by a $\circ$-algebra that uses only finite sets and finitely many operations. The following operations are derivable from a $\circ$-monoid $\mathbf{M} = (M, \pi)$:

- *Finite product*, $\cdot : M^2 \to M$ such that $\cdot(a, b) = \pi(ab)$
- *Omega*, $\omega : M \to M$ such that $\omega(a) = \pi(a^\omega)$
- *Omega**, $\omega^* : M \to M$ such that $\omega^*(a) = \pi(a^{\omega^*})$
- *Shuffle*, $\eta : \mathcal{P}(M) \to M$ such that $\{a_1, \ldots, a_k\}^\eta = \pi(\{a_1, \ldots, a_k\}^\eta)$

The resulting structure $(M, \cdot, \omega, \omega^*, \eta)$ is called a $\circ$-*algebra* if it satisfies some additional axioms relating the operations (for example $a \cdot a^\omega = a^\omega$, $(a^\eta)^\omega = a^\eta$ etc.). We skip these details and refer the reader to the paper by Carton et. al [4] for a detailed discussion. The relavant fact is that, for any $\circ$-monoid there exists a unique $\circ$-algebra and vice versa [4].

An important "tool" to understand finite monoids (in our case $\circ$-monoids) is *Green's relations*. In a $\circ$-monoid $\mathbf{M}$, we say that two elements $u \geq_{\mathcal{J}} v$ if there exists two elements $x, y \in \mathbf{M}$ such that $v = xuy$ and $u\mathcal{J}v$ (called J equivalent) if it is both $u \geq_{\mathcal{J}} v$ and $v \geq_{\mathcal{J}} u$. We also say that two elements are $u \geq_{\mathcal{R}} v$ (similarly $u \geq_{\mathcal{L}} v$) if there exists an element $x \in \mathbf{M}$ such that $v = ux$ $(v = xu)$. Also $u\mathcal{R}v$ if $u \geq_{\mathcal{R}} v$ and $v \geq_{\mathcal{R}} u$. Similarly we can define $u\mathcal{L}v$. The relations $\mathcal{L}$ and $\mathcal{R}$ are right and left congruences respectively. If a $\mathcal{J}$ class contains an idempotent then it is called a *regular* $\mathcal{J}$ class. All elements in a $\mathcal{J}$ class can be described by an "eggbox" structure, such that $u\mathcal{J}v$ iff there exists elements $x, y \in \mathbf{M}$ such that $u\mathcal{R}x\mathcal{L}y\mathcal{R}v$. For a more detailed elaboration on this subject see [16].

The class of $\circ$-monoids that satisfies the property – there exists an $n \in \mathbb{N}$ such that $a^n = a^{n+1}$ for all $a \in \mathbf{M}$ – are called *aperiodic*. It is precisely the class of $\circ$-monoids which do not contain any non-trivial group as a subsemigroup of $(M, \cdot)$ (by Schützenberger's theorem [21]).

One way to denote a class of $\circ$-monoids is by equations. For instance, we say that $\mathbf{M}$ satisfies the equation $x^* = x^\omega x^{\omega^*}$, if for all elements $a \in \mathbf{M}$, $a^* = a^\omega a^{\omega^*}$, where a^* is the unique idempotent power of a.

We say that a language $L \subseteq A^\circ$ is recognised by the $\circ$-monoid $\mathbf{M}$, if there is a morphism, $\gamma : A^\circ \to \mathbf{M}$ and a subset $S \subseteq \mathbf{M}$ such that $L = \gamma^{-1}(S)$. The *syntactic* $\circ$-*monoid* of a language L is the minimal $\circ$-monoid $\mathbf{M}$ recognising L that has the following universal property: any $\circ$-monoid recognising L has a morphism onto $\mathbf{M}$.

Logics. Monadic second-order logic (MSO) over a finite alphabet A is a logic which can be inductively built using the following operations.

$$a(x) \mid x < y \mid x = y \mid \alpha_1 \vee \alpha_2 \mid \neg\alpha \mid x \in X \mid \exists x\ \alpha \mid \exists X\ \alpha$$

Here $a \in A$. If we remove the second-order quantification, we get first-order logic (FO). If we further restrict the logic to use only two variables (but allowing repetitions) we get FO^2. Note that, we do not have the *successor* relation in our logic.

A formula with no free variables is called a sentence. The language of a sentence φ (denoted by $L(\varphi)$) is the set of all $u \in A^\circ$ that satisfies φ.

Over finite words, FO^2 can talk about occurrence of letters and also about the order in which they appear [8, 27]. Over countable linear orders, FO^2 can also talk about an infinite sequence of a letter. For example, the language a^∞ is definable in FO^2 by stating that, every position is labelled by a and there is no maximum position.

$$\big(\forall x\ \exists y > x\big) \wedge \big(\forall x\ a(x)\big)$$

Also, for a subset $S \subseteq A$, we can also express the language S^∞ in FO^2.

$$\big(\forall x \bigwedge_{a \in S} \exists y > x\ a(y)\big) \wedge \big(\forall x \bigvee_{a \in S} a(x)\big)$$

Analogously, FO^2 can also talk about left open words.

The temporal logic $\{\mathtt{F},\mathtt{P}\}$-TL over the alphabet A is the logic with the set of formulas – a when a is a letter in A, and $\mathtt{F}\varphi$ and $\mathtt{P}\varphi$ when φ is a formula – that is closed under Boolean operations. To state the semantics fix a word $u \in A^\circ$. A position $i \in dom(u)$ satisfies – the formula a if i is labelled with the letter a, and the formula $\mathtt{F}\varphi$ (*resp.* $\mathtt{P}\varphi$) if there is a position $i < j \in dom(u)$ (*resp.* $i > j \in dom(u)$) that satisfies the formula φ. The semantics for Boolean connectives are defined in the usual way. The word u satisfies the formula φ if there is a position $i \in dom(u)$ that satisfies the formula (see [8] for a detailed presentation). The language of the formula φ is the set of all $u \in A^\circ$ that satisfies φ.

3 Characterisation

In this section, we give the algebraic characterisation for FO^2($<$) over countable linear orderings. As we noted earlier, $\circ$-monoid captures MSO. Here we identify a subclass which will capture the two-variable first-order fragment. Our characterisation builds on the characterisation for FO^2 on finite words given in [26]. In particular, we crucially use a generalisation of the congruence given there.

▶ **Definition 1.** We define $\circ$-DA to be the subclass of $\circ$-monoids that satisfy the following equations.

1. $(xyz)^* y (xyz)^* = (xyz)^*$
2. $x^* = (x)^\omega (x)^{\omega^*}$
3. $\{x_1, \ldots, x_k\}^\eta = (x_1 \cdots x_k)^{\omega^*} (x_1 \cdots x_k)^\omega$

The first equation corresponds to the variety DA of finite monoids [25]. It identifies the constraints the product operation has to satisfy. The second equation corresponds to FO definable languages of countable linear orderings [6]. This equation states that a $\mathcal{J}$ class with an idempotent will also contain its omega and omega* powers. The last equation says that, $\circ$-DA cannot differentiate between dense and scattered orderings.

The connection between logic and algebra is established using the following congruence.

A congruence on words

Let $u \in A^\circ$ be an arbitrary word. *alphabet*(u) is defined as the set of all letters occurring in u. For a letter a in *alphabet*(u), let $P_u(a)$ denote the set of all positions in u labelled with a. Let $T_r^1(u) \subseteq alphabet(u)$ be the set of all letters a such that $P_u(a)$ has a maximal element. Furthermore, let $T_r^\omega(u)$ be the set $alphabet(u) \setminus T_r^1(u)$, i.e. the set of all letters that do not have a maximal occurrence. Similarly let $T_l^1(u) \subseteq alphabet(u)$ be the set of all letters a such that $P(a)$ has a minimal element, and let $T_l^{\omega^*}(u)$ be the set $alphabet(u) \setminus T_l^1(u)$.

▶ **Definition 2.** The relation $\lesssim_r$ over the set of letters $T_r^\omega(u)$ is defined as follows:

$a \lesssim_r b$ if each a-position i in u has a b-position j to its right (i.e. $j > i$).

▶ **Lemma 3.** *The relation $\lesssim_r$ is a total preorder on the set $T_r^\omega(u)$.*

We write $\sim_r$ to denote the equivalence relation associated with the preorder $\lesssim_r$. For a letter a in $T_r^\omega(u)$ we let $[a]_r \subseteq T_r^\omega(u)$ denote the equivalence class of a with respect to the total preorder $\lesssim_r$, i.e. $[a]_r = \{b \in T_r^\omega(u) : b \sim_r a\}$. Also, we extend the definition of P_u to equivalence classes by defining $P_u([a]_r) = \bigcup_{a \in [a]_r} P_u(a)$. We write $<_r$ to denote the total order on $\{[a]_r : a \in T_r^\omega(u)\}$.

By symmetry, the dual relation $\lesssim_l$ defined as,

$b \lesssim_l a$ if each a-position in u has a b-position to its left, is also a total preorder. The corresponding equivalence relation and strict order relation are denoted as $\sim_l$ and $<_l$. Given a circle word u the preorders $\lesssim_r$ and $\lesssim_l$ associated with u are called the *right preorder* and *left preorder* of u respectively. As before we define $P_u([a]_l) = \bigcup_{a \in [a]_l} P_u(a)$.

▶ **Example 4.** Let $S = \{a, b\}$ and let $u \in S^{-\infty}$ be an arbitrary word. Consider the word $v = ua^{\omega^*}a^\omega ab^\omega \in \{a, b\}^\circ$. Then $T_l^1(u) = T_l^1(v) = \emptyset$ and $T_l^{\omega^*}(u) = T_l^{\omega^*}(v) = \{a, b\}$, since a and b occur infinitely often towards left in both u and v. It also follows that $a \lesssim_l b$ and $b \lesssim_l a$. Since u is an arbitrary word, we do not know about $T_r^1(u)$ and $T_r^\omega(u)$. But, since a has a maximum point in v, we have $T_r^1(v) = \{a\}$ and $T_r^\omega(v) = \{b\}$. Moreover $b \lesssim_r b$.

Consider another word $w = a^\omega b^\omega$. Here we have $T_l^1(w) = \{a, b\}$ and $T_l^{\omega^*}(w) = T_r^1(w) = \emptyset$. We also have $T_r^\omega(w) = \{a, b\}$ and $a \lesssim_r b$ but $b \not\lesssim_r a$.

We will now introduce left/right decomposition of words. The idea is to factorise a word in a particular way to capture the "pivot" points for an FO^2 formula.

▶ **Definition 5.** Let $a \in \mathit{alphabet}(u)$. If $a \in T_l^1(u)$, then there exists a unique factorisation of u as (u_0, a, u_1) such that $u = u_0 a u_1$ and $a \notin \mathit{alphabet}(u_0)$. This is called the a-left decomposition of u. Similarly there is a unique factorisation of u as (u_0, a, u_1) such that $a \notin \mathit{alphabet}(u_1)$, if $a \in T_r^1(u)$. This is called the a-right decomposition of u.

We are also interested in left decomposition obtained by a set of positions $P_u([a]_l)$, where $[a]_l \in T_l^{\omega^*}(u)/\sim_l$. That is for a subset of positions $P_u([a]_l)$ of u, we define the $P_u([a]_l)$*-left decomposition of a word* u to be the unique maximal cut (u_0, u_1) such that $P_u([a]_l) \cap \mathit{dom}(u_0) = \emptyset$. Note that if $S = \{b \mid b \sim_l a\}$, then there is a prefix of u_1 such that $u_1 \in S^{-\infty}$. This follows from the fact that, the decomposition (u_0, u_1) is a maximal cut. Similarly the $P_u([a]_r)$*-right decomposition of a word* u is defined to be the unique minimal cut (u_0, u_1) such that $P_u([a]_r) \cap \mathit{dom}(u_1) = \emptyset$.

With the left/right decomposition defined, we can define the *congruence on words*, $\equiv_n$ which essentially captures a sequence of unique decompositions.

▶ **Definition 6.** For an alphabet A, a natural number $n \in \mathbb{N}$ and words $u, v \in A^\circ$, we define $u \equiv_n v$ by induction on $m = n + |A|$ as follows.

1. If $n = 0$ (the base case): $u \equiv_0 v$ for all $u, v \in A^\circ$.
2. If $n > 0$: We say $u \equiv_n v$ if the following conditions are satisfied:
   a. $\mathit{alphabet}(u) = \mathit{alphabet}(v)$, $T_r^1(u) = T_r^1(v)$, and $T_l^1(u) = T_l^1(v)$. (This condition implies that $T_r^\omega(u) = T_r^\omega(v)$ and $T_l^{\omega^*}(u) = T_l^{\omega^*}(v)$).
   b. The right preorders of u and v (both on the same set by the previous observation) are the same. Similarly the left preorders of u and v are the same. (We denote the left and right preorders as $\lesssim_l, \lesssim_r$ respectively).
   c. For each $a \in T_l^1(u) = T_l^1(v)$, let (u_0, a, u_1) be the a-left decomposition of u, and let (v_0, a, v_1) be the a-left decomposition of v, then $u_0 \equiv_n v_0$ and $u_1 \equiv_{n-1} v_1$. Note that the induction parameter has reduced in both cases: u_0 has at least one letter less than u; and we have a lesser congruence in u_1.
   d. Similarly, for each $a \in T_r^1(u) = T_r^1(v)$, let (u_0, a, u_1) be the a-right decomposition of u and let (v_0, a, v_1) be the a-right decomposition of v, then $u_0 \equiv_{n-1} v_0$ and $u_1 \equiv_n v_1$.
   e. For each class $[a]_l \in T_l^{\omega^*}(u)/\sim_l = T_l^{\omega^*}(v)/\sim_l$, let (u_0, u_1) be the $P_u([a]_l)$-left decomposition of u and let (v_0, v_1) be the $P_v([a]_l)$-left decomposition of v, then $u_0 \equiv_n v_0$ and $u_1 \equiv_{n-1} v_1$. Again, the induction parameter has reduced in both cases: u_0 has at least one letter less than u; and we have a lesser congruence in u_1.

f. Similarly for each class $[a]_r \in T^\omega_r(u)/\sim_r = T^\omega_r(v)/\sim_r$, let (u_0, u_1) be the $P_u([a]_r)$-right decomposition of u and let (v_0, v_1) be the $P_v([a]_r)$-right decomposition of v, then $u_0 \equiv_{n-1} v_0$ and $u_1 \equiv_n v_1$.

▶ **Lemma 7.** *The relation $\equiv_n$ is a congruence relation for every $n \in \mathbb{N}$.*

Main theorem

We are now in a position to state our main theorem.

▶ **Theorem 8.** *Let $L \subseteq A^\circ$. Then the following are equivalent:*
1. *L is definable in $\{\mathtt{F}, \mathtt{P}\}$-TL.*
2. *L is $\mathrm{FO}^2(A, \leq)$ definable.*
3. *L is a union of $\equiv_n$ congruent classes for some $n \in \mathbb{N}$.*
4. *L is recognised by a $\circ$-DA.*
5. *L is recognised by an aperiodic $\circ$-monoid where all regular $\mathcal{J}$ classes are sub $\circ$-monoids.*
6. *The syntactic $\circ$-monoid of L is in $\circ$-DA.*

The proof of $(1 \Leftrightarrow 2)$ follows easily (see [8, 7]).
In subsection 3.1 we show the equivalence of the different monoid views $(4 \Leftrightarrow 5 \Leftrightarrow 6)$.
In subsection 3.2 we show $(4 \Rightarrow 3)$.
To prove $(2 \Rightarrow 4)$, we use 2-pebble *Ehrenfeucht-Fraïssé* (EF) games [26]. The EF game gives a game congruence $\cong_n$ defined as: $u \cong_n v$ if the duplicator wins the n-round 2-pebble game on the pair of words (u, v). See [26] for the game congruence and its equivalence to FO^2. Thus it suffices to show that the game congruence satisfies the equations of $\circ$-DA.
To show direction $(3 \Rightarrow 2)$ we follow the proof in [26]. It suffices to show that if $L \subseteq A^\circ$ is a union of $\equiv_n$ congruent classes for some n, then it is definable in $\mathrm{FO}^2(<)$. More precisely we prove the following lemma (again using the equivalence of game congruence $\cong_n$ and FO^2).

▶ **Lemma 9.** *For words $u, v \in A^\circ$, If $u \not\equiv_n v$, then $u \not\cong_{n+alphabet(u)} v$ i.e. the spoiler has a winning strategy in the 2-pebble $n + alphabet(u)$-round EF game on u and v.*

Since the syntactic $\circ$-monoid (and its finite representation using $\circ$-algebra) is computable given an MSO formula [4], it follows that it is decidable to check whether the language is FO^2 definable.

▶ **Corollary 10.** *For a sentence ϕ in MSO$[<]$, it is decidable whether $L(\phi)$ is FO$^2[<]$ definable.*

In the next subsection we show the equivalence of the different monoid views. The subsection after that shows that if a language is accepted by a $\circ$-monoid, then it is a union of congruence classes $\equiv_n$ for some $n \in \mathbb{N}$.

3.1 The different Monoid views

In this subsection we show that the different views of $\circ$-DA are equivalent. That is, $(4 \Leftrightarrow 5 \Leftrightarrow 6)$ of Theorem 8. The direction $(4 \Rightarrow 5)$, follows from standard ideas in semigroup theory and the reverse direction $(5 \Rightarrow 4)$, follows from the below lemma:

▶ **Lemma 11.** *Let $\mathbf{M}$ be an aperiodic $\circ$-monoid such that all regular $\mathcal{J}$ classes of $\mathbf{M}$ are sub $\circ$-monoids. Let $\gamma : A^\circ \to \mathbf{M}$ be a morphism and $u \in A^\circ$, such that $\gamma(u) = e$ an idempotent. Then, for all words $v \in \{alphabet(u)\}^\circ$, we have $\gamma(uvu) = \gamma(u)$.*

To prove direction $(4 \Rightarrow 6)$, assume L is recognised by a monoid in $\circ$-DA. Since, $\circ$-DA is closed under quotienting, it follows that the syntactic monoid of L satisfies the equations of $\circ$-DA (see [6] for more details about syntactic congruence and monoids).

3.2 Algebra to Congruence

In this subsection we show direction $(4 \Rightarrow 3)$ of Theorem 8. The proof improves on the equivalence of the congruence and algebra given in [26]. We show that a language recognisable by a $\circ$-monoid in $\circ$-DA, satisfies the congruence relation $\equiv_n$ for some $n \in \mathbb{N}$. Let L be recognised by the morphism $\gamma : A^\circ \to \mathbf{M}$, where $\mathbf{M}$ is in $\circ$-DA. It suffices to show that there exists an $n \in \mathbb{N}$ such that $\equiv_n$ is a finer congruence than the monoid congruence. That is for $u, v \in A^\circ$, if $u \equiv_n v$, then $\gamma(u) = \gamma(v)$. Since $\mathbf{M}$ is an aperiodic monoid (follows from equations of $\circ$-DA) it is sufficient to show that $u\mathcal{R}v$ and $u\mathcal{L}v$.

The left/right decomposition of words are closely related to how the $\mathcal{R}$ classes fall in the word. The following definition identifies a sequence of $\mathcal{R}$-smooth factors (those factors where there is no $\mathcal{R}$ fall), and the subsequent lemma shows there exists such a unique sequence.

▶ **Definition 12.** Let $\gamma : A^\circ \to \mathbf{M}$. Let $w \in A^\circ$. Then the $\mathcal{R}$ decomposition of w is defined as the sequence $(w_0, a_1, w_1, a_2, \ldots, a_k, w_k)$ such that

1. $a_i \in A \cup \{\epsilon\}$ and $w_i \in A^*$, for all $i \leq k$.
2. $w = w_0 a_1 \ldots a_k w_k$.
3. For each $0 < i \leq k$, if a_i is empty, then the following conditions hold:
   - **a.** w_i does not have a left end point.
   - **b.** $(w_0 a_1 \ldots a_i w_i')$ $\mathcal{R}$ $\gamma(w_0 a_1 \ldots a_i w_i)$, for all nonempty prefix w_i' of w_i.
   - **c.** $\gamma(w_0 a_1 \ldots w_{i-1})$ $\not\mathcal{R}$ $\gamma(w_0 a_1 \ldots w_{i-1} a_i w_i)$.
4. For each $0 < i \leq k$, if a_i is not empty, then the following holds:
   - **a.** $\gamma(w_0 a_1 \ldots a_i)$ $\mathcal{R}$ $\gamma(w_0 a_1 \ldots a_i w_i)$.
   - **b.** $\gamma(w_0 a_1 \ldots w_{i-1})$ $\not\mathcal{R}$ $\gamma(w_0 a_1 \ldots w_{i-1} a_i)$.

▶ **Lemma 13.** *Let $w \in A^\circ$ be an arbitrary word. Then, there is a unique $\mathcal{R}$ decomposition $(w_0, a_1, \ldots, a_k, w_k)$ of w.*

The following Lemma connects $\mathcal{R}$ decompositions and left decompositions.

▶ **Lemma 14.** *Let $(w_0, a_1, \ldots, a_k, w_k)$ be the $\mathcal{R}$ decomposition of w. Then, for each $0 < i \leq k$,*

1. *If a_i is not empty, then $a_i \notin alphabet(w_{i-1})$.*
2. *If a_i is empty, then there exists an $a \notin alphabet(w_{i-1})$ such that $a \in T_l^{\omega^*}(w_i')$ for all nonempty prefix w_i' of w_i.*

We are now in a position to prove our claim.

Proof of Theorem 8, $(4 \Rightarrow 3)$**.** We show that if $u \equiv_m v$ for a sufficiently large m (depending only on $alphabet(u)$ and $\mathbf{M}$), then $\gamma(u)\mathcal{R}\gamma(v)$. The $\mathcal{L}$ equivalence can be shown symmetrically. As discussed in the beginning, this proves our claim. Our induction hypothesis is as follows:

$$\text{If } u \equiv_m v \text{ for an } m > |alphabet(u)| \times |\mathbf{M}|, \text{ then } \gamma(u) = \gamma(v).$$

The base case, when $m = 0$ is clearly true, since $u = v = \epsilon$ (note that, in this case $alphabet(u) = \emptyset$). Let us now consider the inductive step, for $m > 0$, we have $u \equiv_m v$. Our aim is to show that $\gamma(u) = \gamma(v)$. Consider the $\mathcal{R}$ decomposition of $u = (u_0, a_1, u_1, \ldots, a_k, u_k)$. We give a sequence $v = (v_0, a_1, v_1, \ldots, a_k, v_k)$ such that $\gamma(u_i) = \gamma(v_i)$ for all $i < k$ and hence $\gamma(u) \geq_\mathcal{R} \gamma(v)$.

Define $u_i' = u_i a_{i+1} \ldots u_k$, for all $i \leq k$. We do the following procedure for i ranging from $1, 2, \ldots, k$. During every iteration of i, we give v_i', a suffix of v_i such that the invariant $u_i' \equiv_{m-i} v_i'$ is maintained. To start the iteration we set $v_0' = v$ and $u_0' \equiv_m v_0'$

1. If a_i is non empty, then (u_{i-1}, a_i, u'_i) is the a_i-left decomposition of the word u'_{i-1} (follows from Lemma 14). Since $(u'_{i-1} \equiv_{m-(i-1)} v'_{i-1})$, there exists an a_i-left decomposition of $v'_{i-1} = (v_{i-1}, a_i, v'_i)$ such that $u_{i-1} \equiv_{m-(i-1)} v_{i-1}$ and $u'_i \equiv_{m-i} v'_i$.
2. If a_i is empty, then (u_{i-1}, u'_i) is an $[a]_l$-left decomposition of the word u'_{i-1} for an $[a]_l \in T_l^{\omega^*}(u'_{i-1})/\sim_l$ (follows from Lemma 14). Since $(u'_{i-1} \equiv_{m-(i-1)} v'_{i-1})$, there exists an $[a]_l$-left decomposition of $v'_{i-1} = (v_{i-1}, v'_i)$ such that $u_{i-1} \equiv_{m-(i-1)} v_{i-1}$ and $u'_i \equiv_{m-i} v'_i$.

Assign $v_k = v'_k$ obtained at the end of iteration.

Note that $k \leq |\mathbf{M}|$. For an $i < k$, we have $|alphabet(u_i)| = |alphabet(v_i)| < |alphabet(u)|$ (from Lemma 14) and therefore $m-i > |alphabet(u_i)| \times |\mathbf{M}|$. Since $u_i \equiv_{m-i} v_i$ from induction hypothesis, it follows $\gamma(u_i) = \gamma(v_i)$, for all $i < k$. Therefore $\gamma(u_0 \dots a_k) = \gamma(v_0 \dots a_k)$.

It remains to show that $\gamma(u) \geq_{\mathcal{R}} \gamma(v)$. Depending on whether a_k is empty or not, we get the following cases.

1. If a_k is non empty, then $\gamma(u_0a_1 \dots a_ku_k) \ \mathcal{R} \ \gamma(u_0a_1 \dots a_k) = \gamma(v_0a_1 \dots a_k) \geq_{\mathcal{R}} \gamma(v)$. The first condition follows from the fact that the sequence $(u_0a_1 \dots u_k)$ is an $\mathcal{R}$ decomposition, and the second condition follows from the fact that $\gamma(u_i) = \gamma(v_i)$ for all $i < k$.
2. If a_k is empty, then $(u_0 \dots u_{k-1}, u_k)$ and $(v_0 \dots v_{k-1}, v_k)$ are both S-left decomposition for an $S \in T_l^{\omega^*}(u_i)/\sim_l$. Hence there are prefixes u'_k of u_k and v'_k of v_k such that $u'_k, v'_k \in S^{-\infty}$. From Lemma 11 we know that $\gamma(u'_k)\mathcal{R}\gamma(v'_k)$. Therefore, $\gamma(u_0a_1 \dots a_ku_k) \ \mathcal{R} \ \gamma(u_0a_1 \dots u'_k) \ \mathcal{R} \ \gamma(v_0a_1 \dots v'_k) \ \geq_{\mathcal{R}} \ \gamma(v_0a_1 \dots a_kv_k) = \gamma(v)$.

We now have $\gamma(u) \geq_{\mathcal{R}} \gamma(v)$. By a symmetric argument we get $\gamma(v) \geq_{\mathcal{R}} \gamma(u)$ and therefore $\gamma(u) \ \mathcal{R} \ \gamma(v)$. By $\mathcal{L}$-$\mathcal{R}$ symmetry, $\gamma(u) \ \mathcal{L} \ \gamma(v)$ and since $\mathbf{M}$ is aperiodic $\gamma(u) = \gamma(v)$. ◀

4 Satisfiability

In this section we address the satisfiability problem of two-variable logic over countable linear orderings. The rest of the section is devoted to the proof of the below theorem. Take note of the fact that in this section Σ denotes a set of unary predicates (and not an alphabet). Our models are words over the alphabet $\mathcal{P}(\Sigma)$.

▶ **Theorem 15.** *The following problems are* NEXPTIME*-complete: Satisfiability of* FO$^2(\Sigma, <)$ *over*
1. *arbitrary linear orderings,*
2. *countable linear orderings,*
3. *scattered linear orderings.*

First we deal with the hardness part of the theorem. By downward Löwenheim-Skolem theorem, every satisfiable first-order formula has a countable model, and therefore (1) reduces to (2). Similary by Lemma 16 (given below), if a two-variable logic formula has a countable model, then it has a scattered model. Therefore (2) reduces to (3). Secondly, satisfiability of FO$^2(\Sigma)$ over arbitrary structures already is NEXPTIME-hard [9], and therefore (1), (2) and (3) are NEXPTIME-hard.

Next we prove that (2) and (3) are in NEXPTIME. The idea is to show that for any satisfiable formula there is a model of a particular form that admit at most exponentially big (in the size of the formula) description.

Let φ be a FO$^2(\Sigma, <)$ formula. Using standard ideas we obtain a formula $\varphi' \in$ FO$^2(\Sigma', <)$ in Scott normal form, i.e.

$$\varphi' = \forall x \forall y \, \psi(x, y) \wedge \bigwedge_i \forall x \exists y \, \chi_i(x, y) \, , \tag{1}$$

where $\Sigma' \supseteq \Sigma$, $|\Sigma'| = |\Sigma| + \mathcal{O}(|\varphi|)$, $|\varphi'| = \mathcal{O}(|\varphi|)$, $\psi(x, y)$ and $\chi_i(x, y)$ are quantifier free, such that φ and φ' are equisatisfiable (one is satisfiable if and only if the other is satisfiable). More precisely, the sets of models of φ and φ' are isomorphic upto the erasure of the unary predicates $\Sigma' \setminus \Sigma$.

We introduce some notation. Given a set of unary predicates P, we define a unary type over P to be a maximal conjunction of literals (i.e. $U(x)$ or $\neg U(x)$ where U is a unary predicate in P) over the same variable that is satisfiable. When the set P is clear from the context we just use types to refer to the unary types over P. We write $\mathrm{tp}(P)$ to denote the types over the predicates P. Each position of a $\circ$-word satisfies exactly one type, called the *type of the position.* Models of φ' are $\circ$-words over the alphabet $\mathrm{tp}(\Sigma')$.

Next we prove that formulas φ' in Scott normal form possess particular kind of models.

▶ **Lemma 16.** *If φ' is satisfiable, then it has a model of the form $u_1^{\lambda_1} \cdots u_n^{\lambda_n}$ where $n \geq 1$ is a natural number, for each $1 \leq i \leq n$, u_i is a finite word over the alphabet $\mathrm{tp}(\Sigma')$ and λ_i is in $\{1, \omega, \omega^*\}$, such that*

1. *every type occurs at most once in each u_i, and*
2. *every type occurs in at most two u_i's.*

A model of the form $u = u_1^{\lambda_1} \cdots u_n^{\lambda_n}$ is finitely represented as a sequence of pairs $(u_1, \lambda_1) \cdots (u_n, \lambda_n)$. Lemma 16 guarantees that for every satisfiable formula φ' there is a representation of size at most $3 \cdot \mathrm{tp}(\Sigma) \leq 3 \cdot 2^{|\varphi'|}$.

▶ **Lemma 17.** *Given a sequence of pairs $(u_1, \lambda_1) \cdots (u_n, \lambda_n)$ and a formula φ' checking if the $\circ$-word $u_1^{\lambda_1} \cdots u_n^{\lambda_n}$ satisfies the formula φ in Scott normal form is in* PTIME.

To complete the proof of the Theorem 15 we describe a NEXPTIME algorithm for FO^2 formulas over countable linear orders: The algorithm converts the input formula to Scott normal form and guesses an atmost exponentially large representation of a model of the form described by Lemma 16 and checks that it is indeed a model by Lemma 17.

5 Conclusion

In this paper we characterised first-order logic with two variables over countable linear orderings. It is equivalent to a fragment of temporal logic and is characterised by a subclass of $\circ$-monoids, called $\circ$-DA. The class $\circ$-DA is the class of $\circ$-monoids whose regular $\mathcal{J}$ classes are sub $\circ$-monoids. We also proved an alternate characterisation of this class using equations and this yields decidability of membership in this class. Next we considered the satisfiability problem for FO^2 over arbitrary, countable and scattered linear orderings and showed that all the problems are NEXPTIME-complete.

Finally we note that FO^2 with order and *successor* relation (position $j > i$ is the successor of position i if there is no position between them) is strictly more powerful that FO^2 with only the order relation. To see this it is enough to note that a^ω and $a^\omega a^\omega$ are indistinguishable by any formula in the latter class, while there is a formula, namely "there is exactly one position without a predecessor" that separates them. We leave as future work the question of extending the characterisation in the present paper to handle the successor relation.

References

1 Nicolas Bedon, Alexis Bès, Olivier Carton, and Chloe Rispal. Logic and rational languages of words indexed by linear orderings. *Theory Comput. Syst.*, 46(4):737–760, 2010.

2 Alexis Bès and Olivier Carton. Algebraic characterization of FO for scattered linear orderings. In *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011*, pages 67–81, 2011.

3 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.

4 Olivier Carton, Thomas Colcombet, and Gabriele Puppis. Regular languages of words over countable linear orderings. In *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011, Proceedings, Part II*, pages 125–136, 2011.

5 Thomas Colcombet. Factorization forests for infinite words and applications to countable scattered linear orderings. *Theor. Comput. Sci.*, 411(4-5):751–764, 2010.

6 Thomas Colcombet and A. V. Sreejith. Limited set quantifiers over countable linear orderings. In *Automata, Languages, and Programming – 42nd International Colloquium, ICALP 2015, Proceedings, Part II*, pages 146–158, 2015.

7 Julien Cristau. Automata and temporal logic over arbitrary linear time. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009*, pages 133–144, 2009.

8 Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

9 Martin Fürer. The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems). In *Logic and Machines: Decision Problems and Complexity, Proceedings of Symposium Rekursive Kombinatorik*, pages 312–319, 1983.

10 Erich Grädel, Phokion G. Kolaitis, and Moshe Y. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

11 Manfred Kufleitner and Pascal Weil. On FO2 quantifier alternation over words. In *Mathematical Foundations of Computer Science 2009, 34th International Symposium, MFCS 2009*, pages 513–524, 2009.

12 Manfred Kufleitner and Pascal Weil. The FO2 alternation hierarchy is decidable. In *Computer Science Logic (CSL'12) – 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012*, pages 426–439, 2012.

13 Amaldev Manuel. Two variables and two successors. In *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS*, pages 513–524, 2010.

14 Amaldev Manuel and Thomas Zeume. Two-variable logic on 2-dimensional structures. In *Computer Science Logic 2013 (CSL 2013), CSL*, pages 484–499, 2013.

15 Martin Otto. Two variable first-order logic over ordered domains. *J. Symb. Log.*, 66(2):685–702, 2001.

16 Jean-Éric Pin. Mathematical foundations of automata theory.

17 Jean-Eric Pin and Pascal Weil. Polynomial closure and unambiguous product. In *Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Proceedings*, pages 348–359, 1995.

18 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

19 Alexander Rabinovich. Temporal logics over linear time domains are in PSPACE. *Inf. Comput.*, 210:40–67, 2012.

20 Joseph G. Rosenstein. *Linear orderings.* Academic Press New York, 1981.

21 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

22 Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *Developments in Language Theory, 5th International Conference, DLT 2001*, pages 239–250, 2001.

23 Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. *Logical Methods in Computer Science*, 8(1), 2012.
24 S.Shelah. The monadic theory of order. *Ann. of Math.*, 102:379–419, 1975.
25 Pascal Tesson and Denis Therien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500. World Scientific, 2002.
26 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*, STOC'98, pages 234–240. ACM, 1998.
27 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FOˆ2 on words. *Logical Methods in Computer Science*, 5(3), 2009.

Piecewise Testable Languages and Nondeterministic Automata[*]

Tomáš Masopust

Institute of Theoretical Computer Science and Center of Advancing Electronics Dresden (cfaed), TU Dresden, Dresden, Germany
tomas.masopust@tu-dresden.de

Abstract

A regular language is k-piecewise testable if it is a finite boolean combination of languages of the form $\Sigma^* a_1 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$, where $a_i \in \Sigma$ and $0 \le n \le k$. Given a DFA $\mathcal{A}$ and $k \ge 0$, it is an NL-complete problem to decide whether the language $L(\mathcal{A})$ is piecewise testable and, for $k \ge 4$, it is coNP-complete to decide whether the language $L(\mathcal{A})$ is k-piecewise testable. It is known that the depth of the minimal DFA serves as an upper bound on k. Namely, if $L(\mathcal{A})$ is piecewise testable, then it is k-piecewise testable for k equal to the depth of $\mathcal{A}$. In this paper, we show that some form of nondeterminism does not violate this upper bound result. Specifically, we define a class of NFAs, called ptNFAs, that recognize piecewise testable languages and show that the depth of a ptNFA provides an (up to exponentially better) upper bound on k than the minimal DFA. We provide an application of our result, discuss the relationship between k-piecewise testability and the depth of NFAs, and study the complexity of k-piecewise testability for ptNFAs.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases Automata, Logics, Languages, k-piecewise testability, Nondeterminism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.67

1 Introduction

A regular language L over an alphabet Σ is *piecewise testable* if it is a finite boolean combination of languages of the form

$$L_{a_1 a_2 \ldots a_n} = \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$$

where $a_i \in \Sigma$ and $n \ge 0$. If L is piecewise testable, then there exists a nonnegative integer k such that L is a finite boolean combination of languages L_u, where the length of $u \in \Sigma^*$ is at most k. In this case, the language L is called *k-piecewise testable*.

Piecewise testable languages are studied in semigroup theory [2, 3, 28] and in logic over words [10, 29] because of their close relation to first-order logic FO(<). They actually form the first level of the Straubing-Thérien hierarchy [27, 36]. This hierarchy is closely related to the dot-depth hierarchy [7], see more in [23]. They are indeed studied in formal languages and automata theory [20], recently mainly in the context of separation [29, 38]. Although the separability of context-free languages by regular languages is undecidable, separability by piecewise testable languages is decidable [9] (even for some non-context-free families). Piecewise testable languages form a strict subclass of star-free languages, that is, of the limit of the above-mentioned hierarchies or, in other words, of the languages definable by

[*] This work was supported by the by the German Research Foundation (DFG) in Emmy Noether grant KR 4381/1-1 (DIAMOND).

LTL logic. They are investigated in natural language processing [11, 30], in cognitive and sub-regular complexity [31], in learning theory [12, 21], and in databases in the context of XML schema languages [8, 14, 15]. They have been extended from words to trees [4, 13].

Recently, the complexity of computing the minimal k and/or bounds on k for which a piecewise testable language is k-piecewise testable was studied in [14, 19, 20], motivated by applications in databases and in algebra and logic. However, the knowledge of such a k that is either minimal or of reasonable size is of interest in many other applications as well, see, e.g., [24]. The complexity to test whether a piecewise testable language is k-piecewise testable was shown to be coNP-complete for $k \geq 4$ if the language is given as a DFA [19] and PSPACE-complete if the language is given as an NFA [25]. The complexity for DFAs and $k < 4$ is discussed in detail in [25]. The best upper bound on k known so far is given by the depth of the minimal DFA [20].

In this paper, we define a class of NFAs, called ptNFAs, that characterizes piecewise testable languages. This characterization is based on purely structural properties, therefore it is NL-complete to check whether an NFA is a ptNFA (Theorem 5). We show that the depth of ptNFAs also provides an upper bound on k-piecewise testability (Theorem 8) and that this new bound is up to exponentially lower than the one given by minimal DFAs (Section 3 and Theorem 15). We further show that this property does not hold for general NFAs, and that the gap between k-piecewise testability and the depth of NFAs can be arbitrarily large (Lemma 12). The opposite implication of Theorem 8 does not hold and a brief discussion is provided. We give a non-trivial application of our result in Section 5, where we also provide more discussion. Finally, in Section 6, we study the complexity of k-piecewise testability for ptNFAs.

The paper is organized as follows. Section 2 presents basic notions and definitions, fixes the notation, and defines the ptNFAs. Section 3 motivates and demonstrates Theorem 8 on a simple example. Section 4 then proves Theorem 8 and the related results. Section 5 provides a non-trivial application and further discussion. Section 6 recalls the known complexity results and studies the complexity of the related problems for ptNFAs. Section 7 concludes the paper.

2 Preliminaries and Definitions

We assume that the reader is familiar with automata theory, see, e.g., [1]. The cardinality of a set A is denoted by $|A|$ and the power set of A by 2^A. An alphabet, Σ, is a finite nonempty set; the elements of an alphabet are called symbols or letters. The free monoid generated by Σ is denoted by Σ^*. A word over Σ is any element of Σ^*; the empty word is denoted by ε. For a word $w \in \Sigma^*$, $\mathrm{alph}(w) \subseteq \Sigma$ denotes the set of all letters occurring in w, and $|w|_a$ denotes the number of occurrences of letter a in w. A language over Σ is a subset of Σ^*. For a language L over Σ, let $\overline{L} = \Sigma^* \setminus L$ denote the complement of L.

A *nondeterministic finite automaton* (NFA) is a quintuple $\mathcal{A} = (Q, \Sigma, \cdot, I, F)$, where Q is a finite nonempty set of states, Σ is an input alphabet, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of accepting states, and $\cdot : Q \times \Sigma \to 2^Q$ is the transition function that can be extended to the domain $2^Q \times \Sigma^*$ by induction. The language *accepted* by $\mathcal{A}$ is the set $L(\mathcal{A}) = \{w \in \Sigma^* \mid I \cdot w \cap F \neq \emptyset\}$. In what follows, we usually omit $\cdot$ and write simply Iw instead of $I \cdot w$.

A *path* π from a state q_0 to a state q_n under a word $a_1a_2 \cdots a_n$, for some $n \geq 0$, is a sequence of states and input symbols $q_0a_1q_1a_2 \ldots q_{n-1}a_nq_n$ such that $q_{i+1} \in q_i \cdot a_{i+1}$, for all $i = 0, 1, \ldots, n-1$. The path π is *accepting* if $q_0 \in I$ and $q_n \in F$. We use the notation

$q_0 \xrightarrow{a_1a_2\cdots a_n} q_n$ to denote that there exists a path from q_0 to q_n under the word $a_1a_2\cdots a_n$. A path is *simple* if all states of the path are pairwise distinct. The number of states on the longest simple path of $\mathcal{A}$, starting in an initial state, decreased by one (i.e., the number of transitions on that path) is called the *depth* of the automaton $\mathcal{A}$, denoted by $\mathit{depth}(\mathcal{A})$.

The NFA $\mathcal{A}$ is *complete* if for every state q of $\mathcal{A}$ and every letter a in Σ, the set $q \cdot a$ is nonempty, that is, in every state, a transition under every letter is defined.

Let $\mathcal{A} = (Q, \Sigma, \cdot, I, F)$ be an NFA, and let p be a state of $\mathcal{A}$. The sub-automaton of $\mathcal{A}$ induced by state p is the automaton $\mathcal{A}_p = (\mathit{reach}(p), \Sigma, \cdot_p, p, F \cap \mathit{reach}(p))$ with state p being the sole initial state and with only those states of $\mathcal{A}$ that are reachable from p; formally, $\mathit{reach}(p)$ denotes the set of all states reachable from state p in $\mathcal{A}$ and $\cdot_p$ is a restriction of $\cdot$ to $\mathit{reach}(p) \times \Sigma$.

The NFA $\mathcal{A}$ is *deterministic* (DFA) if $|I| = 1$ and $|q \cdot a| = 1$ for every state q in Q and every letter a in Σ. Then the transition function $\cdot$ is a map from $Q \times \Sigma$ to Q that can be extended to the domain $Q \times \Sigma^*$ by induction. Two states of a DFA are *distinguishable* if there exists a word w that is accepted from one of them and rejected from the other. A DFA is *minimal* if all its states are reachable and pairwise distinguishable.

Let $\mathcal{A} = (Q, \Sigma, \cdot, I, F)$ be an NFA. The reachability relation $\leq$ on the set of states is defined by $p \leq q$ if there exists a word w in Σ^* such that $q \in p \cdot w$. The NFA $\mathcal{A}$ is *partially ordered* if the reachability relation $\leq$ is a partial order. In other words, the automaton is acyclic, but self-loops are allowed. Therefore, partially ordered automata are sometimes also called acyclic automata. For two states p and q of $\mathcal{A}$, we write $p < q$ if $p \leq q$ and $p \neq q$. A state p is *maximal* if there is no state q such that $p < q$.

An NFA $\mathcal{A} = (Q, \Sigma, \cdot, I, F)$ can be turned into a directed graph $G(\mathcal{A})$ with the set of vertices Q, where a pair (p, q) in $Q \times Q$ is an edge in $G(\mathcal{A})$ if there is a transition from p to q in $\mathcal{A}$. For $\Gamma \subseteq \Sigma$, we define the directed graph $G(\mathcal{A}, \Gamma)$ with the set of vertices Q by considering all those transitions that correspond to letters in Γ. For a state p, let $\Sigma(p) = \{a \in \Sigma \mid p \in p \cdot a\}$ denote the set of all letters under which the NFA $\mathcal{A}$ has a self-loop in state p. Let $\mathcal{A}$ be a partially ordered NFA. If for every state p of $\mathcal{A}$, state p is the unique maximal state of the connected component of $G(\mathcal{A}, \Sigma(p))$ containing p, then we say that the NFA satisfies the *unique maximal state (UMS) property*.

An equivalent notion to the UMS property for minimal DFAs has been introduced in the literature. A DFA $\mathcal{A}$ over Σ is *confluent* if, for every state q of $\mathcal{A}$ and every pair of letters a, b in Σ, there exists a word w in $\{a, b\}^*$ such that $(qa)w = (qb)w$.

We adopt the notation $L_{a_1a_2\cdots a_n} = \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \cdots \Sigma^* a_n \Sigma^*$ from [20]. For two words $v = a_1a_2\cdots a_n$ and $w \in L_v$, we say that v is a *subsequence* of w or that v can be *embedded* into w, denoted by $v \preccurlyeq w$. For $k \geq 0$, let $\mathit{sub}_k(v) = \{u \in \Sigma^* \mid u \preccurlyeq v, |u| \leq k\}$. For two words w_1, w_2, we define $w_1 \sim_k w_2$ if and only if $\mathit{sub}_k(w_1) = \mathit{sub}_k(w_2)$. Note that $\sim_k$ is a congruence with finite index.

The following is well known.

▶ **Fact 1** ([33]). *Let L be a regular language, and let $\sim_L$ denote the Myhill congruence [26]. A language L is k-piecewise testable if and only if $\sim_k \subseteq \sim_L$. Moreover, L is a finite union of $\sim_k$ classes.*

In what follows, we will use this fact in several proofs in the form that if L is not k-piecewise testable, then there exist two words u and v such that $u \sim_k v$ and $|L \cap \{u, v\}| = 1$.

▶ **Fact 2.** *Let L be a language recognized by the minimal DFA $\mathcal{A}$. The following is equivalent.*

1. *The language L is piecewise testable.*

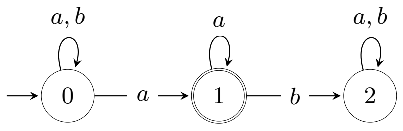


Figure 1 Confluent automaton accepting a non-piecewise testable language.

2. *The minimal DFA $\mathcal{A}$ is partially ordered and confluent [20].*
3. *The minimal DFA $\mathcal{A}$ is partially ordered and satisfies the UMS property [37].*

We now define a special class of nondeterministic automata called ptNFAs. The name comes from piecewise testable, since, as we show below, they characterize piecewise testable languages. And indeed include all minimal DFAs recognizing piecewise testable languages.

▶ **Definition 3.** An NFA $\mathcal{A}$ is called a *ptNFA* if it is partially ordered, complete, and satisfies the UMS property.

The reason why we use the UMS property in the definition of ptNFAs rather than confluence is simply because confluence does not naturally generalize to NFAs as shown in Example 4 below. Moreover, it is known that partially ordered NFAs characterize the level $\frac{3}{2}$ of the Straubing-Thérien hierarchy [32] and that partially ordered NFAs satisfying that $q \in q \cdot a$ implies $q \cdot a = \{q\}$ characterize $\mathcal{R}$-trivial languages [5, 22]. It can be shown that adding confluence and completeness on top of these properties results in ptNFAs.

▶ **Example 4.** Consider the automaton depicted in Figure 1. The notion of confluence is not clear for NFAs. If we consider the point of view that whenever the computation is split, a common state can be reached under a word over the splitting alphabet, then this automaton is confluent. However, it does not satisfy the UMS property and its language is not piecewise testable; there is an infinite sequence $a, ab, aba, abab, \ldots$ that alternates between accepted and non-accepted words, which implies that there is a non-trivial cycle in the corresponding minimal DFA and, thus, it proves non-piecewise testability by Fact 2.

Note that to check whether an NFA is a ptNFA requires to check whether the automaton is partially ordered, complete and satisfies the UMS property. The violation of these properties can be tested by several reachability tests, hence its complexity belongs to coNL=NL. On the other hand, to check the properties is NL-hard even for minimal DFAs [6]. Thus, we have the following.

▶ **Theorem 5.** *It is NL-complete to check whether an NFA is a ptNFA.*

3 Motivation and an Example

Considering applications, such as XML, where the alphabet can hardly be considered as fixed, the results of [19] (cf. Theorem 18 below) say that it is intractable to compute the minimal k for which a piecewise testable language is k-piecewise testable, unless P=NP. This leads to the investigation of reasonably small upper bounds. The result of [20] says that k is bounded by the depth of the minimal DFA. However, applications usually require to work with NFAs, which motivates the research of this paper. Another motivation comes from a simple observation that, given several DFAs, a result of an operation can lead to an NFA that in some sense still has the DFA-like properties, see more discussion below. Moreover, it seems to be a human nature to use a kind of nondeterminism, for instance, to reuse already defined parts as demonstrated here on a very simple example.

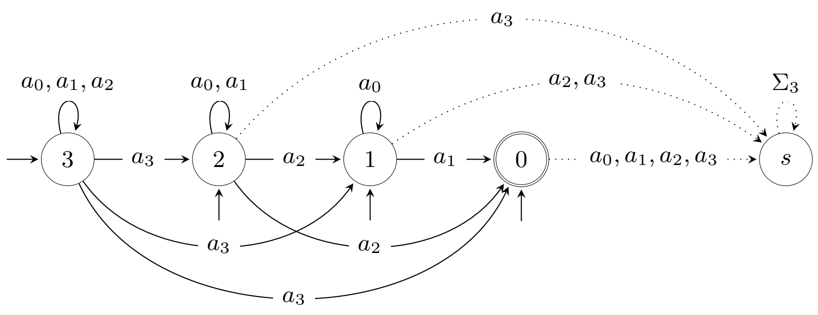


Figure 2 Automaton $\mathcal{A}_3$; the dotted transitions depict the completion of $\mathcal{A}_3$.

Let $L_0 = \{\varepsilon\}$ be a language over the alphabet $\Sigma_0 = \{a_0\}$. Assume that the language L_i over Σ_i is defined, and let $L_{i+1} = L_i \cup \Sigma_i^* a_{i+1} L_i$ over $\Sigma_{i+1} = \Sigma_i \cup \{a_{i+1}\}$, where a_{i+1} is a new symbol not in Σ_i. We now construct the NFAs for the languages L_i,

$$\mathcal{A}_i = (\{0, 1, \ldots, i\}, \{a_0, a_1, \ldots, a_i\}, \cdot, \{0, 1, \ldots, i\}, \{0\})$$

where $\ell \cdot a_j = \ell$ if $i \geq \ell > j \geq 0$ and $\ell \cdot a_\ell = \{0, 1, \ldots, \ell - 1\}$ if $i \geq \ell \geq 1$. The automaton $\mathcal{A}_3$ is depicted in Figure 2. The dotted transitions are to complete the NFA in the meaning that $\ell \cdot a \neq \emptyset$ for any state ℓ and letter a.

Although the example is very simple, the reader can see the point of the construction in nondeterministically reusing the existing parts.

Now, to decide whether the language is piecewise testable and, if so, to obtain an upper bound on its k-piecewise testability, a naive application of the known results for DFAs requires to compute the minimal DFA. Doing so shows that L_i is piecewise testable. However, the minimal DFA for the language L_i is of exponential size and its depth is $2^{i+1} - 1$, cf. [25], which implies that L_i is $(2^{i+1} - 1)$-piecewise testable. Another way is to use the PSPACE algorithm of [25] to compute the minimal k. Both approaches are basically of the same complexity.

This is the place, where our result comes into the picture. According to Theorem 8 proved in the next section, the easily testable structural properties say that the language L_i is $(i + 1)$-piecewise testable. This provides an exponentially better upper bound for every language L_i than the technique based on minimal DFAs. Finally, we note that it can be shown that L_i is not i-piecewise testable, so the bound is tight for L_i.

4 Piecewise Testability and Nondeterminism

In this section, we establish a relation between piecewise testable languages and nondeterministic automata and generalize the bound given by the depth of DFAs to ptNFAs. We first recall the known result for DFAs.

▶ **Theorem 6** ([20]). *Let $\mathcal{A}$ be a partially ordered and confluent DFA. If the depth of $\mathcal{A}$ is k, then the language $L(\mathcal{A})$ is k-piecewise testable.*

This result is currently the best known structural upper bound on k-piecewise testability. The opposite implication of the theorem does not hold and we have shown in [25] (see also Section 3) that this bound can be exponentially far from the minimal value of k.

This observation has motivated our investigation of the relationship between piecewise testability and the depth of NFAs. We have already generalized a structural automata characterization for piecewise testability from DFAs to NFAs as follows.

▶ **Theorem 7** ([25]). *A regular language is piecewise testable if and only if it is recognized by a ptNFA.*

We now generalize Theorem 6 to ptNFAs and discuss the relation between the depth of NFAs and k-piecewise testability in more detail. An informal idea behind the proof is that every ptNFA can be "decomposed" into a finite number of partially ordered and confluent DFAs. We now formally prove the theorem by generalizing the proof of Theorem 6 given in [20].

▶ **Theorem 8.** *If the depth of a ptNFA $\mathcal{A}$ is k, then the language $L(\mathcal{A})$ is k-piecewise testable.*

The proof of Theorem 8 follows directly from Lemmas 9 and 11 proved below.

▶ **Lemma 9.** *Let $\mathcal{A}$ be a ptNFA with I denoting the set of initial states. Then the language $L(\mathcal{A}) = \bigcup_{i\in I} L(\mathcal{A}_i)$, where every sub-automaton $\mathcal{A}_i$ is a ptNFA.*

Based on the previous lemma, it is sufficient to show the theorem for ptNFAs with a single initial state. We make use of the following lemma.

▶ **Lemma 10** ([20]). *Let $\ell \geq 1$, and let $u, v \in \Sigma^*$ be such that $u \sim_\ell v$. Let $u = u'au''$ and $v = v'av''$ such that $a \notin \mathrm{alph}(u'v')$. Then $u'' \sim_{\ell-1} v''$.*

▶ **Lemma 11.** *Let $\mathcal{A}$ be a ptNFA with a single initial state and depth k. Then the language $L(\mathcal{A})$ is k-piecewise testable.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \cdot, i, F)$. If the depth of $\mathcal{A}$ is 0, then $L(\mathcal{A})$ is either $\emptyset$ or Σ^*, which are both 0-piecewise testable by definition. Thus, assume that the depth of $\mathcal{A}$ is $\ell \geq 1$ and that the claim holds for ptNFAs of depth less than ℓ. Let $u, v \in \Sigma^*$ be such that $u \sim_\ell v$. We prove that u is accepted by $\mathcal{A}$ if and only if v is accepted by $\mathcal{A}$.

Assume that u is accepted by $\mathcal{A}$ and fix an accepting path of u in $\mathcal{A}$. If $\mathrm{alph}(u) \subseteq \Sigma(i)$, then the UMS property of $\mathcal{A}$ implies that $i \in F$. Therefore, v is also accepted in i. If $\mathrm{alph}(u) \not\subseteq \Sigma(i)$, then $u = u'au''$ and $v = v'bv''$, where $u', v' \in \Sigma(i)^*$, $a, b \in \Sigma \setminus \Sigma(i)$, and $u'', v'' \in \Sigma^*$. Let $p \in i \cdot a$ be a state on the fixed accepting path of u. Let $\mathcal{A}_p = (\mathit{reach}(p), \Sigma, \cdot_p, p, F \cap \mathit{reach}(p))$ be a sub-automaton of $\mathcal{A}$ induced by state p. Note that $\mathcal{A}_p$ is a ptNFA. By assumption, $\mathcal{A}_p$ accepts u'' and the depth of $\mathcal{A}_p$ is at most $\ell - 1$.

If $a = b$, Lemma 10 implies that $u'' \sim_{\ell-1} v''$. By the induction hypothesis, u'' is accepted by $\mathcal{A}_p$ if and only if v'' is accepted by $\mathcal{A}_p$. Hence, $v = v'av''$ is accepted by $\mathcal{A}$.

If $a \neq b$, then $u = u'au_0''bu_1''$ and $v = v'bv_0''av_1''$, where $b \notin \mathrm{alph}(u'au_0'')$ and $a \notin \mathrm{alph}(v'bv_0'')$. Then

$$u'' = u_0''bu_1'' \sim_{\ell-1} v_0''av_1'' = v''$$

because, by Lemma 10,

$$\mathit{sub}_{\ell-1}(u_0''bu_1'') = \mathit{sub}_{\ell-1}(v_1'') \subseteq \mathit{sub}_{\ell-1}(v_0''av_1'') = \mathit{sub}_{\ell-1}(u_1'') \subseteq \mathit{sub}_{\ell-1}(u_0''bu_1'')\,. \qquad (*)$$

If $p \in i \cdot b$, the induction hypothesis implies that v'' is accepted by $\mathcal{A}_p$, hence $v = v'bv''$ is accepted by $\mathcal{A}$.

If $p \notin i \cdot b$, let $q \in i \cdot b$. By the UMS property of $\mathcal{A}$, there exists a word $w \in \{a, b\}^*$ such that $pw = qw = r$, for some state r with $a, b \in \Sigma(r)$. Indeed, there exists w_1 and a unique maximal state r with respect to $\{a,b\}$ such that $pw_1 = \{r\}$ and $a,b \in \Sigma(r)$. By the UMS property, there exists w_2 over $\{a,b\}$ such that $qw_1w_2 = \{r\}$. Let $w = w_1w_2$. We now show that $wu'' \sim_{\ell-1} u''$ by induction on the length of w. There is nothing to show for $w = \varepsilon$. Thus, assume that $w = xw'$, for $x \in \{a,b\}$, and that $w'u'' \sim_{\ell-1} u''$. Notice that (*) shows that $u'' \sim_{\ell-1} v_1'' \sim_{\ell-1} v'' \sim_{\ell-1} u_1''$. This implies that $sub_{\ell-1}(v_1'') \subseteq sub_{\ell-1}(av_1'') \subseteq sub_{\ell-1}(v_0''av_1'') = sub_{\ell-1}(v'') = sub_{\ell-1}(v_1'')$, which shows that $av_1'' \sim_{\ell-1} v_1''$. Analogously, we can show that $bu_1'' \sim_{\ell-1} u_1''$. If $x = a$, then $w'u'' \sim_{\ell-1} u'' \sim_{\ell-1} v_1''$ implies that $aw'u'' \sim_{\ell-1} av_1'' \sim_{\ell-1} v_1'' \sim_{\ell-1} u''$. If $x = b$, then $w'u'' \sim_{\ell-1} u'' \sim_{\ell-1} u_1''$ implies that $bw'u'' \sim_{\ell-1} bu_1'' \sim_{\ell-1} u_1'' \sim_{\ell-1} u''$. Therefore, $wu'' \sim_{\ell-1} u''$; similarly, $wv'' \sim_{\ell-1} v''$.

Finally, using the induction hypothesis (of the main statement) on $\mathcal{A}_p$, we get that u'' is accepted by $\mathcal{A}_p$ if and only if wu'' is accepted by $\mathcal{A}_p$, which is if and only if u'' is accepted by $\mathcal{A}_r$. Since $u'' \sim_{\ell-1} v''$, the induction hypothesis applied on $\mathcal{A}_r$ gives that u'' is accepted by $\mathcal{A}_r$ if and only if v'' is accepted by $\mathcal{A}_r$. However, this is if and only if wv'' is accepted by $\mathcal{A}_q$. Using the induction hypothesis on $\mathcal{A}_q$, we obtain that wv'' is accepted by $\mathcal{A}_q$ if and only if v'' is accepted by $\mathcal{A}_q$. Together, the assumption that u'' is accepted by $\mathcal{A}_p$ implies that v'' is accepted by $\mathcal{A}_q$. Hence $v = v'bv''$ is accepted by $\mathcal{A}$, which completes the proof. ◀

In other words, the previous theorem says that if k is the minimum number for which a piecewise testable language L is k-piecewise testable, then the depth of any ptNFA recognizing L is at least k.

It is natural to ask whether this property holds for any NFA recognizing the language L. The following result shows that it is not the case. Actually, for any natural number ℓ, there exists a piecewise testable language such that the difference between its k-piecewise testability and the depth of an NFA is at least ℓ.

▶ **Lemma 12.** *For every $k \geq 3$, there exists a k-piecewise testable language that is recognized by an NFA of depth at most $\lfloor \frac{k}{2} \rfloor$.*

Proof. For every $i \geq 1$, let $L_i = a^i + a^{2i+1} \cdot a^*$. We show that the language L_i is $(2i+1)$-piecewise testable and that there exists an NFA of depth at most i recognizing it.

The minimal DFA for L_i consists of $2i+1$ states $\{0,1,\ldots,2i+1\}$, where 0 is the initial state, i and $2i+1$ are accepting, $p \cdot a = p+1$ for $p < 2i+1$, and $(2i+1) \cdot a = 2i+1$. The depth is $2i+1$, which shows that L_i is $(2i+1)$-piecewise testable. Notice that $a^{2i} \sim_{2i} a^{2i+1}$, but a^{2i} does not belong to L_i, hence L_i is not $2i$-piecewise testable.

The NFA for L_i consists of two cycles of length $i+1$, the structure is depicted in Figure 3. The initial state is state 0 and the solely accepting state is state i. The automaton accepts L_i. Indeed, it accepts a^i and no shorter word. After reading a^i, the automaton is in state i or i'. In both cases, the shortest nonempty path to the single accepting state i is of length $i+1$. Thus, the automaton accepts a^{2i+1}, but nothing between a^i and a^{2i+1}. Finally, using the self-loop in state i', the automaton accepts $a^ia^*a^{i+1} = a^{2i+1}a^*$. The depth of the automaton is i. ◀

Piecewise testability and the depth of NFAs. Theorem 8 gives rise to a question whether the opposite implication holds true. This is not the case. Notice that although the depth of ptNFAs is more suitable to provide bounds on k-piecewise testability, the depth is significantly influenced by the size of the input alphabet. For instance, for an alphabet Σ, the language $L = \bigcap_{a\in\Sigma} L_a$ of all words containing all letters of Σ is a 1-piecewise testable language such that any NFA recognizing it requires at least $2^{|\Sigma|}$ states and is of depth $|\Sigma|$, cf. [25]. The

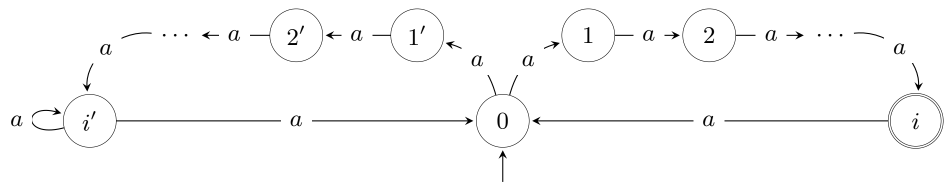


Figure 3 The NFA of depth i recognizing L_i.

depth follows from the fact that the shortest accepted word is of length $|\Sigma|$, hence any path from an initial state to an accepting state must be of length at least $|\Sigma|$.

The dependence on the alphabet is even stronger as shown below.

▶ **Lemma 13.** *For any alphabet of cardinality $n > 1$, there exists an n^2-piecewise testable language such that any NFA recognizing it is of depth at least n^{n-1}.*

Proof. Let $L_n(k)$ denote the maximal length of the shortest representatives of the $\sim_k$-classes over an n-element alphabet. It was shown in [18] that $(L_n(k) + 1)\log n > (\frac{k}{n})^{n-1}\log(\frac{k}{n})$. Setting $k = n^2$ then gives that $L_n(n^2) \geq n^{n-1}$. Let L be a language defined by a single $\sim_{n^2}$-class with shortest representatives of length $L_n(n^2)$. Then L is n^2-piecewise testable, since it is defined as a union of $\sim_{n^2}$ classes. Consider any NFA recognizing L. Since the shortest word of L is of length $L_n(n^2)$, any path from an initial state to an accepting state must be of length at least $L_n(n^2)$. ◀

Recall that it was independently shown in [19, 25] that, given a k-piecewise testable language over an n-letter alphabet, the tight upper bound on the depth of the minimal DFA recognizing it is $\binom{k+n}{k} - 1$. In other words, this formula gives the tight upper bound on the depth of the $\sim_k$-canonical DFA [25] over an n element alphabet. A related question on the size of this DFA is still open, see [18] for more details.

▶ **Theorem 14** ([19, 25])**.** *For any natural numbers k and n, the depth of the minimal DFA recognizing a k-piecewise testable language over an n-letter alphabet is at most $\binom{k+n}{k} - 1$. The bound is tight for any k and n.*

The lower bound for NFAs and ptNFAs remains open.

5 Application and Discussion

The reader might have noticed that the reverse of the automaton $\mathcal{A}_i$ constructed in Section 3 is deterministic and, when made complete, it satisfies the conditions of Fact 2. Since, by definition, a language is k-piecewise testable if and only if its reverse is k-piecewise testable, this observation provides the same upper bound $i+1$ on k-piecewise testability of the language $L(\mathcal{A}_i)$. However, this is just a coincidence and it is not difficult to find an example of a ptNFA whose reverse is not deterministic.

Since both the minimal DFA for L and the minimal DFA for L^R provide an upper bound on k, it could seem reasonable to compute both DFAs in parallel with the hope that (at least) one of them will be computed in a reasonable (polynomial) time. Although this may work for many cases (including the case of Section 3), we now show that there are cases where both the DFAs are of exponential size.

▶ **Theorem 15.** *For every $n \geq 0$, there exists a $(2n+1)$-state ptNFA $\mathcal{B}$ such that the depth of both the minimal DFA for $L(\mathcal{B})$ and the minimal DFA for $L(\mathcal{B})^R$ are exponential with respect to n.*

Proof sketch. The idea of the proof is to make use of the automaton $\mathcal{A}_i$ constructed in Section 3 to build a ptNFA $\mathcal{B}_i$ such that $L(\mathcal{B}_i) = L(\mathcal{A}_i) \cdot L(\mathcal{A}_i)^R$. Then $L(\mathcal{B}_i) = L(\mathcal{B}_i)^R$ and it can be shown that the minimal DFA recognizing the language $L(\mathcal{B}_i)$ requires an exponential number of states compared to $\mathcal{B}_i$. Namely, the depth of both the minimal DFA for $L(\mathcal{B}_i)$ and the minimal DFA for $L(\mathcal{B}_i)^R$ are of length at least $2^{i+1} - 1$. ◀

The previous proof provides another motivation to investigate nondeterministic automata for piecewise testable languages. Given several DFAs, the result of a sequence of operations may result in an NFA that preserves some good properties. Namely, the language $L(\mathcal{B}_i)$ from the previous proof is a result of the operation concatenation of a language L^R with L, where L is a piecewise testable language given as a DFA.

It immediately follows from Theorem 8 that the language $L(\mathcal{B}_i)$ is $(2i+1)$-piecewise testable. This result is not easily derivable from known results, which are either in PSPACE or require to compute an exponentially larger minimal DFA, which provides only the information that the language $L(\mathcal{B}_i)$ is k-piecewise testable for some $k \geq 2^{i+1} - 1$.

Even the information that the language $L(\mathcal{B}_i)$ is of the form $L^R \cdot L$, for a piecewise testable language L, does not seem very helpful, since piecewise testable languages are not closed under concatenation, even with its own reverse, as we show in the example below.

▶ **Example 16.** Let L be the language over the alphabet $\{a, b, c\}$ defined by the regular expression $ab^* + c(a+b)^*$. The reader can construct the minimal DFA for L and check that the properties of Fact 2 are satisfied. In addition, the depth of the minimal DFA is two, hence the language is 2-piecewise testable. Since the properties of Theorem 19 (see below) are not satisfied, the language L is not 1-piecewise testable.

On the other hand, the reader can notice that the sequence $ca, cab, caba, cabab, cababa, \ldots$ is an infinite sequence where every word on the odd position belongs to $L \cdot L^R$, whereas every word on the even position does not. This means that there exists a cycle in the minimal DFA recognizing $L \cdot L^R$, which shows that $L \cdot L^R$ is not a piecewise testable language according to Fact 2. The reader can also directly compute the minimal DFA for $L \cdot L^R$ and notice a non-trivial cycle in it.

To complete this part, we show that the language $L(\mathcal{B}_i)$ is not $(2i)$-piecewise testable. Thus, there are no ptNFAs recognizing the language $L(\mathcal{B}_i)$ with depth less than $2i+1$.

▶ **Lemma 17.** *For every $i \geq 0$, the language $L(\mathcal{B}_i)$ is not $2i$-piecewise testable.*

6 Complexity

In this section, we first give an overview of known complexity results and characterization theorems for DFAs and then discuss the related complexity for ptNFAs.

Simon [33] proved that piecewise testable languages are exactly those regular languages whose syntactic monoid is $\mathcal{J}$-trivial, which shows decidability of the problem whether a regular language is piecewise testable. Later, Stern proved that the problem is decidable in polynomial time for languages represented as minimal DFAs [34], and Cho and Huynh [6] showed that it is NL-complete for DFAs. Trahtman [37] improved Stern's result by giving an algorithm quadratic in the number of states of the minimal DFA, and Klíma and Polák [20] presented an algorithm quadratic in the size of the alphabet of the minimal DFA. If the language is represented as an NFA, the problem is PSPACE-complete [16] (see more details below).

By definition, a regular language is piecewise testable if there exists $k \geq 0$ such that it is k-piecewise testable. It gives rise to a question to find such a minimal k. The *k-piecewise testability problem* asks, given an automaton, whether it recognizes a k-piecewise testable language. The problem is trivially decidable because there are only finitely many k-piecewise testable languages over a fixed alphabet. The coNP upper bound on k-piecewise testability for DFAs was independently shown in [14, 25].[1] The coNP-completeness for $k \geq 4$ was recently shown in [19]. The complexity holds even if k is given as part of the input. The complexity analysis of the problem for $k < 4$ is provided in [25]. We recall the results we need later.

▶ **Theorem 18** ([19]). *For $k \geq 4$, to decide whether a DFA represents a k-piecewise testable language is coNP-complete. It remains coNP-complete even if the parameter $k \geq 4$ is given as part of the input. For a fixed alphabet, the problem is decidable in polynomial time.*

It is not difficult to see that, given a minimal DFA, it is decidable in constant time whether its language is 0-piecewise testable, since it is either empty or Σ^*.

▶ **Theorem 19** (1-piecewise testability DFAs, [25]). *Let $\mathcal{A} = (Q, \Sigma, \cdot, i, F)$ be a minimal DFA. Then $L(\mathcal{A})$ is 1-piecewise testable if and only if* (i) *for every $p \in Q$ and $a \in \Sigma$, $paa = pa$ and* (ii) *for every $p \in Q$ and $a, b \in \Sigma$, $pab = pba$. The problem is in AC^0.*

It is not hard to see that this result does not hold for ptNFAs. Indeed, one can simply consider a minimal DFA satisfying the properties and add a nondeterministic transition that violates them, but not the properties of ptNFAs. On the other hand, the conditions are still sufficient.

▶ **Lemma 20** (1-piecewise testability ptNFAs). *Let $\mathcal{A} = (Q, \Sigma, \cdot, i, F)$ be a complete NFA. If* (i) *for every $p \in Q$ and $a \in \Sigma$, $paa = pa$ and* (ii) *for every $p \in Q$ and $a, b \in \Sigma$, $pab = pba$, then the language $L(\mathcal{A})$ is 1-piecewise testable.*

Note that any ptNFA $\mathcal{A}$ satisfying (i) must have $|pa| = 1$ for every state p and letter a. If $pa = \{r_1, r_2, \ldots, r_m\}$ with $r_1 < r_2 < \ldots < r_m$, then $paa = pa$ implies that $\{r_1, \ldots, r_m\}a = \{r_1, \ldots, r_m\}$. Then $r_1 \in r_1 a$ and the UMS property says that $r_1 a = \{r_1\}$. By induction, we can show hat $r_i a = \{r_i\}$. Consider the component of $G(\mathcal{A}, \Sigma(r_1))$ containing r_1. Then $r_1, \ldots, r_m$ all belong to this component. Since r_1 is maximal, r_1 is reachable from every r_i under $\Sigma(r_1) \supseteq \{a\}$. However, the partial order $r_1 < \ldots < r_m$ implies that r_1 is reachable from r_i only if $r_i = r_1$. Thus, $|pa| = 1$. However, $\mathcal{A}$ can still have many initial states, which can be seen as a finite union of piecewise testable languages rather then a nondeterminism.

The 2-piecewise testability characterization for DFAs is as follows.

▶ **Theorem 21** (2-piecewise testability DFAs, [25]). *Let $\mathcal{A} = (Q, \Sigma, \cdot, i, F)$ be a minimal partially ordered and confluent DFA. The language $L(\mathcal{A})$ is 2-piecewise testable if and only if for every $a \in \Sigma$ and every state s such that $iw = s$ for some $w \in \Sigma^*$ with $|w|_a \geq 1$, $sba = saba$ for every $b \in \Sigma \cup \{\varepsilon\}$. The problem is NL-complete.*

It is again sufficient for ptNFAs.

[1] Actually, [14] gives the bound NEXPTIME for the problem for NFAs where k is part of the input. The coNP bound for DFAs can be derived from the proof omitted in the conference version. The problem is formulated in terms of separability, hence it requires the NFA for the language and for its complement.

▶ **Lemma 22** (2-piecewise testability ptNFAs). *Let $\mathcal{A} = (Q, \Sigma, \cdot, i, F)$ be a ptNFA. If for every $a \in \Sigma$ and every state s such that $iw = s$ for some $w \in \Sigma^*$ with $|w|_a \geq 1$, $sba = saba$ for every $b \in \Sigma \cup \{\varepsilon\}$, then the language $L(\mathcal{A})$ is 2-piecewise testable.*

Considering Theorem 18, the lower bound for DFAs is indeed a lower bound for ptNFAs. Thus, we immediately have that the k-piecewise testability problem for ptNFAs is coNP-hard for $k \geq 4$. We now show that it is actually coNP-hard for every $k \geq 0$. The proof is split into two lemmas.

The proof of the following lemma is based on the proof that the non-equivalence problem for regular expressions with operations union and concatenation is NP-complete, even if one of them is of the form Σ^n for some fixed n [17, 35].

▶ **Lemma 23.** *The 0-piecewise testability problem for ptNFAs is coNP-hard (even if the alphabet is binary).*

It seems natural that the $(k + 1)$-piecewise testability problem is not easier then the k-piecewise testability problem. We now formalize this intuition. We also point out that our reduction introduces a new symbol to the alphabet.

▶ **Lemma 24.** *For $k \geq 0$, k-piecewise testability is polynomially reducible to $(k+1)$-piecewise testability.*

Together, since the k-piecewise testability problem for NFAs is in PSPACE [25], we have the following result.

▶ **Theorem 25.** *For $k \geq 0$, the k-piecewise testability problem for ptNFAs is coNP-hard and in PSPACE.*

The case of a fixed alphabet. The previous discussion is for the general case where the alphabet is arbitrary and considered as part of the input. In this subsection, we assume that the alphabet is fixed. In this case, it is shown in the arxiv versions v1–v4 of [18] that the length of the shortest representatives of the $\sim_k$-classes is bounded by the number $\left(\frac{k+2c-1}{c}\right)^c$, where c is the cardinality of the alphabet. This gives us the following result for 0-piecewise testability for ptNFAs.

▶ **Lemma 26.** *For a fixed alphabet Σ with $c = |\Sigma| \geq 2$, the 0-piecewise testability problem for ptNFAs is coNP-complete.*

Proof. The hardness follows from Lemma 23, since it is sufficient to use a binary alphabet.

We now prove the membership. Let $\mathcal{A}$ be a ptNFA over Σ of depth d recognizing a nonempty language (this can be checked in NL). Then the language $L(\mathcal{A})$ is d-piecewise testable by Theorem 8. This means that if $v \sim_d u$, then either both u and v are accepted or both are rejected by $\mathcal{A}$. Now, the language $L(\mathcal{A}) \neq \emptyset$ is not 0-piecewise testable if and only if $L(\mathcal{A})$ is non-universal. Since Σ is fixed, the shortest representative of any of the $\sim_d$-classes is of length less than $\left(\frac{d+2c-1}{c}\right)^c = O(d^c)$, which is polynomial in the depth of $\mathcal{A}$. Thus, if the language $L(\mathcal{A})$ is not universal, then the nondeterministic algorithm can guess a shortest representative of a non-accepted $\sim_d$-class and verify the guess in polynomial time. ◀

We can now generalize this result to k-piecewise testability.

▶ **Theorem 27.** *Let Σ be a fixed alphabet with $c = |\Sigma| \geq 3$, and let $k \geq 0$. Then the problem to decide whether the language of a ptNFA $\mathcal{A}$ over Σ is k-piecewise testable is coNP-complete.*

Table 1 Complexity of k-piecewise testability – an overview.

	Unary alphabet	Fixed alphabet	Arbitrary alphabet	
			$k \leq 3$	$k \geq 4$
DFA	P	P [19]	NL-complete [25]	coNP-complete [19]
ptNFA	P	coNP-complete	PSPACE & coNP-hard	
NFA	coNP-complete	PSPACE-complete [25]	PSPACE-complete [25]	

Note that this is in contrast with the analogous result for DFAs, cf. Theorem 18, where the problem is in P for DFAs over a fixed alphabet. In addition, the hardness part of the proof of the previous theorem gives us the following corollary, which does not follow from the hardness proof of [19], since the proof there requires a growing alphabet.

► **Corollary 28.** *The k-piecewise testability problem for ptNFAs over an alphabet Σ is coNP-hard for $k \geq 0$ even if $|\Sigma| = 3$.*

The case of a unary alphabet. Lemma 26 (resp. Lemma 23) requires at least two letters in the alphabet to prove coNP-hardness. Thus, it remains to consider the case of a unary alphabet. We now show that the problem is simpler in the unary case, unless P=NP. Namely, a similar argument as in the proof of Lemma 26, improved by the fact that the length of the shortest representatives of $\sim_k$-classes is bounded by the depth of the ptNFA, gives the following result.

► **Theorem 29.** *The k-piecewise testability problem for ptNFAs over a unary alphabet is decidable in polynomial time. The result holds even if k is given as part of the input.*

In contrast to this, the problem is coNP-cotmplete for general NFAs.

► **Theorem 30.** *Both piecewise testability and k-piecewise testability problems for NFAs over a unary alphabet are coNP-complete.*

The complexity of k-piecewise testability for considered automata is summarized in Table 1. Note that the precise complexity of k-piecewise testability for ptNFAs is not yet known in the case the alphabet is considered as part of the input even for $k = 0$.

7 Conclusion

In this paper, we have defined a class of nondeterministic finite automata (ptNFAs) that characterize piecewise testable languages. We have shown that their depth (exponentially) improves the known upper bound on k-piecewise testability shown in [20] for DFAs. We have discussed several related questions, mainly in comparison with DFAs and NFAs, including the complexity of k-piecewise testability for ptNFAs. It can be noticed that the results for ptNFAs generalize the results for DFAs in the sense that the results for DFAs are consequences of the results presented here. This, however, does not hold for the complexity results.

The complexity of k-piecewise testability for the case where the alphabet is consider as part of the input is left open. Recall that the results of [18] give a lower bound on the maximal length of the shortest representative of a class. Specifically, let $L_n(k)$ denote the maximal length of the shortest representatives of the $\sim_k$-classes over an n-element alphabet. Then $L_n(n^2) \geq n^{n-1}$. Thus, the representative can be of exponential length with respect to the size of the alphabet.

However, we conjecture that the problem is PSPACE-complete. A partial evidence for this is that it is possible to construct, for an alphabet of cardinality n, an $O(n^2)$-state ptNFA such that the (unique) non-accepted word is of length $\binom{2n}{n} - 1$. We leave this for the future work and provide more details in an extended version.

Acknowledgements. We thank the authors of [14] and [19] for providing the full versions of their papers, and O. Klíma for his comments on the preliminary version of the manuscript.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.
2. J. Almeida, J. C. Costa, and M. Zeitoun. Pointlike sets with respect to R and J. *Journal of Pure and Applied Algebra*, 212(3):486–499, 2008.
3. J. Almeida and M. Zeitoun. The pseudovariety J is hyperdecidable. *RAIRO – Theoretical Informatics and Applications*, 31(5):457–482, 1997.
4. M. Bojanczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. *Logical Methods in Computer Science*, 8(3), 2012.
5. J. A. Brzozowski and F. E. Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980.
6. S. Cho and D. T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88(1):99–116, 1991.
7. R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.
8. W. Czerwiński, W. Martens, and T. Masopust. Efficient separability of regular languages by subsequences and suffixes. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 7966 of *LNCS*, pages 150–161, 2013.
9. W. Czerwiński, W. Martens, L. van Rooijen, and M. Zeitoun. A note on decidable separability by piecewise testable languages. In *Fundamentals of Computation Theory (FCT)*, volume 9210 of *LNCS*, pages 173–185, 2015.
10. V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(3):513–548, 2008.
11. J. Fu, J. Heinz, and H. G. Tanner. An algebraic characterization of strictly piecewise languages. In *Theory and Applications of Models of Computation (TAMC)*, volume 6648 of *LNCS*, pages 252–263, 2011.
12. P. García and J. Ruiz. Learning k-testable and k-piecewise testable languages from positive data. *Grammars*, 7:125–140, 2004.
13. P. García and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.
14. P. Hofman and W. Martens. Separability by short subsequences and subwords. In *International Conference on Database Theory (ICDT)*, volume 31 of *LIPIcs*, pages 230–246, 2015.
15. Š. Holub, G. Jirásková, and T. Masopust. On upper and lower bounds on the length of alternating towers. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8634 of *LNCS*, pages 315–326, 2014.
16. Š. Holub, T. Masopust, and M. Thomazo. Alternating towers and piecewise testable separators. *CoRR*, 2014. Submitted. URL: `http://arxiv.org/abs/1409.3943`.

17 H. B. Hunt III. *On the Time and Tape Complexity of Languages.* PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1973.

18 P. Karandikar, M. Kufleitner, and Ph. Schnoebelen. On the index of Simon's congruence for piecewise testability. *Information Processing Letters*, 115(4):515–519, 2015.

19 O. Klíma, M. Kunc, and L. Polák. Deciding k-piecewise testability. Submitted.

20 O. Klíma and L. Polák. Alternative automata characterization of piecewise testable languages. In *Developments in Language Theory (DLT)*, volume 7907 of *LNCS*, pages 289–300, 2013.

21 L. Kontorovich, C. Cortes, and M. Mohri. Kernel methods for learning languages. *Theoretical Computer Science*, 405(3):223–236, 2008.

22 M. Krötzsch, T. Masopust, and M. Thomazo. On the complexity of university for partially ordered NFAs. In *Mathematical Foundations of Computer Science (MFCS)*, volume 58 of *LIPIcs*, pages 62:1–62:14, 2016.

23 M. Kufleitner and A. Lauser. Around dot-depth one. *International Journal of Foundations of Computer Science*, 23(6):1323–1340, 2012.

24 W. Martens, F. Neven, M. Niewerth, and T. Schwentick. Bonxai: Combining the simplicity of DTD with the expressiveness of XML schema. In *Principles of Database Systems (PODS)*, pages 145–156. ACM, 2015.

25 T. Masopust and M. Thomazo. On the complexity of k-piecewise testability and the depth of automata. In *Developments in Language Theory (DLT)*, volume 9168 of *LNCS*, pages 364–376, 2015.

26 J. Myhill. Finite automata and representation of events. Technical report, Wright Air Development Center, 1957.

27 D. Perrin and J.-E. Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32(3):393–406, 1986.

28 D. Perrin and J.-E. Pin. *Infinite words: Automata, semigroups, logic and games*, volume 141 of *Pure and Applied Mathematics.* Elsevier, 2004.

29 T. Place, L. van Rooijen, and M. Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8087 of *LNCS*, pages 729–740, 2013.

30 J. Rogers, J. Heinz, G. Bailey, M. Edlefsen, M. Visscher, D. Wellcome, and S. Wibel. On languages piecewise testable in the strict sense. In *Mathematics of Language (MOL)*, volume 6149 of *LNAI*, pages 255–265, 2010.

31 J. Rogers, J. Heinz, M. Fero, J. Hurst, D. Lambert, and S. Wibel. Cognitive and sub-regular complexity. In *Formal Grammar (FG)*, volume 8036 of *LNCS*, pages 90–108, 2013.

32 T. Schwentick, D. Thérien, and H. Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *Developments in Language Theory (DLT)*, volume 2295 of *LNCS*, pages 239–250, 2001.

33 I. Simon. *Hierarchies of Events with Dot-Depth One.* PhD thesis, Department of Applied Analysis and Computer Science, University of Waterloo, Canada, 1972.

34 J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.

35 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Symposium on the Theory of Computing (STOC)*, pages 1–9. ACM, 1973.

36 W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.

37 A. N. Trahtman. Piecewise and local threshold testability of DFA. In *Fundamentals of Computation Theory (FCT)*, volume 2138 of *LNCS*, pages 347–358, 2001.

38 L. van Rooijen. *A combinatorial approach to the separation problem for regular languages.* PhD thesis, LaBRI, University of Bordeaux, France, 2014.

Stably Computing Order Statistics with Arithmetic Population Protocols*

George B. Mertzios[1], Sotiris E. Nikoletseas[2], Christoforos L. Raptopoulos[3], and Paul G. Spirakis[4]

1 **School of Engineering and Computing Sciences, Durham University, UK**
`george.mertzios@durham.ac.uk`

2 **Computer Engineering & Informatics Department, University of Patras, Patras, Greece; and**
Computer Technology Institute & Press "Diophantus", Patras, Greece
`nikole@cti.gr`

3 **Computer Engineering & Informatics Department, University of Patras, Patras, Greece; and**
Computer Technology Institute & Press "Diophantus", Patras, Greece
`raptopox@ceid.upatras.gr`

4 **Computer Engineering & Informatics Department, University of Patras, Patras, Greece; and**
Computer Technology Institute & Press "Diophantus", Patras, Greece; and
Department of Computer Science, University of Liverpool, Liverpool, UK
`p.spirakis@liverpool.ac.uk`

Abstract

In this paper we initiate the study of populations of agents with very limited capabilities that are globally able to compute order statistics of their arithmetic input values via pair-wise meetings. To this extent, we introduce the *Arithmetic Population Protocol (APP)* model, embarking from the well known Population Protocol (PP) model and inspired by two recent papers [1, 11] in which states are treated as integer numbers. In the APP model, every agent has a state from a set $\mathcal{Q}$ of states, as well as a fixed number of registers (independent of the size of the population), each of which can store an element from a *totally ordered* set $\mathcal{S}$ of samples. Whenever two agents interact with each other, they update their states and the values stored in their registers according to a *joint transition function*. This transition function is also restricted; it only allows (a) comparisons and (b) copy / paste operations for the sample values that are stored in the registers of the two interacting agents. Agents can only meet in pairs via a fair scheduler and are required to eventually converge to the same output value of the function that the protocol globally and stably computes. We present two different APPs for stably computing the median of the input values, initially stored on the agents of the population. Our first APP, in which every agent has 3 registers and no states, stably computes (with probability 1) the median under *any* fair scheduler in *any* strongly connected directed (or connected undirected) interaction graph. Under the probabilistic scheduler, we show that our protocol stably computes the median in $O(n^6)$ number of interactions in a connected undirected interaction graph of n agents. Our second APP, in which every agent has 2 registers and $O(n^2 \log n)$ states, computes to the correct median of the input with high probability in $O(n^3 \log n)$ interactions, assuming the probabilistic scheduler and the complete interaction graph. Finally we present a third APP which, for any k, stably computes the kth smallest element of the input of the population under *any* fair scheduler and in *any* strongly connected directed (or connected undirected) interaction graph. In this APP

* Sotiris E. Nikoletseas and Christoforos L. Raptopoulos were partially supported by the MULTIPLEX project – 317532. Paul G. Spirakis was partially supported by the MULTIPLEX project – 317532 and the EEE/CS School initiative NeST of the University of Liverpool.

every agent has 2 registers and n states. Upon convergence every agent has a different state; all these states provide a *total ordering* of the agents with respect to their input values.

1998 ACM Subject Classification I.2.11 Distributed Artificial Intelligence - Multiagent systems

Keywords and phrases arithmetic population protocols, order statistics, median, k-minimum element

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.68

1 Introduction

The *population protocol (PP)* model [2, 5] was originally defined to represent sensor networks consisting of very limited mobile agents with no control over their own movement. It has been defined by analogy to population processes [8] in probability theory and has already been used in various fields, such as in statistical physics, genetics, epidemiology, chemistry and biology [6]. An exciting feature of such systems of simple agents is that, by exchanging information through local pairwise interactions (rendezvous), the entities can collectively perform significant global computational tasks. In the basic model, in each pairwise interaction both participating agents update their state according to a (pre-specified) joint transition function that only depends on the states of the two agents. Furthermore the interactions between agents happen under some kind of a *fairness* condition. Assuming the complete interaction graph, it is known that population protocols compute exactly the class of semilinear predicates [3], i.e., the predicates definable in first-order logic Presburger arithmetic. In addition, it is well known that population protocols do not compose, except for certain cases; for a survey we refer to [5, 10].

One limitation of the PP model is that, as every agent is characterized at every time point by its state, the predicates that can be stably computed are defined over variables that count the number of nodes having initially some specific state from the set of allowable states (see for example [4, 9]). However, in several circumstances, it is natural to assume that nodes may also store some *arithmetic* values, for instance a temperature or some other local measurement. Imagine for instance a huge network of elementary sensors that measure local temperature and want to compute some statistical function of the temperatures of the whole network. The computation of elementary functions (e.g. the median) of such arithmetic values by a population of autonomous agents requires an extension of the standard PP model. On the other hand, it is worth noting that computations of statistics in classical distributed network models have been already considered, e.g. [7].

In order to overcome such limitations, we introduce in this paper the *Arithmetic Population Protocols (APP)* model, which is inspired by the PP model and by two recent papers [1, 11]. Similarly to the spirit of population protocols, we still assume in arithmetic population protocols that agents are weak computational devices with very small local memory, thus still staying on the pragmatic side. In [11], the authors consider the problem of determining the exact difference between the majority and the minority type in a two-type population. The basic idea of their protocol generalizes an idea of [1], where the set of states available to each agent was a set of integers, thus implying a total ordering of the states. In particular, an agent at state i was considered to be more firm in supporting her type than another agent at state i', with $|i| > |i'|$.

In our APP model, every agent has a state from a set $\mathcal{Q}$ of states, as well as $r \geq 1$ registers, each of which can store an element from a *totally ordered* set $\mathcal{S}$ of samples (e.g. $\mathcal{S} \subseteq \mathbb{N}$

or $\mathcal{S} \subseteq \mathbb{R}$). We assume that r is a fixed constant, i.e., independent of the population size. Furthermore, agents have limited knowledge of the set $\mathcal{S}$ and limited computational power over $\mathcal{S}$. In particular, no agent is aware of the whole set $\mathcal{S}$. Furthermore, whenever two agents interact with each other, they update their states and the values stored in their registers according to a *joint transition function*. This transition function only allows (a) comparisons and (b) copy / paste operations for values in $\mathcal{S}$ that are stored in the registers of the two interacting agents. Initially, each agent $v \in \mathcal{V}$ is given an element $x_v \in \mathcal{S}$ as input; her initial state and the initial values of her registers are then determined according to x_v. The goal is that eventually, after a sequence of local pairwise interactions (which are planned by a *scheduler* that satisfies a general "fairness" condition), every agent computes the same function on the input values in one of her registers. We assume that agents are oblivious of their own identity and of identities of other agents they interact with. Therefore, at any time, each agent is completely characterized by the values stored in its registers, together with its state.

1.1 Our model

More formally, let $\mathcal{S}$ denote a totally ordered set of *samples* and let $\mathcal{Q}$ be a totally ordered[1] set of states. We assume that there is a population $\mathcal{V}$ of computationally weak agents with limited memory. In particular, each agent can be in one of the states in $\mathcal{Q}$ and has r registers at its disposal, each one of which can store an element of $\mathcal{S}$.

For any $t \geq 0$ let $\mathbf{R}(t)$ be a $|\mathcal{V}| \times r$ matrix, such that $\mathbf{R}_{v,j}(t)$ is the value of the j-th register of agent $v \in \mathcal{V}$ at time t. Furthermore, for every $t \geq 0$, let $\mathbf{q}(t)$ be a $|\mathcal{V}|$-dimensional vector such that $\mathbf{q}_v(t)$ is the state of agent $v \in \mathcal{V}$ at time t. We refer to $\mathbf{C}(t) \overset{def}{=} (\mathbf{R}(t), \mathbf{q}(t))$ as the *configuration* at time t. We will assume that the number of registers r of each agent is a fixed constant independent of the population size, i.e., every agent has a limited number of registers available. For every agent $v \in \mathcal{V}$, we will refer to the first register of v as the *input register*; we also refer to $\mathbf{R}_{v,1}(t)$ as the *value of the input register* of v. Furthermore, we will say that $\mathbf{R}_{:,1}(0)$ (i.e., the first column of $\mathbf{R}(0)$) is the *population input* at time 0.

An *Arithmetic Population Protocol (APP)* is defined on a population $\mathcal{V}$ of agents and consists of an *input initialization function* $\iota : \mathcal{S} \to \mathcal{S}^r \times \mathcal{Q}$, an *output function* $\gamma : \mathcal{S}^r \times \mathcal{Q} \to \mathcal{D}$ ($\mathcal{D}$ is the set of output values; usually it is the same as the sample set $\mathcal{S}$), and a *joint transition function* $f : (\mathcal{S}^r \times \mathcal{Q}) \times (\mathcal{S}^r \times \mathcal{Q}) \to (\mathcal{S}^r \times \mathcal{Q}) \times (\mathcal{S}^r \times \mathcal{Q})$. In particular, if v interacts with u at time $t+1$, then the new values of their registers and states become $(\mathbf{R}_{v,:}(t+1), \mathbf{q}_v(t+1))$ and $(\mathbf{R}_{u,:}(t+1), \mathbf{q}_u(t+1))$ respectively, where $(\mathbf{R}_{v,:}(t+1), \mathbf{q}_v(t+1), \mathbf{R}_{u,:}(t+1), \mathbf{q}_u(t+1)) = f(\mathbf{R}_{v,:}(t), \mathbf{q}_v(t), \mathbf{R}_{u,:}(t), \mathbf{q}_u(t))$. As also mentioned earlier, the transition function f is *not arbitrary*: it only allows (a) comparisons and (b) copy / paste operations for values in $\mathcal{S}$ that are stored in the registers of the two interacting agents. Therefore, if agents v, u interact at time $t+1$, then for any register $j \in [r]$, we have $\mathbf{R}_{v,j}(t+1), \mathbf{R}_{u,j}(t+1) \in \{\mathbf{R}_{x,j'}(t) : x \in \{v,u\}, j' \in [r]\}$.

Initially, the values of the registers of each agent are determined by the input initialization function ι, i.e., for each agent $v \in \mathcal{V}$ that has input $x_v \in \mathcal{S}$, we initially set $(\mathbf{R}_{v,:}(0), \mathbf{q}_v(0)) = \iota(x_v)$, where $\mathbf{R}_{v,:}(0)$ denotes the row of $\mathbf{R}(0)$ corresponding to agent v. Subsequently, in every time step $t+1 \geq 1$, a pair of agents interacts and updates the values of their registers according to the transition function f.

[1] This assumption on $\mathcal{Q}$ is not necessary for our results, but is given for the sake of presentation.

Agent pairwise interactions are planned by a *scheduler* under a general "fairness" condition; the actual mechanism for choosing which agents interact each time is abstracted away. The fairness condition states that *the scheduler cannot postpone a possible finite sequence of agent interactions indefinitely.* The *direction of interaction* may or may not be relevant (see also the discussion below on the probabilistic scheduler); if it is not relevant, we say that the APP is *symmetric.*

Due to lack of coordination and storage restrictions, the agents in a population executing an APP cannot determine when the computation has finished. Instead, the values of the output registers of every agent is required to converge to a common (correct) value. More formally, we have the following:

▶ **Definition 1** (Stable computation)**.** Let $\mathcal{M}$ be the (infinite) set of multi-sets of a (finite or infinite) totally ordered set $\mathcal{S}$ and let $\mathcal{F} : \mathcal{M} \to \mathcal{D}$ be a function. We say that an APP *stably computes* the function $\mathcal{F}$ if and only if under *any fair scheduler*, for *any multi-set* $M \in \mathcal{M}$, and starting from *any configuration* where the elements in M are assigned bijectively to the input registers of the agents in a population $\mathcal{V}$ (where $|\mathcal{V}| = |M|$) we have that, after a finite time τ, $\gamma(\mathbf{R}_{v,:}(t), \mathbf{q}_v(t)) = \mathcal{F}(M)$, for all agents $v \in \mathcal{V}$ and for all $t \geq \tau$.

To allow for the comparison of our APPs in terms of the number of pairwise interactions needed to compute some function, we will consider in this paper a special case of a fair scheduler, namely the *probabilistic scheduler*, which is defined on directed interaction graphs as follows. In each time step a directed edge (v, u) of the interaction graph is chosen uniformly at random, where v (i.e., the tail of (v, u)) is called the *initiator* and u (i.e., the head of (v, u)) is called the *responder* of the interaction. Then, agents v and u update the values of their registers jointly according to the transition function f. The direction of interaction plays an important role in the general case, as the initiator will update its register values according to the first part of the outcome of f (consisting of r numeric values), while the responder will use the second part of the outcome (which also consists of r numeric values). The values of the registers of all other agents remain unchanged. The probabilistic scheduler is defined on undirected graphs similarly, by replacing every undirected edge $\{v, u\}$ by the two directed edges (v, u) and (u, v).

1.2 Our contribution

In this paper, we initiate the study of populations of agents with very limited capabilities that are globally able to compute order statistics of their input via pair-wise meetings. We initially focus on the fundamental problem of computing the *median* of the input values in a population of n agents, which is defined as the $\lceil n/2 \rceil$-th minimum element of the input values. We provide two different APPs for stably computing the median.

In our first APP (see Section 2) every agent has 3 registers and no states. This APP stably computes (with probability 1) the median under *any* fair scheduler in *any* strongly connected directed (or connected undirected) interaction graph. We also show that, under the probabilistic scheduler, our protocol stably computes the median in $O(n^6)$ number of interactions in a connected undirected interaction graph of n agents.

Our second APP for stably computing the median (see Section 3) is considerably faster than the first one, however it works with high probability (rather than with probability 1) and it requires additional assumptions on the scheduler and the structure of the underlying interaction graph. In particular, in our second APP for the median, every agent has 2 registers and $O(n^2 \log n)$ states. Assuming the probabilistic scheduler and the complete interaction graph, this APP converges to the correct median of the input of the population with high probability in $O(n^3 \log n)$ interactions. Additionally, agents are required to know the size of the population. The latter assumption can be dropped by assuming additional states and computational power for the agents.

As our final contribution, we present in Section 4 an APP which, for any k, stably computes the kth smallest element of the input of the population under *any* fair scheduler and in *any* strongly connected directed (or connected undirected) interaction graph. In this APP every agent has 2 registers and n states. Upon convergence every agent has a different state; all these states provide a *total ordering* of the agents with respect to their input values.

2 A 3-register APP for median

In this section we describe a APP using 3 registers and no states that stably computes the median of the input of a population of agents $\mathcal{V}$, i.e., the median of the (multi-) set of elements $\{x_v : v \in \mathcal{V}\} \in \mathcal{S}^n$. We will assume without loss of generality that the set of measurements $\mathcal{S}$ is the set of real numbers, i.e., $\mathcal{S} = \mathbb{R}$. Our protocol is not symmetric, hence in every interaction we distinguish between initiator and responder. For the sake of clarity of presentation, we will assume that the size $n = |\mathcal{V}|$ of the population is odd, so that the median is well (and uniquely) defined. In fact, without further modifications, our protocol may not converge if there is an even number of nodes (because in this case there are 2 candidates for the median).[2] Additionally, we will initially assume that the interaction graph is complete and we later show how this assumption can be dropped.

For every agent $v \in \mathcal{V}$, the first register will contain the (input) number x_v initially assigned to node v (i.e., $\mathbf{R}_{v,1}(0) = x_v$); the value of $\mathbf{R}_{v,1}(t)$ will remain unchanged throughout the computation. The third register of each agent v, i.e., $\mathbf{R}_{v,3}$, will eventually converge to median of the input of the population $\{x_v : v \in \mathcal{V}\}$. The first two registers (i.e., $\mathbf{R}_{v,1}, \mathbf{R}_{v,2}$) are used to make virtual connections between agents; upon convergence, the agent with the smallest number will be virtually connected to the agent with the largest number, the agent with the second smallest number will be virtually connected to the agent with the second largest number and so on. Furthermore, for any agent $v \in \mathcal{V}$, we will denote by $I_v(t) \overset{def}{=} \{s \in \mathbb{R} : \min(\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t)) \leq s \leq \max(\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t))\}$. For simplicity, we will write $I_v(t) = [\min(\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t)), \max(\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t))]$ and we will refer to $I_v(t)$ as the *closed interval* of v at time t (we use this term loosely, since in general the set of measurements may not be compact). Finally, we denote $\mathbf{C}_v(t) \overset{def}{=} (\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t), \mathbf{R}_{v,3}(t))$.

The initialization function of our protocol is given by $\iota_{\text{MDN}}(x_v) = \mathbf{C}_v(0) = [x_v, x_v, x_v]$, for any $x_v \in \mathcal{S}$ and $v \in \mathcal{V}$, and the output function is given by $\gamma_{\text{MDN}}(\mathbf{C}_v(t)) = \mathbf{R}_{v,3}(t)$, for any $v \in \mathcal{V}$. Consequently, for any $v \in \mathcal{V}$, the closed interval $I_v(0)$ contains just one point. The joint transition function is defined as follows: if agent v (the initiator) interacts with u (the responder) at time $t + 1 = 1, 2, \ldots$, then $(\mathbf{C}_v(t+1), \mathbf{C}_u(t+1)) = f_{\text{MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t))$, where f_{MDN} is given below.

[2] Nevertheless, it is not difficult to extend our APP so that it converges to one of the 2 candidate values (say the smallest) for the median in the case where n is even.

```
Transition function $f_{\mathrm{MDN}}$
Input: $\mathbf{C}_v(t), \mathbf{C}_u(t)$
```

Case I: $\mathbf{R}_{v,1}(t) = \mathbf{R}_{v,2}(t)$ AND $I_v(t) \subseteq I_u(t)$.

$$f_{\mathrm{MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = ([\mathbf{C}_v(t)], [x_u, \mathbf{R}_{u,2}(t), x_v])$$

Case II: $I_v(t) \nsubseteq I_u(t)$ AND $I_u(t) \nsubseteq I_v(t)$ AND $x_v \in \{r_2, r_3\}$ AND $x_u \in \{r_1, r_4\}$, where r_1, r_2, r_3 and r_4 are the first, second, third and fourth smallest value in the set $\{\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t), \mathbf{R}_{u,1}(t), \mathbf{R}_{u,2}(t)\}$ respectively.

$$f_{\mathrm{MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = ([x_v, \{r_2, r_3\} \backslash x_v, \mathbf{R}_{v,3}(t)], [x_u, \{r_1, r_4\} \backslash x_u, \mathbf{R}_{u,3}(t)])$$

Case III: Every other case.

$$f_{\mathrm{MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = (\mathbf{C}_v(t), \mathbf{C}_u(t))$$

We can prove the following:

▶ **Theorem 2** (Correctness). *The APP with initialization function ι_{MDN}, output function γ_{MDN} and transition function f_{MDN} stably computes the median function when the underlying interaction graph is the complete graph.*

Proof. Let $n = |\mathcal{V}|$ and define the potential function $\phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ as follows:

$$\phi(\mathbf{R}_{:,1}(t), \mathbf{R}_{:,2}(t)) = \sum_{v \in \mathcal{V}} (\mathbf{R}_{v,2}(t) - \mathbf{R}_{v,1}(t))^2.$$

Clearly, ϕ remains unchanged if two agents interact according to one of the cases I, or III, since the values of the first two registers of the agents remain unchanged. On the other hand, ϕ strictly increases in case II. Indeed, suppose that agent v (the initiator) interacts with agent u (the responder) at time $t+1$ according to case II, and let r_1, r_2, r_3 and r_4 be the first, second, third and fourth smallest value in the set $\{\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t), \mathbf{R}_{u,1}(t), \mathbf{R}_{u,2}(t)\}$. Then

$$\begin{aligned} &\phi(\mathbf{R}_{:,1}(t+1), \mathbf{R}_{:,2}(t+1)) - \phi(\mathbf{R}_{:,1}(t), \mathbf{R}_{:,2}(t)) \\ &\quad = \left[(r_3 - r_2)^2 + (r_4 - r_1)^2\right] - \left[(\mathbf{R}_{v,2}(t) - \mathbf{R}_{v,1}(t))^2 + (\mathbf{R}_{u,2}(t) - \mathbf{R}_{u,1}(t))^2\right] \\ &\quad = 2\left(\mathbf{R}_{v,1}(t)\mathbf{R}_{v,2}(t) + \mathbf{R}_{u,1}(t)\mathbf{R}_{u,2}(t) - r_2 r_3 - r_1 r_4\right) > 0. \end{aligned} \tag{1}$$

Notice also that, by the definition of ι_{MDN} and f_{MDN}, our protocol maintains the following invariant:

(INV) For any $v \in \mathcal{V}$ and any time $t \geq 0$, there are exactly 2 copies of x_v, one in $\mathbf{R}_{:,1}(t)$ and one in $\mathbf{R}_{:,2}(t)$ (obviously, if two or more nodes have the same input, there will be two copies for each).

Let $0 \leq i \leq \lceil \frac{n}{2} \rceil$. We say that a configuration $\mathbf{C}(t)$ is *non-crossing up to i* if, for every $1 \leq k \leq i$, the agent with the kth smallest input value is matched in an interval with the kth largest input value. That is, a configuration $\mathbf{C}(t)$ is non-crossing up to i if there are sets of distinct agents $L^{(i)}$ and $U^{(i)}$ for which the following conditions hold:

(C1) $L^{(i)} = \{z_1, \ldots, z_i\} \subseteq \mathcal{V}$, such that $x_{z_1} \leq \cdots \leq x_{z_i}$ and, for each $z \in L^{(i)}$ and $z' \in \mathcal{V} \backslash L^{(i)}$, we have $x_z \leq x_{z'}$.

(C2) $U^{(i)} = \{u_1, \ldots, u_i\} \subseteq \mathcal{V}$, such that $x_{u_1} \geq \cdots \geq x_{u_i}$ and, for each $u \in U^{(i)}$ and $u' \in \mathcal{V} \backslash U^{(i)}$, we have $x_u \geq x_{u'}$.

(C3) $I_{z_j}(t) = I_{u_j}(t) = [z_j, u_j]$, for all $j \in [i]$.

The sets $L^{(i)}, U^{(i)}$ will be called *witnesses* for the fact that $\mathbf{C}(t)$ is non-crossing up to i. We can prove the following:

▶ **Claim 3.** *Let $i < \lceil \frac{n}{2} \rceil$. If $\mathbf{C}(t)$ is non-crossing up to i but not up to $i+1$, then there is a pair of agents that can interact according to Case II. Equivalently, there is an interaction of agents that can strictly increase the value of ϕ.*

Proof. Let i be the maximum index such that $\mathbf{C}(t)$ is non-crossing up to i and let $L^{(i)}, U^{(i)}$ be witnesses of this fact. Let $L^+ \overset{def}{=} \{z : z \notin L^{(i)}, x_z \leq x_{z'}, \text{ for every } z' \in \mathcal{V} \backslash L^{(i)}\}$ and $U^- \overset{def}{=} \{u : u \notin U^{(i)}, x_u \geq x_{u'}, \text{ for every } u' \in \mathcal{V} \backslash U^{(i)}\}$. Notice that, by definition, x_z (respectively x_u) is the same for all $z \in L^+$ (respectively $u \in U^-$).

The fact that $\mathbf{C}(t)$ is non-crossing up to i (and not up to $i+1$) implies that at least one of the following holds:

(i) For all agents $z \in L^+$, there is no agent $u \in U^-$, such that $I_z(t) = [x_z, x_u]$.
(ii) For all agents $u \in U^-$, there is no agent $z \in L^+$, such that $I_u(t) = [x_z, x_u]$.

Assume without loss of generality that (i) holds; the case where (ii) holds is similar, by symmetry. Let $z_{\max} \in L^+$ be such that the length of the interval $I_{z_{\max}}(t)$ is maximal. In particular, this means that there is some agent w such that $I_{z_{\max}}(t) = [x_{z_{\max}}, x_w]$, with $x_{z_{\max}} \leq x_w < x_u$, for each $u \in U^-$. Indeed, by assumption we have that $x_w \neq x_u$, for each $u \in U^-$. But also, we cannot have $x_w > x_u$ or $x_w < x_{z_{\max}}$, by definition of the sets L^+, U^-.

Notice now that, by invariant (INV) there also exists a pair of agents $z', w' \notin L^{(i)} \cup U^{(i)}$, such that $I_{z'}(t) = [x_{z'}, x_{w'}]$, with $x_{w'} > x_w$ (and $x_{z'} \leq x_{w'}$). Indeed, let $I_w(t)$ be the interval of node w at time t. If $\max(\mathbf{R}_{w,1}(t), \mathbf{R}_{w,2}(t)) > x_w$, then we have $z' = w$. Otherwise, let x_y be the other endpoint of $I_w(t)$ (in fact $x_y = \min(\mathbf{R}_{w,1}(t), \mathbf{R}_{w,2}(t))$. Similarly, if $\max(\mathbf{R}_{y,1}(t), \mathbf{R}_{y,2}(t)) > x_w$, then we have $z' = y$. Proceeding inductively, we will eventually find an agent z' with $\max(\mathbf{R}_{z',1}(t), \mathbf{R}_{z',2}(t)) > x_w$. Notice also that z' does not belong to U^-, by construction, but also neither to L^+, by maximality of $I_{z_{max}}(t)$.

We can then see that the pair of agents $(z', z_{\max})$ is suitable for interaction according to Case II. Indeed, $\mathbf{R}_{z',1}(t) = x_{z'} > x_{z_{\max}} = \mathbf{R}_{z_{\max},1}(t)$, so $I_{z_{\max}}(t) \nsubseteq I_{z'}(t)$, $\mathbf{R}_{z',2}(t) = x_{w'} > x_w = \mathbf{R}_{z_{\max},2}(t)$, so $I_{z'}(t) \nsubseteq I_{z_{\max}}(t)$ and finally $x_{z'} \in \{r_2, r_3\} = \{x_{z'}, x_w\}$ and $x_{z_{max}} \in \{r_1, r_4\} = \{x_{z_{max}}, x_{w'}\}$. This completes the proof of the claim, since by equation (1), any interaction according to Case II strictly increases the value of ϕ. ◀

By the above claim, we can now show that the potential function ϕ is maximized whenever the system reaches a configuration $\mathbf{C}(t)$ that is non-crossing up to $\lceil \frac{n}{2} \rceil$. Indeed, by inequality (1), any interaction according to case II strictly increases the value of the potential ϕ, and since this increment is independent of t and the maximum value of ϕ is finite (e.g. it is at most $n(\max\{x_v\} - \min\{x_v\})^2$), we conclude that, in finite time (because of the fairness assumption of the scheduler) we will have reached the desired configuration. Finally, notice that in such a configuration, the node that corresponds to the median will have the same value stored in both its first two registers (by the assumption that n is odd). In particular, whenever it interacts as an initiator with another agent, it will do so according to case I. In fact, no agent with assigned value different than the median will be able to interact with other nodes according to this case after convergence of the protocol. This implies that, eventually, all agents will have the value of the correct median stored in their third register. This completes the proof. ◀

Notice that our protocol does not make any assumption on the uniqueness of assigned values to agents. For example, if there are several agents that have $\mathbf{R}_{v,1}(t) = \mathbf{R}_{v,2}(t)$ and ϕ is maximum, then this means that there are at least 3 nodes that are equal to the median; our protocol will still stably compute the correct median value. It is only required that nodes have registers that can store any element in the multi-set $\{x_v : x \in \mathcal{V}\}$.

Before moving on to the running time analysis of our protocol under the probabilistic scheduler, it is worth noting that we can slightly modify our transition function f_{MDN} so that the protocol works on arbitrary strongly connected directed interaction graphs. Indeed, we just need to guarantee that every two agents will eventually be able to compare their input. This can be achieved if agents that interact also exchange the values of their registers. Therefore, we have the following more general result:

▶ **Theorem 4.** *Let $f'_{MDN} : \mathcal{S}^3 \times \mathcal{S}^3 \to \mathcal{S}^3 \times \mathcal{S}^3$ be a transition function defined as follows: for any $\vec{x}, \vec{y}, \vec{x}', \vec{y}' \in \mathcal{S}^3$, $f'_{MDN}(\vec{x}, \vec{y}) = (\vec{y}', \vec{x}')$ if and only if $f_{MDN}(\vec{x}, \vec{y}) = (\vec{x}', \vec{y}')$. Then the APP with input initialization function ι_{MDN}, output function γ_{MDN} and transition function f'_{MDN} stably computes the median function under any strongly connected directed interaction graph.*

We note that the potential function ϕ defined for the proof of correctness of our protocol, together with inequality (1) can be used to give upper bounds on the expected time needed for the protocol to stably compute the median under the probabilistic scheduler. However, this bound will depend on the input of the population. Nevertheless, by using Claim 3 and the definition of non-crossing configurations from the proof of Theorem 2, we can also provide an upper bound that only depends on the population size.

▶ **Theorem 5.** *Assuming the probabilistic scheduler under any connected undirected interaction graph, the expected time needed for the APP with input initialization function ι_{MDN}, output function γ_{MDN} and transition function f'_{MDN} to stably compute the median of the input of a population $\mathcal{V}$ of n agents is $O(n^6)$.*

Proof. Let T_1 be the time until the population reaches a configuration that is non-crossing up to $\lceil \frac{n}{2} \rceil$ (see the definition in the proof of Theorem 2) and let T_2 be the additional time needed for the value of the median to be propagated to all agents in the population. Clearly, the expected time needed for the protocol to stably compute the correct median value is $\mathbb{E}[T_1 + T_2]$.

Notice that, by definition of f'_{MDN}, whenever two agents interact, they also exchange the values of their registers. Therefore, it is as if each agent is performing a random walk on the interaction graph $G = (V, E)$. Furthermore, these random walks performed by two agents are independent until those 2 agents interact with each other.

Remember that, (arguing as in the proof of Theorem 2) if at some time t the configuration reached is non-crossing up to i, but not up to $i + 1$, we have that at least one of the following holds:

(i) For all agents $z \in L^+$, there is no agent $u \in U^-$, such that $I_z(t) = [x_z, x_u]$.
(ii) For all agents $u \in U^-$, there is no agent $z \in L^+$, such that $I_u(t) = [x_z, x_u]$.

In particular, by definition of f_{MDN}, agents $z \in L^+ \cup U^-$ can never decrease the length of the interval $I_z(t')$, for any $t' \geq t$. But then, the proof of Claim 3 implies that there is a pair of agents z, z', with $z \in L^+ \cup U^-$, who can interact according to Case II, which will strictly increase the length of $I_z(t')$. By Corollary 1 of [12], the expectation of the maximum meeting time is $O(n^3)$. If we also take into account the interactions which do not involve agents z, z', we then have that the expected number of steps before some agent $z \in L^+ \cup U^-$ increases the length of its interval is at most $O(n^4)$, where we also used the fact that the expected number of steps between two interactions that involve z, z' is $O(n)$. Since the length of the interval of any $z \in L^+ \cup U^-$ can increase at most n times and there are at most n agents, we have that $\mathbb{E}[T_1] = O(n^6)$.

To bound $\mathbb{E}[T_2]$, we can use Theorem 3 of [12], which states that the expected number of steps needed for a single random walk to cover all vertices of a graph is $O(n^4)$. If we also take into account the interactions which do not involve the median (there are $O(n)$ such steps in expectation between any interaction involving the median), we have that $\mathbb{E}[T_2] = O(n^5)$, which completes the proof. ◀

3 A faster protocol for median using random walks

In this section we describe another APP for stably computing the median of the input of a population of agents $\mathcal{V}$. Our protocol is not symmetric (hence in every interaction we distinguish between initiator and responder) and it converges with high probability (rather than with probability 1) to the correct median faster than the one in Section 2, at the expense of additional assumptions on the scheduler and the structure of the underlying interaction graph; in particular, we assume the probabilistic scheduler, under the complete interaction graph. For the sake of clarity of presentation, we will assume that the size $n = |\mathcal{V}|$ of the population is known to all agents. However, it is worth noting that this assumption can be dropped if one combines the APP presented here with the APP described in the Remark at the end of Section 4 for stably computing the population size, provided that agents are able to locally compute the value of a certain function $p(n)$ (for any n) that will be defined later.

Our APP uses 2 registers and $3p(n) + 2$ states per agent; the value of $p(n)$ will be determined in Theorem 6 below. For every agent $v \in \mathcal{V}$, the value of the input register $\mathbf{R}_{v,1}(t)$ will be initialized to x_v and will remain unchanged throughout the computation. The second register $\mathbf{R}_{v,2}(t)$ will eventually contain the median, i.e., with high probability, $\mathbf{R}_{v,2}(t)$ will eventually converge to the median of the set $\{x_v : v \in \mathcal{V}\}$. We view the state space $\mathcal{Q}$ for each agent $v \in \mathcal{V}$ as two counters $\mathbf{q}_{v,1}(t)$ and $\mathbf{q}_{v,2}(t)$; the first one can store an integer between $-p(n)$ and $p(n)$, and the second an integer between 0 and $p(n)$. These counters are used as follows: whenever agent v (the initiator) interacts with another agent u (the responder) at time $t + 1$, it will increase (respectively decrease) the value of $\mathbf{q}_{v,1}(t)$ by 1 if $x_v \geq x_u$ (respectively if $x_v < x_u$). Therefore, for each agent v, $\mathbf{q}_{v,1}(t)$ describes a (possibly) biased random walk, the least biased (or ideally unbiased) of which will correspond to the median. This means that, with high probability, if we stop those random walks after a sufficient number of steps, the one closest to 0 will correspond to the median of the population. The total number of steps that the random walk for agent v takes are counted in $\mathbf{q}_{v,2}(t)$.

More precisely, let $p(n)$ be any large enough integer function of the size of the population, which will stand for the maximum number of steps that we allow each random walk to take. We denote $\mathbf{C}_v(t) \overset{def}{=} \left(\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t), \mathbf{q}_{v,1}(t), \mathbf{q}_{v,2}(t)\right)$. Also, for any agent v, we denote by $\mathbf{1}_v$ the indicator variable that is equal to 1 if $\mathbf{q}_{v,2}(t) < p(n)$ and 0 otherwise. The initialization function of our protocol is given by $\iota_{\text{RW-MDN}}(x_v) = \mathbf{C}_v(0) = [x_v, x_v, 0, 0]$, for any $x_v \in \mathcal{S}$ and $v \in \mathcal{V}$, and the output function is given by $\gamma_{\text{RW-MDN}}(\mathbf{C}_v(t)) = \mathbf{R}_{v,2}(t)$, for any $v \in \mathcal{V}$. The transition function is defined as follows: if agent v (the initiator) interacts with agent u (the responder) at time $t + 1 = 1, 2, \ldots$, then $(\mathbf{C}_v(t + 1), \mathbf{C}_u(t + 1)) = f_{\text{RW-MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t))$, where $f_{\text{RW-MDN}}$ is given below.

```
Transition function $f_{\text{RW-MDN}}$
Input: $\mathbf{C}_v(t), \mathbf{C}_u(t)$
Case I: $x_v \geq x_u$ AND $\mathbf{q}_{v,2}(t) < p(n)$.
    $f_{\text{RW-MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) =$
      $([x_v, \mathbf{R}_{v,2}(t), \mathbf{q}_{v,1}(t) + \mathbf{1}_v, \mathbf{q}_{v,2}(t) + \mathbf{1}_v], [x_u, \mathbf{R}_{u,2}(t), \mathbf{q}_{u,1}(t) - \mathbf{1}_u, \mathbf{q}_{u,2}(t) + \mathbf{1}_u])$
Case II: $x_v < x_u$ AND $\mathbf{q}_{v,2}(t) < p(n)$.
    $f_{\text{RW-MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) =$
      $([x_v, \mathbf{R}_{v,2}(t), \mathbf{q}_{v,1}(t) - \mathbf{1}_v, \mathbf{q}_{v,2}(t) + \mathbf{1}_v], [x_u, \mathbf{R}_{u,2}(t), \mathbf{q}_{u,1}(t) + \mathbf{1}_u, \mathbf{q}_{u,2}(t) + \mathbf{1}_u])$
Case III: $\mathbf{q}_{v,2}(t) = \mathbf{q}_{u,2}(t) = p(n)$ AND $|\mathbf{q}_{v,1}(t)| \leq |\mathbf{q}_{u,1}(t)|$.
    $f_{\text{RW-MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = (\mathbf{C}_v(t), [x_u, \mathbf{R}_{v,2}(t), \mathbf{q}_{u,1}(t), \mathbf{q}_{u,2}(t)])$
Case IV: Every other case.
    $f_{\text{RW-MDN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = (\mathbf{C}_v(t), \mathbf{C}_u(t))$
```

We can prove the following:

▶ **Theorem 6.** *If $p(n) \geq 16n^2 \ln n$, then the APP with input initialization function $\iota_{RW\text{-}MDN}$, output function $\gamma_{RW\text{-}MDN}(v)$ and transition function $f_{RW\text{-}MDN}$ stably computes the median of the input of a population of n agents with high probability (i.e., with probability tending to 0 as n goes to ∞), under the complete interaction graph, assuming the probabilistic scheduler.*

Proof. Consider an ordering $v_1, \ldots, v_n$ of the agents of the population $\mathcal{V}$ in non-decreasing order of their input, i.e., $x_{v_1} \leq x_{v_2} \leq \cdots \leq x_{v_n}$. For any agent v, we will denote by m_v the number of agents that have the same input as v (including v herself), i.e., $m_v \stackrel{def}{=} |\{u \in \mathcal{V} : x_u = x_v\}|$.

For any agent v, and $\ell = 1, 2, \ldots, p(n)$, define the random variable $X_\ell^{(v)}$ as follows: $X_\ell^{(v)} = 1$ if at the ℓ-th interaction of agent v (either as an initiator or responder) the value of $\mathbf{q}_{v,1}(t)$ increases and $X_\ell^{(v)} = -1$ otherwise. In particular, for any $\ell = 1, 2, \ldots, p(n)$, at the ℓ-th interaction of agent v, the following disjoint events may happen: (a) either v will interact with an agent u that has $x_v > x_u$, in which case $X_\ell^{(v)} = 1$, or (b) v will interact with an agent u that has $x_v < x_u$, in which case $X_\ell^{(v)} = -1$, or (c) v will interact with another agent u that has $x_v = x_u$, in which case if v is the initiator (which happens with probability $\frac{1}{2}$, by definition of the probabilistic scheduler) then $X_\ell^{(v)} = 1$, otherwise (i.e., if v is the responder of the interaction) $X_\ell^{(v)} = -1$.

Therefore, for any fixed agent $v \in \mathcal{V}$, the $p(n)$ random variables $X_\ell^{(v)}$, $1 \leq \ell \leq p(n)$ are independent and also

$$\Pr\left(X_\ell^{(v)} = 1\right) = 1 - \Pr\left(X_\ell^{(v)} = -1\right) = \frac{|\{u : x_v > x_u\}|}{n-1} + \frac{1}{2}\frac{(m_v - 1)}{n-1}.$$

For each agent $v_i, i \in [n]$, define now the discrete time stochastic process $\{Y_\ell^{(v_i)}\}_{\ell \geq 0}$ as follows:

(i) $Y_0^{(v_i)} = 0$

(ii) $Y_\ell^{(v_i)} = Y_{\ell-1}^{(v_i)} + X_\ell^{(v_i)} - \mathbb{E}[X_\ell^{(v_i)}] = \sum_{i=1}^{\ell} X_\ell^{(v_i)} - \ell \frac{n-2|\{u : x_{v_i} > x_u\}| - m_{v_i}}{n-1}$, for any $1 \leq \ell \leq p(n)$

(iii) $Y_\ell^{(v_i)} = Y_{\ell-1}^{(v_i)}$, for any $\ell > p(n)$

Since $X_\ell^{(v_i)}$ are independent, for every $1 \leq \ell \leq p(n)$, it is easy to prove that $\{Y_\ell^{(v_i)}\}_{\ell \geq 0}$ is a Martingale, that also satisfies $|Y_\ell^{(v_i)} - Y_{\ell-1}^{(v_i)}| \leq 2$, for all $\ell \geq 1$. Therefore, by Azuma's inequality, for any $x \geq 0$,

$$\Pr\left(\left|Y^{(v_i)}_{p(n)} - Y^{(v_i)}_0\right| \geq x\right) = \Pr\left(\left|Y^{(v_i)}_{p(n)}\right| \geq x\right) \leq 2e^{\frac{-x^2}{8p(n)}}.$$

Let now $t^*_{v_i}$ the time just after the $p(n)$-th interaction of v_i, i.e., the values of the counters $\mathbf{q}_{v_i,1}, \mathbf{q}_{v_i,2}$ remain unchanged after $t^*_{v_i}$. By the definition of $X^{(v_i)}_\ell, 1 \leq \ell \leq p(n)$, we then have that $\mathbf{q}_{v_i,1}(t^*_{v_i}) = \sum_{\ell=1}^{p(n)} X^{(v_i)}_\ell$. Furthermore, by the previous inequality and by definition of $\{Y^{(v_i)}_\ell\}_{\ell \geq 0}$, we have that

$$\Pr\left(\left|\mathbf{q}_{v_i,1}(t^*_{v_i}) - p(n)\frac{n - 2|\{u : x_{v_i} > x_u\}| - m_{v_i}}{n-1}\right| \geq x\right) = \Pr\left(\left|Y^{(v_i)}_{p(n)}\right| \geq x\right) \leq 2e^{\frac{-x^2}{8p(n)}}. \quad (2)$$

For any agent $v \in \mathcal{V}$, let $A_v = p(n)\frac{n-2|\{u:x_v>x_u\}|-m_v}{n-1}$. Notice now that, for any two agents v, v', such that $x_v \neq x'_v$, we have that $|(|\{u : x_v > x_u\}| - |\{u : x_{v'} > x_u\}|)| \geq 1$, since for one of v, v', the number of agents with strictly greater input must be larger. Similarly, $|(|\{u : x_v > x_u\}| + m_v) - (|\{u : x_{v'} > x_u\}| + m_{v'})| = |(|\{u : x_v \geq x_u\}| - |\{u : x_{v'} \geq x_u\}|)| \geq 1$. Therefore, we have proved the following separation inequality:

$$|A_v - A_{v'}| \geq 2\frac{p(n)}{n-1}, \quad \text{for every } v, v' \text{ with } x_v \neq x'_v.$$

Now set $x = \frac{p(n)}{n}$ in (2). Since $p(n) \geq 16n^2 \ln n$, we have that, for any $v_i, i \in [n]$,

$$\Pr\left(\left|\mathbf{q}_{v_i,1}(t^*_{v_i}) - A_{v_i}\right| \geq \frac{p(n)}{n}\right) \leq \frac{2}{n^2}$$

By the union bound, we conclude that, with probability at least $1 - \frac{2}{n}$, not only are the values $\mathbf{q}_{v_i,1}(t^*_{v_i})$, for all nodes v_i, concentrated around their mean values A_{v_i}, but they are also well separated, i.e., just by looking at $\mathbf{q}_{v_i,1}(t^*_{v_i})$, we can uniquely determine the value of A_{v_i}. Furthermore, node v_i will never change the value of its first register after time $t^*_{v_i}$.

Finally note that the smallest value of $|A_{v_i}|$ (hence also the smallest value of $|\mathbf{q}_{v_i,1}(t^*_{v_i})|$ whp, because of well separation) will correspond to agent $v_{\lceil n/2 \rceil}$, i.e., the agent with the median of $\{x_v, v \in \mathcal{V}\}$ as input. Indeed, it is not hard to see that, by definition, $|A_{v_i}|$ is a function of i, for which $|A_{v_i}| \leq |A_{v_{i-1}}|$ for every $i \leq \lceil \frac{n}{2} \rceil$, and $|A_{v_{i+1}}| \geq |A_{v_i}|$ for every $i \geq \lceil \frac{n}{2} \rceil$. Therefore, with high probability, the correct median will be propagated because of Case III of the protocol. This completes the proof. ◀

4 k-th minimum element

In this section, we present an APP with n states and 2 registers per agent (the first of which serves as the input register and does not change throughout the computation) that stably computes the k-th minimum element of the input values $x_v, v \in \mathcal{V}$. Our protocol is not symmetric, hence in every interaction we distinguish between initiator and responder. For simplicity, we first present a protocol that stably computes the k-minimum function assuming the complete interaction graph and then we generalize it for arbitrary interaction graphs.

The set of states of our protocol is $\mathcal{Q} = \{0, 1, \ldots, n-1\}$; without loss of generality, we treat the state of each agent v at time t as a counter $\mathbf{q}_v(t)$ which is initialized to the value 0 and can count up to $n-1$. For simplicity, we will denote $\mathbf{C}_v(t) \overset{def}{=} (\mathbf{R}_{v,1}(t), \mathbf{R}_{v,2}(t), \mathbf{q}_v(t))$.

The input initialization function of our protocol is given by $\iota_{k\text{-MIN}}(x_v) = \mathbf{C}_v(0) = (x_v, x_v, 0)$, for any $x_v \in \mathcal{S}$ and $v \in \mathcal{V}$, and the output function is given by $\gamma_{k\text{-MIN}}(\mathbf{C}_v(t)) =$

$\mathbf{R}_{v,2}(t)$, for any $v \in \mathcal{V}$. The transition function is defined as follows: if agent v (the initiator) interacts with agent u (the responder) at time $t+1 = 1, 2, \ldots$, then $(\mathbf{C}_v(t+1), \mathbf{C}_u(t+1)) = f_{k\text{-MIN}}(\mathbf{C}_v(t), \mathbf{C}_u(t))$, where $f_{k\text{-MIN}}$ is given below.

```
Transition function $f_{k\text{-MIN}}$
Input: $\mathbf{C}_v(t), \mathbf{C}_u(t)$
Case I: $\mathbf{q}_v(t) = \mathbf{q}_u(t)$ AND $\mathbf{R}_{v,1}(t) \geq \mathbf{R}_{u,1}(t)$.
    $f_{k\text{-MIN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = ([x_v, \mathbf{R}_{v,2}(t), \mathbf{q}_v(t)+1], [\mathbf{C}_u(t)])$
Case II: $\mathbf{q}_v(t) = \mathbf{q}_u(t)$ AND $\mathbf{R}_{v,1}(t) < \mathbf{R}_{u,1}(t)$.
    $f_{k\text{-MIN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = ([\mathbf{C}_v(t)], [x_u, \mathbf{R}_{u,2}(t), \mathbf{q}_u(t)+1])$
Case III: $\mathbf{q}_v(t) \neq \mathbf{q}_u(t)$ AND $\mathbf{q}_v(t) = k-1$.
    $f_{k\text{-MIN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = ([x_v, x_v, \mathbf{q}_v(t)], [x_u, x_v, \mathbf{q}_u(t)])$
Case IV: Every other case.
    $f_{k\text{-MIN}}(\mathbf{C}_v(t), \mathbf{C}_u(t)) = (\mathbf{C}_v(t), \mathbf{C}_u(t))$
```

▶ **Theorem 7.** *The APP with initialization function $\iota_{k\text{-MIN}}$, output function $\gamma_{k\text{-MIN}}$ and transition function $f_{k\text{-MIN}}$ stably computes the k-minimum of the set $\{x_v : v \in \mathcal{V}\}$.*

Proof. We first prove that, eventually, there will be exactly one agent in state i, for each $0 \leq i \leq n-1$. To this end, define t_i to be the earliest time when exactly one agent has state j, for each $j \leq i$. Note that, after time t_i, every agent v_j, $1 \leq j \leq i+1$, which has $\mathbf{q}_{v_j}(t_i) = j-1$, will never change her state. Indeed, by definition of $f_{k\text{-MIN}}$, for every $v \in \mathcal{V}$, $\mathbf{q}_v(t)$ never decreases; it can only increase if there is another agent $u \neq v$ with $\mathbf{q}_u(t) = \mathbf{q}_v(t)$. Thus we only need to prove that t_{n-1} is finite; we do this by induction on i.

For the base case of our inductive argument we need to prove that t_0 is finite. This is true because, if at some time t there are at least 2 agents at state 0, then, since the scheduler is fair, after finite time, two of these agents will eventually interact, which, by Cases I and II of $f_{k\text{-MIN}}$, will result in one of the agents increasing her state by 1. This will continue until there is only one agent in state 0.

For the inductive step, suppose that t_i is finite, for some i. By definition, at time t_i we have exactly one agent in state j, for each $j = 0, \ldots, i$. However, by definition of $f_{k\text{-MIN}}$, we also have that at time t_i some agent v set $\mathbf{q}_v(t_i) = i+1$ (i.e., either the one with the largest value stored in its first register, or the initiator in the interaction if both agents had the same value stored in their first registers). If this agent v was the only agent in state $i+1$ after t_i, then we also have that $t_i = t_{i+1}$. If not, then, similarly to the base case, after finite time the number of agents in state $i+1$ will decrease by 1, and this will continue until (within finite time) only one agent remains in state $i+1$. This completes the induction step, implying that t_i is finite, for every $i = 0, 1, \ldots, n-1$. In particular, in finite time t_{n-1}, there will be exactly one agent in state i, for each $i = 0, \ldots, n-1$ and these values will never change thereafter. Moreover, after time t_{n-1}, the agent in state $k-1$ will have the k-minimum of the set $\{x_v : v \in \mathcal{V}\}$ stored in its first register and this item will eventually appear in the second register of each agent in the population (because of Case III of $f_{k\text{-MIN}}$). ◀

We now slightly modify our transition function $f_{k\text{-MIN}}$ so that the protocol works on arbitrary strongly connected directed interaction graphs. Notice that we just need to guarantee that every two agents will eventually be able to exchange their local information.

This can be achieved if agents that interact also swap their register values and states. Therefore, we have the following more general result:

▶ **Theorem 8.** *Let $f'_{k\text{-}MIN} : (\mathcal{S}^2 \times \mathcal{Q}) \times (\mathcal{S}^2 \times \mathcal{Q}) \to (\mathcal{S}^2 \times \mathcal{Q}) \times (\mathcal{S}^2 \times \mathcal{Q})$ be a transition function defined as follows: for any $\vec{x}, \vec{y}, \vec{x}', \vec{y}' \in \mathcal{S}^2 \times \mathcal{Q}$, $f'_{k\text{-}MIN}(\vec{x}, \vec{y}) = (\vec{y}', \vec{x}')$ if and only if $f_{k\text{-}MIN}(\vec{x}, \vec{y}) = (\vec{x}', \vec{y}')$. Then the APP with initialization function $\iota_{k\text{-}MIN}$, output function $\gamma_{k\text{-}MIN}$ and transition function $f'_{k\text{-}MIN}$ stably computes the k-minimum of the set $\{x_v : v \in \mathcal{V}\}$ under any strongly connected interaction graph.*

Note that, upon convergence in this APP, each of the n agents has a different state which is an integer between 0 and $n-1$. Therefore all these states provide a *total ordering* of the agents with respect to their input values.

▶ Remark. This APP can be slightly modified to stably compute the population size $n = |\mathcal{V}|$, under any strongly connected interaction graph. Indeed, we only need that the second register of each agent $v \in \mathcal{V}$ is replaced by another counter $\mathbf{q}'_v(t)$, which stores the value n (instead of the k-minimum element). This can be achieved trivially by initializing $\mathbf{q}'_v(0)$ to 1, for each agent v and modifying $f'_{k\text{-MIN}}$ as follows: whenever agents v, u interact at time $t+1$, we set $\mathbf{q}'_v(t+1) = \mathbf{q}'_u(t+1) = \max\{\mathbf{q}_v(t)+1, \mathbf{q}_u(t)+1, \mathbf{q}'_v(t), \mathbf{q}'_u(t)\}$; the rest of $f'_{k\text{-MIN}}$, i.e., the part that affects the first registers and the states, remains unchanged.

References

1 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 47–56, 2015.

2 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18:235–253, 2006.

3 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 292–299, 2006.

4 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

5 James Aspnes and Eric Ruppert. An introduction to population protocols. In Benoît Garbinato, Hugo Miranda, and Luís Rodrigues, editors, *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer-Verlag, 2009.

6 Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In Anne Condon, David Harel, Joost N. Kok, Arto Salomaa, and Erik Winfree, editors, *Algorithmic Bioprocesses*, Natural Computing Series, pages 543–584. Springer Berlin Heidelberg, 2009. `doi:10.1007/978-3-540-88869-7_27`.

7 Fabian Kuhn, Thomas Locher, and Stefan Schmid. Distributed computation of the mode. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 15–24, 2008.

8 Thomas G. Kurtz. *Approximation of Population Processes.* Society for Industrial and Applied Mathematics, 1987.

9 George B. Mertzios, Sotiris E. Nikoletseas, Christoforos L. Raptopoulos, and Paul G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1, pages 871–882, 2014.

10 Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. *New Models for Population Protocols*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2011.

11 Yves Mocquard, Emmanuelle Anceaume, James Aspnes, Yann Busnel, and Bruno Sericola. Counting with population protocols. In *Proceedings of the 14th International Symposium on Network Computing and Applications (NCA)*, pages 35–42, 2015.

12 Prasad Tetali and Peter Winkler. On a random walk problem arising in self-stabilizing token management. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 273–280, 1991.

Shortest Unique Substring Queries on Run-Length Encoded Strings

Takuya Mieno[1], Shunsuke Inenaga[2], Hideo Bannai[3], and Masayuki Takeda[4]

1 **Department of Informatics, Kyushu University, Fukuoka, Japan**
takuya.mieno@inf.kyushu-u.ac.jp
2 **Department of Informatics, Kyushu University, Fukuoka, Japan**
inenaga@inf.kyushu-u.ac.jp
3 **Department of Informatics, Kyushu University, Fukuoka, Japan**
bannai@inf.kyushu-u.ac.jp
4 **Department of Informatics, Kyushu University, Fukuoka, Japan**
takeda@inf.kyushu-u.ac.jp

Abstract

We consider the problem of answering shortest unique substring (SUS) queries on run-length encoded strings. For a string S, a unique substring $u = S[i..j]$ is said to be a *shortest unique substring* (*SUS*) of S containing an interval $[s,t]$ $(i \leq s \leq t \leq j)$ if for any $i' \leq s \leq t \leq j'$ with $j - i > j' - i'$, $S[i'..j']$ occurs at least twice in S. Given a run-length encoding of size m of a string of length N, we show that we can construct a data structure of size $O(m + \pi_s(N,m))$ in $O(m \log m + \pi_c(N,m))$ time such that queries can be answered in $O(\pi_q(N,m) + k)$ time, where k is the size of the output (the number of SUSs), and $\pi_s(N,m)$, $\pi_c(N,m)$, $\pi_q(N,m)$ are, respectively, the size, construction time, and query time for a predecessor/successor query data structure of m elements for the universe of $[1,N]$. Using the data structure by Beam and Fich (JCSS 2002), this results in a data structure of $O(m)$ space that is constructed in $O(m \log m)$ time, and answers queries in $O(\sqrt{\log m / \log\log m} + k)$ time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases string algorithms, shortest unique substring, run-length encoding

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.69

1 Introduction

The *shortest unique substring* (*SUS*) problem is, given a string S of length N and query interval $[s,t] \subseteq [1,N]$ of positions in S, find the shortest substring $S[x..y]$ of S that contains $[s,t]$ (i.e., $[s,t] \subseteq [x,y]$) and is *unique* (i.e., occurs only once) in S. Finding SUSs has possible applications in various fields, including alignment free genome comparison [10], PCR primer design [17], and display of search results [17]. The problem was first introduced by Pei et al. [17], who considered only a single position (i.e., an interval of size 1) as the query input, and showed that the string can be preprocessed in $O(N^2)$ time and $O(N)$ space so that a single SUS for a query position can be returned in constant time. Again for the single position query, Tsuruta et al. [20], Ileri et al. [13], and Hon et al. [11] independently showed that the preprocessing can be improved to $O(N)$ time and space, with constant query time. Tsuruta et al. [20] and Ileri et al. [13] also showed that all SUSs that contain the query position can be answered in $O(k)$ time, where k is the number of SUSs to output. Hu et al. [12] further generalized the problem to interval queries, and showed that it can be solved in $O(N)$ time and space, and all SUSs that contain the query interval can be answered in $O(k)$ time.

In this paper, we consider the SUS problem for interval queries in the case where the string is given in *run-length encoding* (*RLE*). The RLE of a string is a natural compressed representation where each maximal character run of a character a of length e is encoded as a^e. String processing on the compressed representation of a string without explicit decompression [1] is a heavily studied topic, and can lead to time and space efficient processing [18]. There have been many studies on efficient algorithms for processing RLE strings [7, 8, 2, 3, 16, 14, 9, 15, 4]. We show that given a run-length encoding of size m of a string, we can construct a data structure of size $O(m + \pi_s(N, m))$ in $O(m \log m + \pi_c(N, m))$ time such that all SUSs that contain the query interval can be answered in $O(\pi_q(N, m) + k)$ time, where k is the number of such SUSs and $\pi_s(N, m)$, $\pi_c(N, m)$, $\pi_q(N, m)$ are, respectively, the size, construction time, and query time for a predecessor/successor query data structure of m elements for the universe of $[1, N]$. Using the data structure by Beam and Fich [5], this results in a data structure of size $O(m)$ space that is constructed in $O(m \log m)$ time, and answers queries in $O(\sqrt{\log m / \log \log m} + k)$ time. Thus, compared to previous work [12], our algorithm allows for more time and space efficient preprocessing for RLE compressible strings, with a slight increase in query time.

Our result is an outcome of a non-trivial mixed use of combinatorial properties of RLE strings and data structures built on RLE strings: All existing solutions [17, 20, 13, 12, 11] to the SUS problem precompute *minimal unique substrings* (*MUSs*) of a given string, which are minimal substrings of S occurring exactly once in S, and store them in $\Theta(N)$ space, since, in general, there can be $\Theta(N)$ MUSs in a given string. However, using combinatorial properties of MUSs and RLE strings, we show in this paper that any string of RLE size m contains at most $2m - 1$ MUSs, enabling our space-efficient $O(m)$-size data structure for the SUS problem. This bound is indeed tight, namely, some strings contain $2m - 1$ MUSs. In our algorithm, we separately treat MUSs that are completely contained in runs, those that start at the last characters of runs, and the rest. We then show that all the MUSs can be precomputed in $O(m \log m)$ time using a special type of suffix arrays for RLE strings [19]. Finally, we show how, given all MUSs, to efficiently compute all SUSs for any given query interval.

2 Preliminaries

2.1 Notations

Let $\Sigma = \{1, \ldots, \sigma\}$ be an *alphabet*. An element of Σ^* is called a *string*. The length of a string S is denoted by $|S|$. The empty string ε is the string of length 0, namely, $|\varepsilon| = 0$. The i-th character of a string S of length N is denoted by $S[i]$ for $1 \leq i \leq N$. For $1 \leq i \leq j \leq N$, let $S[i..j] = S[i] \cdots S[j]$, i.e., $S[i..j]$ is the *substring* of S starting at position i and ending at position j in S. For convenience, let $S[i..j] = \varepsilon$ if $j < i$. For any $1 \leq i \leq N$, non-empty strings $S[1..i]$ and $S[i..N]$ are respectively called *prefixes* and *suffixes* of S. Let $\mathit{suf}_S(i) = S[i..N]$. For any strings X and Y, let $\mathit{lcp}(X, Y)$ denote the length of the *longest common prefix* of X and Y. For any string S of length n and any $1 \leq i \leq j \leq N$, let $\mathit{lce}_S(i, j) = \mathit{lcp}(\mathit{suf}_S(i), \mathit{suf}_S(j))$. If a string X is lexicographically smaller than another string Y, then we write $X \prec Y$ or $Y \succ X$.

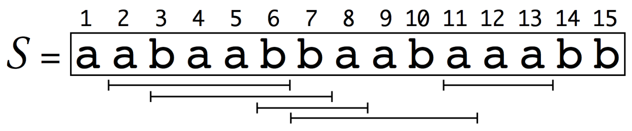


Figure 1 All the MUSs of string S = aabaabbaabaaabb are $S[2..6]$ = abaab, $S[3..7]$ = baabb, $S[6..8]$ = bba, $S[7..11]$ = baaba, and $S[11..13]$ = aaa.

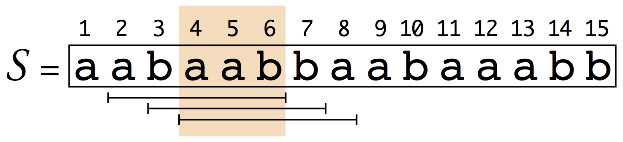


Figure 2 Consider the same string S = aabaabbaabaaabb as in Figure 1. All the SUSs that contain a query interval $[4, 6]$ are $S[2..6]$ = abaab, $S[3..7]$ = baabb, and $S[4..8]$ = aabba. The first two SUSs are also MUSs of S, while the last SUS is obtained by taking the beginning position 4 of the query interval $[4, 6]$ and the ending position of a MUS $S[6..8]$ that overlaps the query interval.

2.2 MUSs and SUSs

For any non-empty strings S and w, let $occ_S(w)$ denote the set of occurrences of w in S, namely, $occ_S(w) = \{i \mid 1 \leq i \leq |S| - |w| + 1, w = S[i..i + |w| - 1]\}$. A substring w of a string S is called a *unique substring* (resp. a *repeat*) of S if $|occ_S(w)| = 1$ (resp. $|occ_S(w)| \geq 2$). In the sequel, we will identify each unique substring w of S with its corresponding (unique) interval $[i, j]$ in S such that $w = S[i..j]$. A unique substring $u = S[i..j]$ is said to be *right minimal unique* if for any $i \leq j' < j$, $S[i..j']$ is a repeat of S. A unique substring $u = S[i..j]$ is said to be *left minimal unique* if for any $i < i' \leq j$, $S[i'..j]$ is a repeat of S. A substring $u = S[i..j]$ is said to be a *minimal unique substring* (*MUS*) of S if u is right minimal unique and left minimal unique. Let $\mathcal{M}_S$ denote the set of all MUSs of S. Also, a unique substring $u = S[i..j]$ is said to be a *shortest unique substring* (*SUS*) of S containing an interval $[s, t]$ $(i \leq s \leq t \leq j)$ if for any $i' \leq s \leq t \leq j'$ with $j - i > j' - i'$, $S[i'..j']$ is a repeat of S. Hu et al. [12] showed that precomputing all MUSs $\mathcal{M}_S$ in a given string S, later allows to efficiently answer all SUSs that contain any query range $[s, t]$. See Figures 1 and 2 for examples of MUSs and SUSs of a string.

2.3 Run-length encodings and our problem

The *run-length encoding* (*RLE*) of string S of length N is a compact representation of S which encodes each maximal character run $S[i..i + e - 1]$ by a^e, if (1) $S[j] = a$ for all $i \leq j \leq i + e - 1$, (2) $S[i - 1] \neq S[i]$ or $i = 1$, and (3) $S[i + e - 1] \neq S[i + e]$ or $i + e - 1 = N$. E.g., $RLE(\mathtt{aabbbbcccaaa\$}) = \mathtt{a}^2\mathtt{b}^4\mathtt{c}^3\mathtt{a}^3\$^1$. The *size* of $RLE(S) = a_1^{e_1} \cdots a_m^{e_m}$ is the number m of maximal character runs in S and is denoted by $|RLE(S)|$. For any $1 \leq i \leq m$, let $bpos_S(i)$, $epos_S(i)$, and $exp_S(i)$ respectively denote the beginning position, ending position, and exponent of the ith run of $RLE(S)$ in the original string S; namely, $bpos_S(i) = 1 + \sum_{k=1}^{i-1} e_k$, $epos_S(i) = \sum_{k=1}^{i} e_k$, and $exp_S(i) = e_i$.

In this paper, we will tackle the following problem:

▶ **Problem 1** (SUSs on RLE strings).
Preprocess: *$RLE(S) = a_1^{e_1} \cdots a_m^{e_m}$ of size m of string S of length N.*
Query: *An interval $[s, t] \in [1, N]$.*
Return: *All SUSs of S containing the query interval $[s, t]$.*

	tRLESA_S^{-1}	EXP_S	tRLESA_S	$\mathsf{tRLELCP}_S$	
1	3	2	10	0	$\mathtt{a}^{(2)}\ \mathtt{b}^2\ \mathtt{c}^3\ \$^1$
2	6	1	6	1	$\mathtt{a}^{(1)}\ \mathtt{c}^2\ \mathtt{a}^2\ \mathtt{b}^2\ \mathtt{c}^3\ \$^1$
3	2	3	3	4	$\mathtt{a}^{(3)}\ \mathtt{c}^2\ \mathtt{a}^1\ \mathtt{c}^2\ \mathtt{a}^2\ \mathtt{b}^2\ \mathtt{c}^3\ \$^1$
4	5	2	12	0	$\mathtt{b}^{(2)}\ \mathtt{c}^3\ \$^1$
5	1	2	8	0	$\mathtt{c}^{(2)}\ \mathtt{a}^2\ \mathtt{b}^2\ \mathtt{c}^3\ \$^1$
6	4	2	5	2	$\mathtt{c}^{(2)}\ \mathtt{a}^1\ \mathtt{c}^2\ \mathtt{a}^2\ \mathtt{b}^2\ \mathtt{c}^3\ \$^1$
7	7	3	15	1	$\mathtt{c}^{(3)}\ \$^1$
8	8	1	16	0	$\$^{(1)}$
9				0	

Figure 3 tRLESA_S, tRLESA_S^{-1}, $\mathsf{tRLELCP}_S$, and EXP_S for $RLE(S) = \mathtt{a}^3\mathtt{c}^2\mathtt{a}^1\mathtt{c}^2\mathtt{a}^2\mathtt{b}^2\mathtt{c}^3\$^1$ with $m = 8$ and $N = |S| = 16$. We remark that the exponents of the first runs in parentheses are all regarded as 1. For instance, consider the suffixes of lexicographical ranks 2 and 3. Although $\mathtt{a}^1\mathtt{c}^2\mathtt{a}^2\mathtt{b}^2\mathtt{c}^3\$^1$ is lexicographically greater than $\mathtt{a}^3\mathtt{c}^2\mathtt{a}^1\mathtt{c}^2\mathtt{a}^2\mathtt{b}^2\mathtt{c}^3\$^1$, $\mathtt{a}^1\mathtt{c}^2\mathtt{a}^2\mathtt{b}^2\mathtt{c}^3\$^1$ is lexicographically smaller than $\mathtt{a}^1\mathtt{c}^2\mathtt{a}^1\mathtt{c}^2\mathtt{a}^2\mathtt{b}^2\mathtt{c}^3\$^1$, and tRLESA_S builds on the latter ordering.

Our model of computation is a standard word RAM with machine word size $\Omega(\log N)$. The space complexity of our algorithm to solve Problem 1 will be evaluated by the number of words (rather than bits).

3 Tools

In this section, we list some data structure which we use to solve Problem 1.

3.1 Suffix arrays and related arrays for RLE strings

Let S be a string of length N and let $B \subseteq [1, N]$ be any subset of positions in S called sampled positions. The sparse suffix array SSA_B of a string S w.r.t. B is an array of size $|B|$ such that $\mathsf{SSA}_B[i] \in B$ for all $1 \le i \le |B|$ and $suf_S(\mathsf{SSA}_B[i]) \prec suf_S(\mathsf{SSA}_B[i+1])$ for all $1 \le i < N$.

We will use the following arrays in our algorithm for computing SUSs on RLE strings. These arrays were first introduced in [19]. Let $m = |RLE(S)|$ and $E = \{epos_S(i) \mid 1 \le i \le m\}$. The *truncated RLE suffix array* for $RLE(S)$, denoted tRLESA_S, is the sparse suffix array of S w.r.t. E. Namely, for any $1 \le i \le m$, $\mathsf{tRLESA}_S[i] = j$ iff $j \in E$ and the lexicographical rank of the suffix $S[j..N]$ is i among all suffixes of S that begin with positions in E. Let tRLESA_S^{-1} be an array of size m such that $\mathsf{tRLESA}_S[\mathsf{tRLESA}_S^{-1}[i]] = epos_S(i)$ for all $1 \le i \le m$. Let $\mathsf{tRLELCP}_S$ be an array of size $m+1$ such that $\mathsf{tRLELCP}_S[1] = \mathsf{tRLELCP}_S[m+1] = 0$ and $\mathsf{tRLELCP}_S[i] = lce_S(\mathsf{tRLESA}_S[i-1], \mathsf{tRLESA}_S[i]) = lcp(suf_S(\mathsf{tRLESA}_S[i-1]), suf_S(\mathsf{tRLESA}_S[i]))$ for all $2 \le i \le m$. Also, let EXP_S be an array of size m such that $\mathsf{EXP}_S[i] = exp_S(k)$ where $\mathsf{tRLESA}_S[i] = epos_S(k)$ for all $1 \le i \le m$, namely, $\mathsf{EXP}_S[i]$ stores the ignored exponent of the first run of the ith suffix in tRLESA_S. See Figure 3 for concrete examples of these arrays.

▶ **Lemma 2** ([19]). *Given $RLE(S)$ of size m, tRLESA_S, tRLESA_S^{-1}, $\mathsf{tRLELCP}_S$, and EXP_S can be computed in a total of $O(m \log m)$ time with $O(m)$ working space.*

The following is a simple observation of these arrays we will exploit.

▶ **Observation 3.** *For any* $1 \leq i \leq m$*, let*

$$l = \max\{\mathsf{tRLELCP}_S[p], \mathsf{tRLELCP}_S[p+1]\},$$

where $p = \mathsf{tRLESA}_S^{-1}[i]$*. If* $l \neq 0$*, then* l *is the length of the longest repeat of* S *that starts at* $epos_S(i)$*.*

For example of Observation 3, see Figure 3. There, for position $i = 3$, we have $p = \mathsf{tRLESA}_S^{-1}[3] = 2$. Then, observe that $l = \max\{1, 4\} = 4$ is the length of the longest repeat $\mathtt{ac}^2\mathtt{a}$ that starts at position $epos_S(3) = 6$. On the other hand, for position $i = 6$, we have $p = \mathsf{tRLESA}_S^{-1}[6] = 4$. Then, $l = \max\{0, 0\} = 0$, but this is not equal to the length 1 of the longest repeat $\mathtt{b}$ that starts at position $epos_S(6) = 12$. In our algorithm, we will use Observation 3 only the case where $l \neq 0$.

3.2 Range minimum/maximum query data structure

Let A be an integer array of size m. Give a query range $[i, j] \in [1, m]$, a *range minimum query* $\mathsf{RmQ}_A(i, j)$ returns the index of a minimum element in the subarray $A[i, j]$, namely, it returns one of $\arg\min_{i \leq k \leq j}\{A[k]\}$. Similarly, *range maximum query* $\mathsf{RMQ}_A(i, j)$ returns the index of a maximum element in the subarray $A[i, j]$, namely, it returns one of $\arg\max_{i \leq k \leq j}\{A[k]\}$. It is well-known (see e.g. [6]) that after an $O(m)$-time preprocessing over the input array A, $\mathsf{RmQ}_A(i, j)$ and $\mathsf{RMQ}_A(i, j)$ can be answered in $O(1)$ time for any query range $[i, j]$, using $O(m)$ space.

3.3 Some functions related to tRLESA

In this subsection, we introduce some functions related to tRLESA_S and the other arrays, which will be used in our algorithm to compute SUSs on RLE strings.

Consider $RLE(S)$ of size m. For any pair $(i, j) \in [1, m] \times [1, m]$, let $trle\_lce_S(i, j) = lce_S(\mathsf{tRLESA}_S[i], \mathsf{tRLESA}_S[j])$. Since

$$trle\_lce_S(i, j) = \begin{cases} \mathsf{RmQ}_{\mathsf{tRLELCP}_S}(i+1, j) & \text{if } i < j, \\ \mathsf{RmQ}_{\mathsf{tRLELCP}_S}(j+1, i) & \text{otherwise}, \end{cases}$$

after a linear-time preprocessing on $\mathsf{tRLELCP}_S$, we can answer $trle\_lce_S(i, j)$ in $O(1)$ time for any given pair (i, j).

For any $1 \leq q \leq m$ and $e \geq 1$, let $exp\_pos(q, e)$ denote a query which returns a position $q' \neq q$, if it exists, that satisfies $\mathsf{EXP}_S[q'] \geq e$ and $S[\mathsf{tRLESA}_S[q']] = S[\mathsf{tRLESA}_S[q]]$ while maximizing $trle\_lce(q, q')$, and *nil* otherwise. Thus, with $q' = exp\_pos(q, e)$, we can obtain the length of the longest repeating substring starting at position $\mathsf{tRLESA}_S[q] - e + 1$ as $e - 1 + trle\_lce(q, q')$.

▶ **Lemma 4.** *Given* EXP_S *for* $RLE(S)$ *of size* m*, we can preprocess* EXP_S *in* $O(m)$ *time so that subsequent* $exp\_pos(q, e)$ *queries can be answered in* $O(\log m)$ *time for any* $1 \leq q \leq m$ *and* $e \geq 1$*.*

Proof. We construct an RMQ data structure for EXP_S in $O(m)$ time. Since lexicographically close strings share a longer prefix, $exp\_pos(q, e)$ is one of the two closest neighbours of q in EXP_S that stores an exponent at least e, corresponding to a run of the same character. Thus, we can compute $exp\_pos(q, e)$ using two binary searches on EXP, by comparing e with the answer of the RMQ queries, starting with the initial range $[1, q-1]$ and $[q+1, m]$. Since the size of EXP_S is m and each RMQ query takes $O(1)$ time, it takes $O(\log m)$ time to locate $exp\_pos(q, e)$. ◀

For any $1 \le q \le m$ and $\ell \ge 0$, let $lce\_pos(q, \ell)$ denote a query which returns a position $q' \ne q$, if it exists, such that $trle\_lce(q, q') \ge \ell$ while maximizing $\mathsf{EXP}_S[q']$, and nil otherwise. In other words, $lce\_pos(q, \ell)$ corresponds to a suffix that has the maximum exponent out of suffixes which, have a common prefix of length ℓ with the suffix corresponding to q. Note that if $\ell > \max\{\mathsf{tRLELCP}[q], \mathsf{tRLELCP}[q+1]\}$, $lce\_pos(q, \ell) = nil$.

▶ **Lemma 5.** *Given* $\mathsf{tRLELCP}_S$ *for* $RLE(S)$ *of size* m*, we can preprocess* $\mathsf{tRLELCP}_S$ *in* $O(m)$ *time so that subsequent* $lce\_pos(q, \ell)$ *queries can be answered in* $O(\log m)$ *time for any* $1 \le q \le m$ *and* $\ell \ge 0$.

Proof. We construct an RmQ data structure on $\mathsf{tRLELCP}_S$. Since, as noted previously, lexicographically close strings share a longer prefix, values of $trle\_lce(q, q'')$ are larger when q'' is closer to q. Thus, similar to Lemma 4, we can conduct two binary searches on $\mathsf{tRLELCP}_S$ using RmQ and obtain the maximal range $[q_p, q_n]$ such that $trle\_lce(q, q'') \ge \ell$ if and only if $q'' \in [q_p, q_n]$. After finding the range, the larger of the two RMQ queries for the ranges $[q_p, q-1]$ and $[q+1, q_n]$ on EXP_S gives the answer. ◀

3.4 Predecessor/successor query data structure

Let $A[1..m]$ be an array containing positive integers less than or equal to N, in increasing order. The predecessor and successor queries on A are defined for any $1 \le d \le N$ as

$$\mathsf{Pred}_A(d) = \begin{cases} \max\{i \mid A[i] \le d\} & \text{if it exists,} \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathsf{Succ}_A(d) = \begin{cases} \min\{i \mid A[i] \ge d\} & \text{if it exists,} \\ N+1 & \text{otherwise.} \end{cases}$$

There exists a data structure of $O(m)$ space that can be built in $O(m\sqrt{\log m / \log\log m})$ time, such that later, for any given $1 \le d \le N$, $\mathsf{Pred}_A(d)$ and $\mathsf{Succ}_A(d)$ can be answered in $O(\sqrt{\log m / \log\log m})$ time [5].

4 Computing MUSs from RLE strings

In this section we show how we can compute $\mathcal{M}_S$ given $RLE(S)$, which is the main part of our preprocessing. As will be seen in Section 4.1, we partition MUSs into three disjoint groups; those that are completely contained in runs, those that start at the last characters of runs, and the rest.

4.1 Size of $\mathcal{M}_S$

We begin with the analysis of the size of $\mathcal{M}_S$ in terms of $m = |RLE(S)|$. Let

$$\begin{aligned} \mathcal{M}^{(1)} &= \{[x, y] \in \mathcal{M}_S \mid bpos_S(i) \le x \le y \le epos_S(i) \text{ for some } 1 \le i \le m\}, \\ \mathcal{M}^{(2)} &= \{[x, y] \in \mathcal{M}_S \mid x = epos_S(i) < y \text{ for some } 1 \le i < m\}, \text{ and} \\ \mathcal{M}^{(3)} &= \{[x, y] \in \mathcal{M}_S \mid bpos_S(i) \le x < epos_S(i) < y \text{ for some } 1 \le i < m\}. \end{aligned}$$

Clearly, $\mathcal{M}_S = \mathcal{M}^{(1)} \cup \mathcal{M}^{(2)} \cup \mathcal{M}^{(3)}$. For example, for the same string $S = \mathtt{aaaccaccaabbccc\$}$ as in Figure 3, $\mathcal{M}_S = \{[1,3],[2,4],[5,7],[8,10],[10,11],[11,12],[12,13],[13,15],[16,16]\} = \{\mathtt{aaa}, \mathtt{aac}, \mathtt{cac}, \mathtt{caa}, \mathtt{ab}, \mathtt{bb}, \mathtt{bc}, \mathtt{ccc}, \$\}$, $\mathcal{M}^{(1)} = \{\mathtt{aaa}, \mathtt{bb}, \mathtt{ccc}, \$\}$, $\mathcal{M}^{(2)} = \{\mathtt{cac}, \mathtt{caa}, \mathtt{ab}, \mathtt{bc}\}$, and $\mathcal{M}^{(3)} = \{\mathtt{aac}\}$.

Since, by definition, a MUS cannot be a proper substring of another MUS, there can be at most one MUS that starts at any given position. Thus, it follows that $|\mathcal{M}^{(2)}| \leq m-1$.

For $|\mathcal{M}^{(3)}|$, we have the following lemma.

▶ **Lemma 6.** *For any $[x,y] \in \mathcal{M}^{(3)}$ and $p \in occ_S(S[x+1..y]) \setminus \{x+1\}$, we have that $p = bpos_S(i)$ for some $1 \leq i < m$.*

Proof. Since $S[x..y]$ is a MUS, $S[x+1..y]$ is not unique and thus $occ_S(S[x+1..y]) \setminus \{x+1\}$ is not empty. If $p \neq bpos_S(i)$ for any $1 \leq i < m$, then $S[p-1] = S[p]$ and thus gives another occurrence of $S[x..y]$ contradicting that it is unique. ◀

We now show $|\mathcal{M}^{(3)} \cup \mathcal{M}^{(1)}| \leq m$. Let $\mathcal{R} = \{bpos_S(i) \mid 1 \leq i \leq m\}$. From Lemma 6, we can define a function $f : \mathcal{M}^{(3)} \cup \mathcal{M}^{(1)} \to \mathcal{R}$ as follows:

$$f([x,y]) = \begin{cases} \min(occ_S(S[x+1..y]) \setminus \{x+1\}) & \text{if } [x,y] \in \mathcal{M}^{(3)} \\ x & \text{if } [x,y] \in \mathcal{M}^{(1)} \end{cases}$$

Suppose f is not an injective function, i.e., there exist distinct intervals $[x_1,y_1],[x_2,y_2] \in \mathcal{M}^{(3)} \cup \mathcal{M}^{(1)}$ such that $x_1 \neq x_2$ and $p = f([x_1,y_1]) = f([x_2,y_2])$. Note that by definition, $\mathcal{M}^{(3)} \cap \mathcal{M}^{(1)} = \emptyset$.

If $[x_1,y_1],[x_2,y_2] \in \mathcal{M}^{(3)}$, assume w.l.o.g. $y_1 - x_1 \leq y_2 - x_2$. By definition of f, we have $S[x_1+1..y_1] = S[p..p+y_1-x_1-1]$ and $S[x_2+1..y_2] = S[p..p+y_2-x_2-1]$. Also, from the definition of $\mathcal{M}^{(3)}$, we have $S[x_1] = S[x_1+1] = S[p] = S[x_2+1] = S[x_2]$. It follows that $S[x_1..y_1]$ is a prefix of $S[x_2..y_2]$, contradicting that $S[x_1..y_1]$ is unique. If $[x_1,y_1] \in \mathcal{M}^{(3)}$ and $[x_2,y_2] \in \mathcal{M}^{(1)}$, this implies that $S[x_2..y_2]$ is a prefix of $S[x_1+1..y_1]$ which is not unique, thus contradicting that $S[x_2..y_2]$ is unique. Finally, if $[x_1,y_1],[x_2,y_2] \in \mathcal{M}^{(1)}$, $p = f([x_1,y_1]) = f([x_2,y_2])$ implies that $p = x_1 = x_2$ contradicting that $x_1 \neq x_2$. Thus, f must be an injective function. Therefore, $|\mathcal{M}^{(3)} \cup \mathcal{M}^{(1)}| \leq |\mathcal{R}| = m$.

From the above arguments, we have:

▶ **Lemma 7.** $|\mathcal{M}_S| \leq 2m-1$.

We note that the upper bound of Lemma 7 is tight, and there exists a string S such that $|\mathcal{M}_S| = 2m-1$. Consider $S = a_1^{e_1}a_2^{e_2}\cdots a_m^{e_m}$ such that for any $1 \leq i,j \leq m$, $e_i \geq 2$, and $a_i \neq a_j$ when $i \neq j$. Clearly, $a_i^{e_i}$ is a MUS for all $1 \leq i \leq m$, and a_ia_{i+1} is a MUS for all $1 \leq i < m$, giving $2m-1$ MUSs.

4.2 Computing $\mathcal{M}_S$

We now show how to obtain $\mathcal{M}_S$ in $O(m \log m)$ time and $O(m)$ space, by computing the sets $\mathcal{M}^{(1)}, \mathcal{M}^{(2)}, \mathcal{M}^{(3)}$ as defined in Section 4.1.

4.2.1 Computing $\mathcal{M}^{(1)}$

To compute $\mathcal{M}^{(1)}$, we first show a necessary and sufficient condition for an interval $[x,y]$ to be in $\mathcal{M}^{(1)}$.

▶ **Lemma 8.** *For any string S where $RLE(S) = a_1^{e_1}\cdots a_m^{e_m}$, an interval $[x,y] \in \mathcal{M}^{(1)}$ if and only if there exists some $1 \leq i \leq m$ such that $bpos_S(i) = x$, $epos_S(i) = y$, and for any $j \in [1,m] \setminus \{i\}$, either $a_i \neq a_j$ or $e_j < e_i$.*

Proof. ($\Rightarrow$) Since $[x, y]$ is a MUS and any proper substring of $[x, y]$ is not unique, it must be that $x = \mathit{bpos}_S(i), y = \mathit{epos}_S(i)$ for some $1 \leq i \leq m$. Furthermore, it must be that $a_i \neq a_j$ or $e_j < e_i$ for any $j \in [1, m] \setminus \{i\}$, since otherwise, $[x, y]$ will not be unique. ($\Leftarrow$) The condition implies that $S[x..y]$ is the longest run of character a_i in S and is unique. Since any proper substring of $S[x..y]$ is not unique, $[x, y]$ is a MUS and is thus in $\mathcal{M}^{(1)}$. ◀

Let Σ_S be the subset of Σ consisting of letters occurring in S. Using Lemma 8, we can compute $\mathcal{M}^{(1)}$ by simply checking for each character $a \in \Sigma_S$, whether there exists a run of character a with a unique (w.r.t. runs of character a) maximum exponent, and if so, include the interval corresponding to the run in $\mathcal{M}^{(1)}$. Since $|\Sigma_S| \leq m$, this can be done in $O(m \log m)$ time and $O(m)$ space using any standard sorting algorithm.

4.2.2 Computing $\mathcal{M}^{(2)}$

To compute, $\mathcal{M}^{(2)}$, we check for each $1 \leq i \leq m-1$, whether there exists a MUS that starts at $\mathit{epos}_S(i)$ and insert it in $\mathcal{M}^{(2)}$ if there is. More specifically, we first compute y such that $S[\mathit{epos}_S(i)..y]$ is right minimal unique. Next, we check whether $S[\mathit{epos}_S(i) + 1..y]$ is unique or not, and if not, we have that $[\mathit{epos}_S(i), y]$ is also left minimal unique and thus is a MUS.

Let $r = \mathsf{tRLESA}_S^{-1}[i]$. By Observation 3, we have that $l = \max\{\mathsf{tRLELCP}_S[r], \mathsf{tRLELCP}_S[r+1]\}$ is the length of the longest repeat of S that starts at $\mathit{epos}_S(i)$. This implies that $S[\mathit{epos}_S(i)..\mathit{epos}_S(i) + l]$ is right minimal unique. Thus, given the $\mathsf{tRLELCP}_S$ array, $y = \mathit{epos}_S(i) + l$ can be computed in constant time. Next, to determine whether $S[\mathit{epos}_S(i) + 1..y]$ is unique or not, we compute y' such that $S[\mathit{epos}_S(i) + 1..y']$ is right minimal unique. Then, $[\mathit{epos}_S(i) + 1, y]$ is unique iff $y' \leq y$. Noticing that $\mathit{epos}_S(i) + 1 = \mathit{bpos}_S(i + 1)$, we can compute y' as follows. Let $q = \mathsf{tRLESA}_S^{-1}[i + 1]$ and $x = \mathsf{EXP}_S[q]$. We compute $l' = x - 1 + \mathit{trle\_lce}_S(q, q')$, where $q' = \mathit{exp\_pos}(q, x)$. By definition, we have that l' is the length of the longest repeat of S that starts at $\mathit{bpos}_S(i + 1)$. Thus, $y' = \mathit{bpos}_S(i + 1) + l'$. By Lemma 4, this can be computed in $O(m \log m)$ total time and $O(m)$ space for all i.

4.2.3 Computing $\mathcal{M}^{(3)}$

For each $1 \leq i < m$, we will compute the elements of $\mathcal{M}^{(3)}$ that start in the ith run. Let $s = \mathit{bpos}_S(i)$ and we repeat the following while $s < \mathit{epos}_S(i)$. First, compute y such that $S[s..y]$ is right minimal unique. If such y does not exists, i.e., $S[s..|S|]$ is not unique, then we are done. If y does exist, $y \geq \mathit{epos}_S(i)$ since, as noted earlier, no proper substring of a run can be unique. If $y = \mathit{epos}_S(i)$, we must have that $s = \mathit{bpos}_S(i)$ and $[s, y]$ is a MUS in $\mathcal{M}^{(1)}$ and not in $\mathcal{M}^{(1)}$; thus we simply increment s by 1 and repeat the process. Otherwise, if $y > \mathit{epos}_S(i)$, we try to find x such that $S[x..y]$ is left minimal unique. Then, by definition, $[x, y]$ is a MUS. If $x < \mathit{epos}_S(i)$, then we have that $[x, y]$ is a MUS in $\mathcal{M}^{(3)}$, and since there can be no other MUS that starts in the interval $[s, x]$, we set $s = x + 1$ and repeat the process. Otherwise, if $x \geq \mathit{epos}_S(i)$, then $[x, y]$ is either a MUS in $\mathcal{M}^{(2)}$ or does not start in the ith run, so we are finished for the current value of i. Because we obtain one distinct MUS each time we determine y and x, the above process is repeated for a total of $O(m)$ times for all i by Lemma 7. What remains is how to determine y and x.

Whether $y = \mathit{epos}_S(i)$ or not can be determined by checking if $[s, \mathit{epos}_S(i)]$ is a MUS in $\mathcal{M}^{(1)}$ as described in Section 4.2.1. Next, we assume $y \geq \mathit{epos}_S(i) + 1 = \mathit{bpos}_S(i + 1)$. Let $q = \mathsf{tRLESA}_S^{-1}[i]$, $q' = \mathit{exp\_pos}(q, \mathit{epos}_S(i) - s + 1)$. If q' is nil, this implies that no run other than the ith one contains a run of character $S[\mathit{epos}_S(i)]$ with length at least $\mathit{epos}_S(i) - s + 1$. Since $y > \mathit{epos}_S(i)$, we have that $S[s..\mathit{epos}_S(i)]$ is not unique but $S[s..\mathit{bpos}_S(i + 1)]$ is unique and thus, $y = \mathit{bpos}_S(i+1)$. Otherwise, if q' is not nil, then, we have that $\mathit{epos}_S(i) - s + l$, where

$l = \mathit{trle\_lce}_S(q, q')$, is the length of the longest repeat of S that starts at s. Therefore, we have $y = \mathit{epos}_S(i) + l$. From the above arguments and Lemma 4, y can be determined in $O(\log m)$ time. Whether $x \geq \mathit{epos}_S(i)$ or not can be determined by the arguments for checking whether $[\mathit{epos}_S(i), y]$ is a MUS in $\mathcal{M}^{(2)}$, as described in Section 4.2.2. Next, we assume $x < \mathit{epos}_S(i)$. Then, $S[\mathit{epos}_S(i)..y]$ is a repeat. Let $q = \mathsf{tRLESA}_S^{-1}(i)$, $q' = \mathit{lce\_pos}(q, y - \mathit{epos}_S(i) + 1)$. From the definition of $\mathit{lce\_pos}$, we have $x = \mathit{epos}_S(i) - \mathsf{EXP}_S[q'] + 1$. Thus, from the above arguments and Lemma 5, x can be determined in $O(\log m)$ time.

The arguments from Sections 4.2.1-4.2.3 lead to the following lemma.

▶ **Lemma 9.** *For any string S, the set $\mathcal{M}_S$ can be computed from $RLE(S)$ in $O(m \log m)$ time using $O(m)$ space, where $m = |RLE(S)|$.*

5 Solution to the SUS Problem

5.1 Data structure

Our data structure consists of three arrays: X_S, Y_S, and MUSlen_S. Arrays X_S and Y_S are arrays of size $|\mathcal{M}_S|$ such that for any $1 \leq i \leq |\mathcal{M}_S|$, $[\mathsf{X}_S[i], \mathsf{Y}_S[i]]$ is the ith MUS in order of their start position in S. Also, let the array $\mathsf{MUSlen}_S[i] = \mathsf{Y}_S[i] - \mathsf{X}_S[i] + 1$ hold the length of each MUS. Arrays X_S and Y_S are preprocessed for Succ and Pred queries, and MUSlen_S is preprocessed for RmQ queries. From arguments in previous sections, the preprocessing can clearly be done in a total of $O(m \log m)$ time and $O(m)$ space.

5.2 Answering queries

For any two intervals $[s, t]$ and $[x, y]$, let $\mathit{cover}([s, t], [x, y])$ be the smallest interval that contains both $[s, t], [x, y]$, i.e., $\mathit{cover}([s, t], [x, y]) = [\min\{s, x\}, \max\{t, y\}]$.

Given a query interval $[s, t]$, let $i = \mathsf{Pred}_{\mathsf{Y}_S}(t)$ and $j = \mathsf{Succ}_{\mathsf{X}_S}(s)$. Clearly, all SUSs that contain interval $[s, t]$ are contained in the set $\{|\mathit{cover}([s, t], [\mathsf{X}_S[r], \mathsf{Y}_S[r]])| \mid i \leq r \leq j\}$. Thus, it suffices to find the intervals of smallest size in this set, i.e., if $p \in \arg\min\{|\mathit{cover}([s, t], [\mathsf{X}[r], \mathsf{Y}[r]])| \mid i \leq r \leq j\}$, then $\mathit{cover}([s, t], [\mathsf{X}_S[p], \mathsf{Y}_S[p]])$ is a SUS. Notice that for all $i < r < j$, we have that $\mathit{cover}([s, t], [\mathsf{X}_S[r], \mathsf{Y}_S[r]]) = [\mathsf{X}_S[r], \mathsf{Y}_S[r]]$. Thus, the shortest of these can be found by considering $\mathit{cover}([s, t], [\mathsf{X}_S[i], \mathsf{Y}_S[i]])$, $\mathit{cover}([s, t], [\mathsf{X}_S[j], \mathsf{Y}_S[j]])$, and performing an RmQ query on MUSlen_S. An example is shown in Figure 4. For finding a single SUS, the query time is dominated by the Pred and Succ queries, and thus is $O(\pi_q(N, m))$ time. To output all SUSs that contain $[s, t]$, recursive RmQ on sub-intervals of MUSlen_S can be conducted in constant time per output, in order to find all the shortest intervals in the range $[i, j]$. Thus, the total query time is $O(\pi_q(N, m) + k)$, where k is the total number of SUSs that are output.

Putting everything together, we have proved the following theorem:

▶ **Theorem 10.** *Given $RLE(S)$ of size m representing a string S of length N, we can compute in $O(m \log m + \pi_c(N, m))$ time a data structure of size $O(m + \pi_s(N, m))$ which answers SUS queries for any interval $[s, t] \subseteq [1, N]$ in $O(\pi_q(N, m) + k)$ time, where k is the number of SUSs to output.*

Using known results for predecessor/successor queries [5], we obtain the following corollary.

▶ **Corollary 11.** *Given $RLE(S)$ of size m representing a string S of length N, we can compute in $O(m \log m)$ time a data structure of size $O(m)$ which answers SUSs queries for any interval $[s, t] \subseteq [1, N]$ in $O(\sqrt{\log m / \log \log m} + k)$ time, where k is the number of such SUSs.*

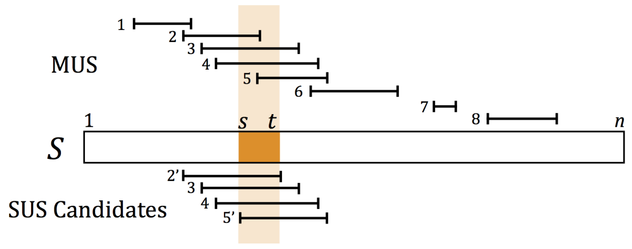


Figure 4 Finding SUSs that contains query interval $[s, t]$. The SUS must be either a MUS that completely contains $[s, t]$ (MUS 3,4), or, it must be an interval that covers both $[s, t]$ and the preceding MUS (MUS 2) or succeeding MUS (MUS 5). Of these, the intervals with shortest length are the SUSs that contain $[s, t]$.

6 Conclusions and open question

We considered the problem of finding all shortest unique substrings (SUSs) of a string S given as the run-length encoding (RLE) of size m. We showed that we can preprocess the RLE in $O(m \log m)$ time and $O(m)$ space so that subsequent SUS queries for S can be answered in $O(\sqrt{\log m / \log\log m} + k)$ time, where k is the number of outputs for the query interval. Notice that none of the preprocessing time, space requirement, or query time depends on the original length N of the string S. This efficiency was achieved by a non-trivial use of the suffix arrays for RLE strings and by revealing combinatorial properties of MUSs and SUSs on RLE strings.

The $\sqrt{\log m / \log\log m}$ term in our query time is due to the use of the $O(m)$-space dynamic predecessor/successor data structure by Beame and Fich [5]. They also showed that for a static set $\mathcal{A}$ of m integers from the universe $[1, N]$, any predecessor/successor data structure for $\mathcal{A}$ of polynomial size in m must use $\Omega(\sqrt{\log m / \log\log m})$ query time (Corollary 3.10 of [5]). Notice that once we build arrays X_S and Y_S, they will remain static. Hence, we cannot hope for faster SUS query time as long as we use predecessor/successor queries to find a MUS for a given interval. Thus, an interesting open question is whether there exists a data structure of size $O(m)$ that can efficiently answer SUS queries without using predecessor/successor queries.

Acknowledgements. The authors especially thank an anonymous reviewer for invaluable comments which helped improve the paper. SI, HB, MT were partly supported by JSPS KAKENHI Grant Numbers 26280003, 16H02783, 25240003.

References

1 Amihood Amir, Gary Benson, and Martin Farach. Let sleeping files lie: pattern matching in z-compressed files. *Journal of Computer and System Sciences*, 52(2):299–307, April 1996.

2 Alberto Apostolico, Gad M. Landau, and Steven Skiena. Matching for run-length encoded strings. *J. Complex.*, 15(1):4–16, March 1999. doi:10.1006/jcom.1998.0493.

3 Ora Arbell, Gad M. Landau, and Joseph S.B. Mitchell. Edit distance of run-length encoded strings. *Information Processing Letters*, 83(6):307–314, 2002. doi:10.1016/S0020-0190(02)00215-6.

4 Golnaz Badkobeh, Gabriele Fici, Steve Kroon, and Zsuzsanna Lipták. Binary jumbled string matching for highly run-length compressible texts. *Information Processing Letters*, 113(17):604–608, 2013. doi:10.1016/j.ipl.2013.05.007.

5 Paul Beame and Faith E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002. `doi:10.1006/jcss.2002.1822`.

6 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, LATIN 2000, pages 88–94, 2000. URL: `http://dl.acm.org/citation.cfm?id=646388.690192`.

7 Horst Bunke and János Csirik. An algorithm for matching run-length coded strings. *Computing*, 50(4):297–314, 1993.

8 Horst Bunke and János Csirik. An improved algorithm for computing the edit distance of run-length coded strings. *Information Processing Letters*, 54(2):93–96, April 1995.

9 Kuan-Yu Chen, Ping-Hui Hsu, and Kun-Mao Chao. Efficient retrieval of approximate palindromes in a run-length encoded string. *Theoretical Computer Science*, 432:28–37, 2012. `doi:10.1016/j.tcs.2012.01.023`.

10 Bernhard Haubold, Nora Pierstorff, Friedrich Möller, and Thomas Wiehe. Genome comparison without alignment using shortest unique substrings. *BMC Bioinformatics*, 6(1):123, 2005.

11 Wing-Kai Hon, Sharma V. Thankachan, and Bojian Xu. An in-place framework for exact and approximate shortest unique substring queries. In *ISAAC 2015*, pages 755–767, 2015.

12 Xiaocheng Hu, Jian Pei, and Yufei Tao. Shortest unique queries on strings. In *Proc. SPIRE 2014*, pages 161–172, 2014.

13 Atalay Mert Ileri, M. Oguzhan Külekci, and Bojian Xu. A simple yet time-optimal and linear-space algorithm for shortest unique substring queries. *Theor. Comput. Sci.*, 562:621–633, 2015.

14 Jin Wook Kim, Amihood Amir, Gad M. Landau, and Kunsoo Park. Computing similarity of run-length encoded strings with affine gap penalty. *Theoretical Computer Science*, 395(2–3):268–282, 2008. SAIL – String Algorithms, Information and Learning: Dedicated to Professor Alberto Apostolico on the occasion of his 60th birthday. `doi:10.1016/j.tcs.2008.01.008`.

15 Jia-Jie Liu, Guan-Shieng Huang, and Yue-Li Wang. A fast algorithm for finding the positions of all squares in a run-length encoded string. *Theoretical Computer Science*, 410(38–40):3942–3948, 2009. `doi:10.1016/j.tcs.2009.05.032`.

16 Mäkinen, Ukkonen, and Navarro. Approximate matching of run-length compressed strings. *Algorithmica*, 35(4):347–369, 2003. `doi:10.1007/s00453-002-1005-2`.

17 Jian Pei, Wush Chi-Hsuan Wu, and Mi-Yen Yeh. On shortest unique substring queries. In *Proc. ICDE 2013*, pages 937–948, 2013.

18 Masayuki Takeda. *Encyclopedia of algorithms*, chapter "Compressed Pattern Matching", pages 171–174. Springer US, 2008.

19 Yuya Tamakoshi, Keisuke Goto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. An opportunistic text indexing structure based on run length encoding. In *Proc. CIAC 2015*, pages 390–402, 2015.

20 Kazuya Tsuruta, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Shortest unique substrings queries in optimal time. In *Proc. SOFSEM 2014*, pages 503–513, 2014.

Shattered Sets and the Hilbert Function

Shay Moran[1] **and Cyrus Rashtchian**[2]

1 **Department of Computer Science, Technion, Israel; and Microsoft Research, Hertzelia, Israel; and Max Planck Institute for Informatics, Saarbrücken, Germany**
shaymoran1@gmail.com

2 **Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA**
cyrash@cs.washington.edu

Abstract

We study complexity measures on subsets of the boolean hypercube and exhibit connections between algebra (the Hilbert function) and combinatorics (VC theory). These connections yield results in both directions. Our main complexity-theoretic result demonstrates that a large and natural family of linear program feasibility problems cannot be computed by polynomial-sized constant-depth circuits. Moreover, our result applies to a stronger regime in which the hyperplanes are fixed and only the directions of the inequalities are given as input to the circuit. We derive this result by proving that a rich class of extremal functions in VC theory cannot be approximated by low-degree polynomials. We also present applications of algebra to combinatorics. We provide a new algebraic proof of the Sandwich Theorem, which is a generalization of the well-known Sauer-Perles-Shelah Lemma. Finally, we prove a structural result about downward-closed sets, related to the Chvátal conjecture in extremal combinatorics.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, G.2.3 Applications

Keywords and phrases VC Dimension, Shattered Sets, Sandwich Theorem, Hilbert Function, Polynomial Method, Linear Programming, Chvatal's Conjecture, Downward-closed Sets.

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.70

1 Introduction

Understanding the properties and structure of subsets of the boolean hypercube is a central theme in theoretical computer science and combinatorics. When studying a family of mathematical objects, endowing the objects with algebraic structure often sheds new light on interesting properties. This phenomena appears classically in areas such as algebraic topology and algebraic geometry. It also provides much utility when studying the boolean hypercube. Let $C \subseteq \{0,1\}^n$ be a subset of the boolean hypercube, and let $\mathbb{F}$ be a field. Consider the linear space of functions from C to $\mathbb{F}$, that is, $\mathbb{F}^C$. This is clearly a $|C|$-dimensional vector space over $\mathbb{F}$. Every function in this space can be represented as a multilinear polynomial with degree at most n. Interestingly, for certain sets C, smaller degree actually suffices. For example, when C is the standard basis, denoted $C = \{\vec{e}_1, \ldots, \vec{e}_m\}$, then any function $f : C \to \mathbb{F}$ can be expressed as the linear function $f(\vec{e}_1)x_1 + \ldots + f(\vec{e}_m)x_m$.

The Hilbert function, denoted $h_d(C, \mathbb{F})$, is the dimension of the space of functions $\{f : C \to \mathbb{F}\}$ that have representations as polynomials with degree at most d. This classical algebraic object will be useful in our study of how the structure of C affects the function space. In complexity theory, Smolensky [40] has used the Hilbert function to unify polynomial approximation lower bounds relating to bounded-depth circuits.

We establish new connections between the Hilbert function and VC theory. Our main technical contributions are bounds $h_d(C, \mathbb{F})$ in terms of basic concepts in VC theory, such as shattering, strong shattering, and down-shifts. Previous results on bounding the Hilbert function utilize a more intricate analysis and focus on symmetric sets, that is, unions of slices of the hypercube [40, 11]. In addition to giving new bounds on the Hilbert function, our connection between Algebra and Combinatorics allows us to derive results in both directions.

Our main complexity theoretical application is that determining feasibility of a large family of linear programs is hard for the class of bounded-depth circuits. More specifically, let $h_1, \dots, h_m$ be affine functions. Each sign vector s in $\{\pm\}^m$ defines the following feasibility problem: does there exist $x \in \mathbb{R}^d$ such that $h_i(x) > 0$ when $s_i = +$, and $h_i(x) < 0$ when $s_i = -$, for all $i \in [m]$? This defines a boolean function that takes an input s and outputs one if and only if the problem is feasible. We prove that if $m = 2d + 1$, and the affine functions h_i are in general position, then this function cannot be approximated by low-degree polynomials, over any field. This implies a lower bound on the computability of this function by constant-depth circuits, due to the polynomial approximation technique introduced by Razborov [37] and Smolensky [41]. The above linear programming problem relates to the study of hyperplane arrangements (see the books of Matoušek [30] and Stanley [43] for more details and applications). Our results implicitly provide algebraic proofs of some known facts regarding the combinatorics of hyperplane arrangements.

As a combinatorial application of our bounds on the Hilbert function, we provide a short algebraic proof of the Sandwich Theorem. This theorem comes from VC theory and is a well-studied generalization of the Sauer-Shelah-Perles Lemma [29, 36, 15, 14, 18, 5, 8, 33, 28, 32, 35]. Similar proofs of related upper bounds have appeared previously, due to Frankl and Pach [7], Gurvits [24], and Smolensky [42]. We contribute new lower bounds and applications.

Facts we prove about the function space $\mathbb{F}^C$ also lead to a new result about downward-closed sets. A family D of subsets is *downward-closed* if $b \subseteq a$ and $a \in D$ implies $b \in D$. A theorem of Berge [10] implies that for any downward-closed set D there exists a bijection $\pi : D \to D$ such that $a \cap \pi(a) = \emptyset$ for all $a \in D$. We generalize his result to arbitrary, prescribed intersections. Let $\phi : D \to D$ have the property $\phi(a) \subseteq a$ for all $a \in D$. We show that there always exists a bijection $\pi : D \to D$ such that $a \cap \pi(a) = \phi(a)$. Note that choosing $\phi(a) = \emptyset$ for all a implies Berge's result.

Our algebra-combinatorics connection fits within the framework of the polynomial method. This method has been successful in providing elegant proofs of fundamental results in many areas, such as circuit complexity [41, 6, 37, 9], discrete geometry [25, 19, 39, 44], extremal combinatorics [1, 26, 7], and more.

The paper is organized as follows. We state our main theorems in Section 2. In Section 3, we prove our bounds on the Hilbert function. In Section 4, we use our Hilbert function bounds to prove that linear program feasibility is hard for bounded-depth circuits. Finally, in Section 5, we prove results about downward-closed sets. We now review preliminaries.

1.1 Preliminaries

We begin with algebraic preliminaries. Let $C \subseteq \{0,1\}^n$ and $\mathbb{F}$ be a field. Every $f : C \to \mathbb{F}$ can be expressed as a multilinear polynomial over variables $x_1, \dots, x_n$ with coefficients in $\mathbb{F}$.

▶ **Definition 1.** For $d \in [n]$ the *Hilbert function* $h_d(C, \mathbb{F})$ is the dimension of the space of functions $f : C \to \mathbb{F}$ that can be represented as polynomials with degree at most d.

Notice that $h_d(C, \mathbb{F}) \le \min\{\sum_{j=0}^{d} \binom{n}{j}, |C|\}$. A basic fact about the Hilbert function is that

$$1 = h_0(C, \mathbb{F}) \le h_1(C, \mathbb{F}) \le \ldots \le h_n(C, \mathbb{F}) = |C|.$$

The final equality holds because all $f : C \to \mathbb{F}$ have representations with degree at most n.

It is natural to wonder when the Hilbert function attains its maximum and how the structure of C influences the Hilbert function. We introduce the following measure.

▶ **Definition 2.** The **interpolation degree of** C denoted $\mathsf{intdeg}(C, \mathbb{F})$ is the minimum d such that any $f : C \to \mathbb{F}$ can be expressed as a multilinear polynomial with degree at most d. In other words,

$$\mathsf{intdeg}(C, \mathbb{F}) = \min\{d \in [n] : h_d(C, \mathbb{F}) = |C|\}.$$

Intuitively, a smaller interpolation degree implies a less complex set.

We move on to combinatorial preliminaries. Our bounds on the Hilbert function use basic concepts in VC theory. We define these concepts now.

▶ **Definition 3.** A subset $I \subseteq [n]$ is **shattered** by $C \subseteq \{0, 1\}^n$ if for every pattern $s : I \to \{0, 1\}$ there is $c \in C$ that realizes s. In other words, the restriction of c to I equals s. A subset $I \subseteq [n]$ is **strongly shattered** by C if C contains all elements of some subcube on I. In other words, there exists a pattern $\bar{s} : ([n] \setminus I) \to \{0, 1\}$ such that all extensions of $\bar{s}$ to a vector in $\{0, 1\}^n$ are in C.

These definitions lead to natural families of sets, which will be important to our work.

▶ **Definition 4.** The **shattered sets** with respect to C are

$$\mathsf{str}(C) = \{I \subseteq [n] : I \text{ is shattered by } C\}.$$

The **strongly shattered sets** with respect to C are

$$\mathsf{sstr}(C) = \{I \subseteq [n] : I \text{ is strongly shattered by } C\}.$$

▶ **Definition 5.** The **VC dimension** of C is defined as $\mathsf{VC}(C) = \max\{|I| : I \in \mathsf{str}(C)\}$.

Note that $\mathsf{sstr}(C) \subseteq \mathsf{str}(C)$ and that both of these families are downward-closed.

We also lower bound the Hilbert function using down-shifts, a standard tool in extremal combinatorics. Let $C \subseteq \{0, 1\}^n$ and let $i \in [n]$. We denote as S_i the down-shift operator on the i'th coordinate. Obtain the set $S_i(C) \subseteq \{0, 1\}^n$ from C as follows. Replace every $c \in C$ such that both (i) $c_i = 1$, and (ii) the i'th neighbor[1] of c is not in C with the i'th neighbor of c. Authors have referred to this operation as "compression", "switching", and "polarization'. Previous works that use down-shifts include [27, 20, 13, 22, 23, 33].

An important property of down-shifts is that they transform an arbitrary subset of $\{0, 1\}^n$ into a downward-closed set, without changing cardinality. Specifically, if

$$D = S_n(S_{n-1}(\cdots S_1(C)))$$

is the result of sequentially applying S_i on C for each i, then D is downward-closed. It is also convenient in this context to think of D as a family of subsets of $[n]$ rather than a set of boolean vectors via the natural correspondence between boolean vectors and sets.

We move on to explaining our results in more formality and detail.

[1] Vectors $u, v \in \{0, 1\}^n$ are *i'th-neighbors* if they differ in coordinate i and are the same elsewhere.

2 Our Results

We start with the result about linear program feasibility. We then state the bounds on the Hilbert function in terms of shattered sets and down-shifts. We show this leads to bounded-depth circuit lower bounds. Finally, we state two combinatorial applications.

2.1 Linear Program Feasibility

We formalize and prove a strong version of the statement "linear programming feasibility can not be decided by polynomial-sized, constant-depth circuits." Clearly, linear programming being P-complete [17] implies a version of this statement for specific linear programs representing functions previously known not to have efficient bounded-depth circuits. We prove a stronger version stating that any linear feasibility problem, in which the number of constraints is roughly twice the number of variables and the constraints are non-degenerate, cannot be decided by an efficient bounded-depth circuit. For a set of hyperplanes $\mathcal{H}$ in $\mathbb{R}^k$ we will define a boolean function $f_{\mathcal{H}}$. It takes orientations as inputs and outputs one if and only if a certain polytope is nonempty. In particular, we establish hardness of this problem even when the hyperplanes are fixed in advance and only the orientations are given as input.

We express linear program feasibility as a boolean function as follows. Specify an arrangement of m hyperplanes $\mathcal{H} = \{h_1, \dots, h_m\}$ with normal vectors $\vec{n}_i$ and translation scalars b_i as

$$h_i = \{\vec{x} : \langle \vec{n_i}, \vec{x} \rangle = b_i\}.$$

A sign pattern $s \in \{-1,1\}^m$ encodes the following linear programming feasibility problem:

Does there exist $\vec{x} \in \mathbb{R}^k$ *satisfying* $\mathsf{sign}(\langle x, \vec{n}_i \rangle - b_i) = s_i$ *for all* $i \in [m]$*?*

This corresponds to checking the feasibility of a linear program with m constraints and k variables. Define $f_{\mathcal{H}} : \{-1,1\}^m \to \{0,1\}$ as the boolean function such that $f_{\mathcal{H}}(s) = 1$ if and only if the linear program encoded by s is feasible.

As an example, consider the following arrangement in $\mathbb{R}^2$. The three hyperplanes

$$h_1 = \{(x_1, x_2) : 5x_1 + 3x_2 = 3\},$$
$$h_2 = \{(x_1, x_2) : 8x_1 - x_2 = 8\},$$
$$h_3 = \{(x_1, x_2) : 4x_1 - 5x_2 = 0\}$$

form an arrangement of three lines in the plane. The vector $s = (+1, -1, +1)$ encodes the system

$$5x_1 + 3x_2 > 3 \qquad (s(1) = +1)$$
$$8x_1 - x_2 < 8 \qquad (s(2) = -1)$$
$$4x_1 + 5x_2 > 0 \qquad (s(3) = +1)$$

In the example, the system encoded by $(+1, -1, +1)$ is not satisfiable (see Figure 1). For more background material on hyperplane arrangements and related results, see the books by Stanley [43] and Matoušek [30].

We prove the following theorem.

▶ **Theorem 6.** *Let $\mathcal{H}$ be an arrangement of $2k+1$ hyperplanes in $\mathbb{R}^k$ that are in general position. Any $AC^0[p]$ circuit, for a prime p, with depth d computing $f_{\mathcal{H}}$ requires $exp(\Omega(k^{1/2d}))$ gates.*

We prove Theorem 6 in Section 4, using the framework of Razborov [37] and Smolensky [41].

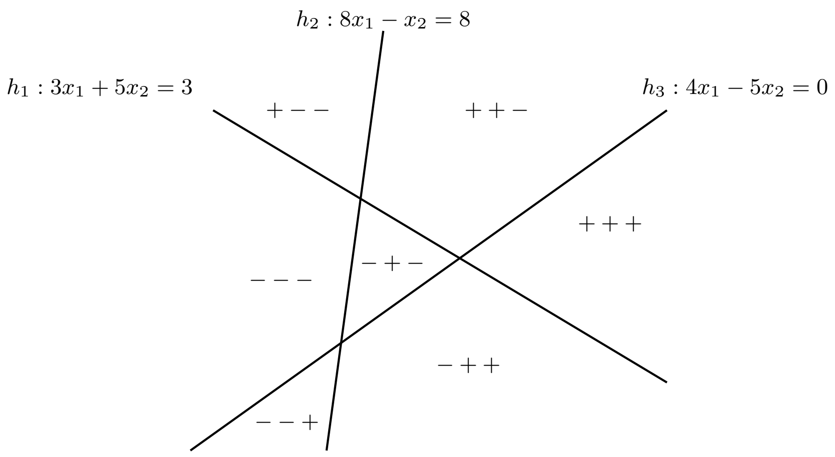


Figure 1 Three lines divide $\mathbb{R}^2$ into seven regions, each labeled by a feasible sign pattern.

Explicit Arrangements. The space of oriented hyperplanes is a rich and well-studied object. The books [30, 43] provide many facts and examples. The paper [2] and references therein give bounds on how many different boolean functions can be represented as $f_{\mathcal{H}}$ for some hyperplane arrangement $\mathcal{H}$.

General position hyperplane arrangements come from any $2k+1$ vectors in $\mathbb{R}^{k+1}$ such that every $k+1$ of them are linearly independent. For a vector $v \in \mathbb{R}^{k+1}$ the hyperplane has normal $(v_1, \ldots, v_k)$ and translation v_{k+1}. Explicit families of m vectors in $\mathbb{R}^d$ such that every d of them are independent are known for any m, d. For example, take the rows of an $m \times d$ Cauchy or Vandermonde matrix.

2.2 Hilbert Function Bounds

Our results are based on the following theorem.

▶ **Theorem 7.** *Any $C \subseteq \{0,1\}^n$ and any $d \in [n]$ satisfy the relationships*

$$|\{I \in \mathsf{sstr}(C) \ : \ |I| \le d\}| \ \le \ h_d(C, \mathbb{F}) \ \le \ |\{I \in \mathsf{str}(C) \ : \ |I| \le d\}|$$

and

$$\max\{|I| : I \in \mathsf{sstr}(C)\} \ \le \ \mathsf{intdeg}(C, \mathbb{F}) \ \le \ \max\{|I| : I \in \mathsf{str}(C)\}.$$

The upper bounds on interpolation degree are not new. Smolensky [42] derives the Sauer-Perles-Shelah Lemma using very similar polynomial-based arguments. The upper bounds on interpolation degree in terms of VC dimension also appear implicitly in the work of Frankl and Pach [7] and explicitly in Gurvits [24]. Our technical contributions center around the lower bounds and the applications. We prove Theorem 7 in Section 3.1.

We strengthen the lower bound on the Hilbert function in Theorem 7 using down-shifts.

▶ **Theorem 8.** *Let $C \subseteq \{0,1\}^n$ and let $D = S_n(S_{n-1}(\ldots S_1(C)))$. Then*

$$|\{I \in D \ : \ |I| \le d\}| \ \le \ h_d(C, \mathbb{F}) \quad \text{and} \quad \max\{|I| : I \in D\} \ \le \ \mathsf{intdeg}(C, \mathbb{F}).$$

In Section 3.2 we prove this theorem and show that the parity function provides a tight example over $GF(2)$. We also discuss how Theorem 8 implies the lower bound in Theorem 7.

2.3 Low-Degree Polynomial Approximations

Classic results in bounded-depth circuit complexity reduce the task of proving circuit lower bounds to showing that a boolean function has no low-degree approximation [37, 41, 6]. Smolensky shows in [40] how to express all known degree lower bounds in terms of the Hilbert function. For a boolean function f consider the set $S = f^{-1}(1)$ as a subset of the boolean cube. Smolensky shows that if $h_d(S, \mathbb{F})$ is large, then f is hard to approximate.

▶ **Theorem 9** ([40]). *Consider $f : \{0,1\}^n \to \{0,1\}$ and $p : \{0,1\}^n \to \mathbb{F}$. Define $S = f^{-1}(1)$ and fix $d = \lfloor (n - \deg_{\mathbb{F}}(p) - 1)/2 \rfloor$. Then,*

$$\Pr_x[p(x) \neq f(x)] \geq \frac{2 \cdot h_d(S, \mathbb{F}) - |S|}{2^n},$$

where x is uniform over $\{0,1\}^n$.

Theorem 7 implies the following corollary in terms of strongly shattered sets.

▶ **Corollary 10.** *Assume n is odd. Consider $f : \{0,1\}^n \to \{0,1\}$. If $|f^{-1}(1)| = 2^{n-1}$ and $\mathsf{sstr}(f^{-1}(1)) = \{I \subseteq [n] : |I| \leq \frac{n-1}{2}\}$, then for any polynomial $p \in \mathbb{F}[x_1, \ldots, x_n]$ we have*

$$\Pr_x[p(x) \neq f(x)] \geq \frac{1}{2} - \frac{10 \deg_{\mathbb{F}}(p)}{\sqrt{n}},$$

where x is uniform over $\{0,1\}^n$.

Proof. Since $\mathsf{sstr}(f^{-1}(1)) = \{I \subseteq [n] : |I| \leq \frac{n-1}{2}\}$, we have that

$$|\{I \in \mathsf{sstr}(C) \ : \ |I| \leq d\}| = \sum_{j=0}^{d} \binom{n}{j}$$

for all $d = 0, 1, \ldots, (n-1)/2$. Theorem 7 implies that $h_d(f^{-1}(1), \mathbb{F}) = \sum_{j=0}^{d} \binom{n}{j}$ as well. Plugging these into Theorem 9, along with $|f^{-1}(1)| = 2^{n-1}$, gives the corollary. ◀

Bernasconi and Egidi [11] thoroughly characterize the Hilbert function for symmetric sets and prove that any nearly-balanced, symmetric boolean function is hard to approximate. They leave as an open question deriving bounds for non-symmetric sets. Our connection to VC theory leads to new families of functions satisfying the conditions of Corollary 10. Many of these functions, such as the linear programming feasibility functions from Section 2.1, are non-monotone and non-symmetric. As a final remark, recent work shows that Smolensky's lower bound (and thus our result) extends to nonclassical polynomials [12].

2.4 The Sandwich Theorem

The following relationship which is a generalization of the Sauer-Perles-Shelah Lemma was discovered several times and independently [14, 36, 18, 5].

▶ **Theorem 11** (Sandwich Theorem). *For any $C \subseteq \{0,1\}^n$ we have $|\mathsf{sstr}(C)| \leq |C| \leq |\mathsf{str}(C)|$.*

Since $|\mathsf{str}(C)| \leq \sum_{i=0}^{\mathsf{VC}(C)} \binom{n}{i}$, this implies the Sauer-Perles-Shelah Lemma.

Theorem 7 yields a new algebraic proof of the Sandwich Theorem. Indeed, this follows from examining the case of $d = n$ and observing that $h_n(C, \mathbb{F}) = |C|$.

The Sandwich Theorem is tight in the sense that there are sets that achieve equality in both of its inequalities[2]. These sets are calles *shattering extremal sets*. For example, downward-closed sets are shattering extremal. Shattering extremal sets have been rediscovered and studied in different contexts [29, 15, 14, 18, 8, 33, 28, 32, 35]. In our context, Corollary 10 says that shattering extremal sets S of size $|S| = 2^{n-1}$ and VC dimension $\frac{n-1}{2}$ correspond to boolean functions that cannot be approximated by low-degree polynomials.

2.5 Downward-closed Sets and Chvátal's Conjecture

Downward-closed sets have a well-studied, rich combinatorial structure. A theorem of Berge [10] implies the following fact. For any downward-closed set D, there is a bijection $\pi : D \to D$ such that $a \cap \pi(a) = \emptyset$, for all $a \in D$. We refer to such a bijection as a *pseudo-complement*. We prove the following generalization of the existence of a pseudo-complement.

▶ **Theorem 12.** *Let D be any downward-closed set. Fix any mapping $\phi : D \to D$ with the property that $\phi(a) \subseteq a$ for all $a \in D$. Then there exists a bijection $\pi : D \to D$ satisfying the condition that $a \cap \pi(a) = \phi(a)$ for all $a \in D$.*

Note that choosing $\phi(a) = \emptyset$ for all a implies the existence of a pseudo-complement.

In topology, downward-closed sets correspond to simplicial complexes. We think of the ϕ as prescribing intersections. For simplicial complexes, this corresponds to prescribing that complexes intersect in certain faces. We prove Theorem 12 in Section 5. Our proof proceeds by proving that a certain matrix is invertible. A non-zero determinant implies that the matrix contains a permutation matrix that yields the desired bijection.

We next discuss the result by Berge for the existence of pseudo-complements and its connections with Chvátal's conjecture in extremal combinatorics [16]. Berge's result about pseudo-complements follows from the following stronger theorem that he proved.

▶ **Theorem 13** ([10]). *If D is a downward-closed set, then either D or $D \setminus \emptyset$ can be partitioned into pairs of disjoint sets.*

We need two definitions to explain Berge's motivation. A family B of subsets of $[n]$ is called a *star* if there is an element $x \in [n]$ such that $x \in b$ for all $b \in B$. It is called an *intersecting family* if every pair of sets in B intersects. Chvátal's conjecture is the following.

▶ **Conjecture 14** (Chvátal's conjecture). *If D is a downward closed set, then the cardinality of the largest star in D is equal to the cardinality of the largest intersecting family in D.*

This conjecture remains open, aside from partial results, such as the following corollary of Berge's theorem.

▶ **Corollary 15.** *In a downward-closed set D, any intersecting family has cardinality at most $|D|/2$.*

We contrast Berge's theorem and our Theorem 12. Berge's pair decomposition induces a permutation π such that $\pi(\pi(a)) = a$, whereas a permutation decomposes D into disjoint cycles with unspecified lengths. Many people have observed that the above corollary only needs the pseudo-complement result, instead of the stronger statement in Berge's theorem [4]. Indeed, consider each disjoint cycle in the guaranteed permutation, and note that at most half of the sets in the cycle may mutually intersect. Therefore, our Theorem 12 implies the above corollary.

[2] In fact, it is well known (see for example [33]) that any set achieving equality in one of the inequalities, also achieves equality in the other.

3 The Hilbert Function for Subsets of the Boolean Cube

We prove upper and lower bounds on the Hilbert function. First, we prove the bounds in Theorem 7 involving the shattered and the strongly shattered sets. Then, we prove the bounds in Theorem 8 using shifting. Finally we consider an example of applying these bounds to analyze the Hilbert function of the parity function.

3.1 Bounding the Hilbert Function Using Shattered Sets

The high-level idea of the proof of Theorem 7 is to define a vector space V with $\dim(V) = |C|$ and prove that $|\mathsf{sstr}(C)| \le \dim(V) \le |\mathsf{str}(C)|$. We sandwich the dimension $\dim(V)$ by finding a linearly independent set of size $|\mathsf{sstr}(C)|$ and a spanning set of size $|\mathsf{str}(C)|$.

We analyze the $|C|$-dimensional vector space $\{f : C \to \mathbb{F}\}$. Evaluation on C induces a natural mapping from $P \in \mathbb{F}[x_1, \dots, x_n]$ to the restriction $P|_C \in \{f : C \to \mathbb{F}\}$. The following lemma provides the desired sets of spanning monomials and linearly independent monomials.

▶ **Lemma 16.** *For all fields $\mathbb{F}$ and sets $C \subseteq \{0,1\}^n$ the following two facts hold.*
1. *The monomials $\prod_{i\in I} x_i$ for $I \in \mathsf{str}(C)$ span $\{f : C \to \mathbb{F}\}$.*
2. *The monomials $\prod_{i\in I} x_i$ for $I \in \mathsf{sstr}(C)$ are linearly independent in $\{f : C \to \mathbb{F}\}$.*

Proof. For $I \subseteq [n]$, let x_Y denote the monomial $x_I = \prod_{i\in I} x_i$. For the first item, we express every $f : C \to \mathbb{F}$ as a linear combination of monomials $(x_I)|_C$ where $I \in \mathsf{str}(C)$. It suffices to express the monomials $(x_I)|_C$ for all $I \subseteq [n]$. We prove this by induction. For the base case, if $I \in \mathsf{str}(C)$, we are done. Otherwise, I is not shattered by C and there exists $s \in \{0,1\}^I$ such that for all $c \in C$, we have $c|_I \ne s$. Consider

$$P = \prod_{i\in I} (x_i - (1 - s_i)) .$$

Note that $P(c) = 0$ for all $c \in C$ and hence $P|_C = 0|_C$. Specifically, by expanding the product $\prod_{i\in I} (x_i - (1 - s_i))$ we see

$$0|_C = P = (x_I)|_C + (Q)|_C,$$

where the degree of Q is smaller than $|I|$. By induction, we can write Q as a combination of $x_{I'}$ for $I' \in \mathsf{str}(C)$. Since $(x_I)|_C = (-Q)|_C$ we get that x_I is in this span as well.

We now prove the second item. Consider a linear combination

$$P = \sum_{I\in \mathsf{sstr}(C)} \alpha_I x_I$$

such that not all α_I are zero. We want to show that there is $c \in C$ such that $P(c) \ne 0$. Let $Z \in \mathsf{sstr}(C)$ be a maximal set such that $\alpha_Z \ne 0$. Since Z is strongly shattered by C, there is some $\bar{s} : ([n] \setminus Z) \to \{0,1\}$ such that all extensions of it in $\{0,1\}^n$ are in C. Let $Q(x_i)_{i\in Z}$ be the polynomial obtained by plugging in the values of $\bar{s}$ in the variables of $([n] \setminus Z)$. By maximality of Z it follows that the coefficient of x_Z in Q is $\alpha_Z \ne 0$, and so Q is not the 0 polynomial. Therefore there is $s \in \{0,1\}^Z$ such such that $Q(s) \ne 0$. Pick $c \in C$ such that

$$c_i = \begin{cases} s_i & i \in Z, \\ \bar{s}_i & i \in ([n] \setminus Z). \end{cases}$$

It follows that $P(c) = Q(s) \ne 0$, which finishes the proof. ◀

We use this lemma to prove bounds on the Hilbert function and interpolation degree.

Proof of Theorem 7. For the upper bound on $h_d(C,\mathbb{F})$, the above proof shows how to express all monomials of degree d using monomials of equal or smaller degree. For the lower bound on $h_d(C,\mathbb{F})$, linear independence still holds after restricting set size.

The upper bound on $\mathsf{intdeg}(C,\mathbb{F})$ is immediate. For the lower bound on $\mathsf{intdeg}(C,\mathbb{F})$, since $\mathsf{sstr}(C)$ is downward-closed, the linear independence of the monomials in $\mathsf{sstr}(C)$ implies any maximal degree monomial in $\{(x_I)|_C : I \in \mathsf{sstr}(C)\}$ cannot be expressed solely by lower degree monomials. ◀

3.2 Down-shifts, Downward-closed Bases, and the Hilbert Function

We prove Theorem 8. We also use the theorem to analyze the Hilbert function for the parity function. Theorem 8 is a direct corollary of the following theorem.

▶ **Theorem 17.** *Let $C \subseteq \{0,1\}^n$ and let $D = S_n(S_{n-1}(\ldots S_1(C)))$. Then the set of monomials $\{\prod_{i\in I} x_i : I \in D\}$ is a basis for the vector space of functions $\{f : C \to \mathbb{F}\}$.*

A theorem, equivalent in content, but expressed with respect to Gröbner bases, is proved in [31]. For completeness we include an elementary proof in the full version of this paper.

The lower bound given in Theorem 8 subsumes the lower bound in Theorem 7. This is a direct corollary of the following simple lemma.

▶ **Lemma 18.** *Let $C \subseteq \{0,1\}^n$ and let $D = S_n(S_{n-1}(\ldots S_1(C)))$. We have that $\mathsf{sstr}(C) \subseteq D$, where we associate $\{0,1\}^n$ with subsets of $[n]$ in the natural way.*

Proof. Since D is downward-closed, it follows that it is shattering extremal and therefore $\mathsf{sstr}(D) = D$. So, it is enough to show that $\mathsf{sstr}(C) \subseteq \mathsf{sstr}(D)$. To this end, it suffices to show that for every class C', $\mathsf{sstr}(C') \subseteq \mathsf{sstr}(S_i(C'))$. Let $I \in \mathsf{sstr}(C')$. Therefore C' contains a subcube B in coordinates I. During the down-shift, B is either shifted or stays in place, but in any case also $S_i(C')$ contains a subcube in coordinates I and therefore $I \in \mathsf{sstr}(S_i(C'))$. ◀

The Hilbert Function of Parity. A simple example which demonstrates an application of Theorem 8 is the parity function. Let P denote the set of all vectors of even hamming weight. Notice that P does not contain subcubes other than $\emptyset$. Therefore, $\mathsf{sstr}(P) = \{\emptyset\}$. As a consequence, the lower bound on the Hilbert function in Theorem 7 reveals no information in this case. In contrast, shifting gives a better bound. If we down-shift P, say on the first coordinate, we get the set $S_1(P) = D = \{v : v_1 = 0\}$. Therefore, as D is downward closed, shifting it on other coordinates does not change it. Thus, $S_n(S_{n-1}(\ldots S_1(P))) = D$. By Theorem 8 we have that $h_d(P,\mathbb{F}) \geq \binom{n-1}{\leq d} = \binom{n-1}{d} + \binom{n-1}{d-1} + \ldots + \binom{n-1}{0}$.

This lower bound is tight when the field has characteristic two and $d \leq n/2$. It suffices to show every polynomial q of degree at most d can be expressed by a polynomial of degree at most d that does not depend on x_1. Therefore the $\binom{n-1}{\leq d}$ multilinear monomials that do not depend on x_1 span the space of degree at most d polynomials with domain P. Note that $(x_1 + \ldots + x_n)|_P = 0$, and therefore every appearance of x_1 can be replaced by $x_2 + \ldots + x_n$. This transforms q to a polynomial that does not depend on x_1 without changing the represented function.

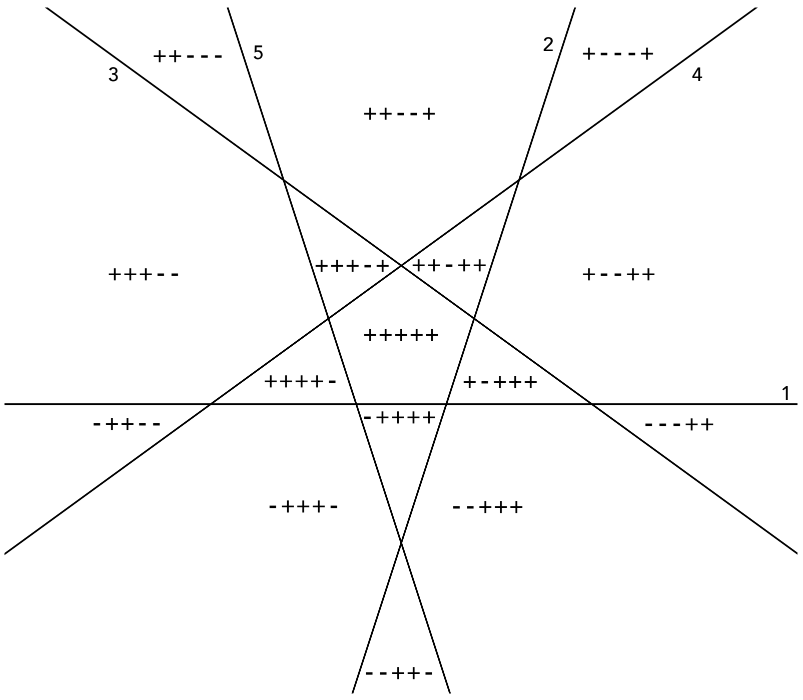


Figure 2 Five hyperplanes divide $\mathbb{R}^2$ into 16 cells. Cell labels in $\{-,+\}^5$ correspond to oriented hyperplane feasibility. Notice that every two coordinates are strongly shattered, but no three coordinates are shattered. This provides a proof-by-picture of Proposition 20 for $m = 5$ and $d = 2$.

4 Linear Programming and Low-degree Polynomial Approximations

We now prove Theorem 6. By the Razborov-Smolensky framework, it suffices to prove that $f_{\mathcal{H}}$ cannot be approximated by a low-degree polynomial over any field.[3]

▶ **Theorem 19.** *Let $\mathcal{H}$ be an arrangement of $2k+1$ hyperplanes in $\mathbb{R}^k$ that are in general position. For any any polynomial $p \in \mathbb{F}[x_1, \ldots, x_{2k+1}]$ we have*

$$\Pr_s[p(s) \neq f_{\mathcal{H}}(s)] \geq \frac{1}{2} - \frac{10 \deg_{\mathbb{F}}(p)}{\sqrt{2k+1}},$$

where s is uniform over $\{-1,1\}^{2k+1}$.

The proof of Theorem 19 proceeds via a reduction to Corollary 10. Let

$$S_{\mathcal{H}} = \{s \in \{-1,1\}^n : \ f_{\mathcal{H}}(s) = 1\}.$$

To apply Corollary 10 on $f_{\mathcal{H}}$ we will show $|S_{\mathcal{H}}| = 2^{2k}$ and $\mathsf{sstr}(S_{\mathcal{H}}) = \{I \subseteq [2k+1] : |I| \leq k\}$. We establish this by the following proposition. The facts we need about hyperplane arrangements follow from standard arguments [21, 43]. For intuition about the following proposition, see Figure 2 for a pictorial proof in $\mathbb{R}^2$.

▶ **Proposition 20.** *For any m hyperplanes $\mathcal{H}$ in $\mathbb{R}^d$ in general position*

$$\mathsf{sstr}(S_{\mathcal{H}}) = \mathsf{str}(S_{\mathcal{H}}) = \{I \subseteq [m] : |I| \leq d\}.$$

[3] We state the following theorem for $\{-1,1\}$ inputs to $f_{\mathcal{H}}$. This only makes sense for fields containing these elements. When $\mathbb{F} = \mathbb{F}_2$ simply replace $\{-1,1\}$ with $\{0,1\}$ in the definition of $f_{\mathcal{H}}$.

Proof. In the full version of the paper we include two lemmas that characterize the shattered and strongly shattered sets of $S_{\mathcal{H}}$ when $\mathcal{H}$ is in general position. The first lemma shows $\mathsf{str}(S_{\mathcal{H}}) \subseteq \{I \subseteq [m] : |I| \leq d\}$. The second lemma shows $\{I \subseteq [m] : |I| \leq d\} \subseteq \mathsf{sstr}(S_{\mathcal{H}})$. Since $\mathsf{sstr}(S_{\mathcal{H}}) \subseteq \mathsf{str}(S_{\mathcal{H}})$ these two lemmas combine to finish the proof. ◀

Proposition 20 implies Theorem 6. The equality $\mathsf{sstr}(S_{\mathcal{H}}) = \mathsf{str}(S_{\mathcal{H}})$ along with the Sandwich Theorem implies that $|S_{\mathcal{H}}| = |\mathsf{sstr}(S_{\mathcal{H}})|$. Let k be the ambient dimension in Theorem 6. The above proposition for $m = 2k+1$ and $d = k$ gives $|S_{\mathcal{H}}| = 2^{2k}$ and also $\mathsf{sstr}(S_{\mathcal{H}}) = \{I \subseteq [2k+1] : |Y| \leq k\}$. Thus $f_{\mathcal{H}}$ satisfies the premises of Corollary 10, and Theorem 6 follows.

5 Downward-closed Sets and Prescribed Intersections

We prove Theorem 12. Let $D \subseteq \{0,1\}^n$ be a downward-closed set. Fix $\phi : D \to D$ with the property that $\phi(a) \subseteq a$ for all $a \in D$. We will show that there exists a bijection $\pi : D \to D$ satisfying the condition that $a \cap \pi(a) = \phi(a)$ for all $a \in D$. We first prove two lemmas about the function space $\{f : D \to GF(2)\}$ and then use these to prove the existence of π. The first lemma holds for all subsets of the boolean cube.

▶ **Lemma 21.** *Let $C \subseteq \{0,1\}^n$ be a subset of the boolean hypercube. The monomials*

$$\prod_{i\in a} x_i \text{ for } a \in C$$

form a basis for $\{f : C \to GF(2)\}$.

Proof. We proceed using induction on $|C|$. When $C = \{a\}$ for $a \in \{0,1\}^n$ the function space has dimension one and the monomial $\prod_{i\in a} x_i$ represent the constant "1" function in this space, which spans it. Let $z \in C$ denote a maximal Hamming weight element in C. Notice $\prod_{i\in z} x_i$ is an indicator function in $\{f : C \to GF(2)\}$ for the input z. By the inductive hypothesis on $(C \setminus \{z\})$, we know the set of monomials $\prod_{i\in a} x_i$ for $a \in (C \setminus \{z\})$ form a basis for $\{f : (C \setminus \{z\}) \to GF(2)\}$. Since $\prod_{i\in z} x_i$ is an indicator function, we may add it to the basis for $\{f : (C \setminus \{z\}) \to GF(2)\}$ and achieve a basis for $\{f : C \to GF(2)\}$. ◀

We remark that if C is downward-closed, then it is shattering extremal, and the above lemma is a corollary of the Sandwich theorem. We prove the following stronger claim as well.

▶ **Lemma 22.** *Let $D \subseteq \{0,1\}^n$ be a downward-closed set. Fix any mapping $\phi : D \to D$ with the property that $\phi(a) \subseteq a$ for all $a \in D$. The functions*

$$\prod_{i\in\phi(a)} x_i \prod_{i\in a\setminus\phi(a)} (1 + x_i)$$

for $a \in D$ form a basis for $\{f : D \to GF(2)\}$.

Proof. Let $\mathcal{B}$ denote the set of polynomials that we wish to show is a basis. Since the cardinality of $\mathcal{B}$ is $|D|$ it is enough to show that it is a spanning set. By Lemma 21, it is enough to show that every monomial of the form $\prod_{i\in a} x_i$ for $a \in D$ can be expressed as a linear combination of polynomials in $\mathcal{B}$. We proceed by induction on the size of a. The case of $a = \emptyset$ is trivial. For the induction step, let $a \in D$ be non-empty. Expand the polynomial

$$\prod_{i\in\phi(a)} x_i \prod_{i\in a\setminus\phi(a)} (1 + x_i) = \left(\prod_{i\in a} x_i\right) + r,$$

where r is a linear combination of monomials $\prod_{i\in b} x_i$ for $b \subseteq a$ and $b \neq a$. Since D is downward-closed, by induction hypothesis r is in the span of $\mathcal{B}$. Thus,

$$\prod_{i\in a} x_i = \left(\prod_{i\in\phi(z)} x_i \prod_{i\in a\setminus\phi(a)} (1+x_i) \right) + r$$

is also in the span of $\mathcal{B}$, and we are done. ◀

Proof of Theorem 12. We show there exists a bijection $\pi : D \to D$ such that $a\cap\pi(a) = \phi(a)$ for all $a \in D$, for the given map ϕ. Consider the $|D|\times|D|$ boolean matrix M defined as follows. Index the rows and columns both by D, and define the element in location $(a, b) \in D \times D$ to be one if and only if $a \cap b = \phi(a)$. We claim that M is nonsingular. Indeed, the rows of M correspond to the functions in Lemma 22. Since they form a basis, the row space of M is $|D|$-dimensional. This implies the determinant of M is nonzero. There must exist a permutation $\pi : [n] \to [n]$ such that $\prod_{i=1}^{|D|} M_{i,\pi(i)} = 1$. By the definition of M, we found the bijection π we were looking for. ◀

6 Conclusion

We exhibited a connection between algebra and combinatorics. We provided a general way to lower bound the Hilbert function. We showed a new family of functions cannot be approximated by low-degree polynomials. We provided a polynomial method proof of the Sandwich theorem and for a new theorem about prescribed intersections.

6.1 Open Directions

Our work suggests that the interpolation degree is a useful complexity measure on subsets of the boolean hypercube. Therefore, an open direction is to better understand the structure of sets with low interpolation degree. As noted by Remscrim [38], one can equivalently define interpolation degree in terms of the rank of a certain incidence matrix. The matrix corresponds to the monomials in our Lemma 21 with a cut-off on the degree. For the case of interpolation degree one, this characterization is particularly simple.

▶ **Proposition 23.** *A set $C \subseteq \{0,1\}^n$ has $\mathsf{intdeg}(C, \mathbb{F}) = 1$ if and only if the boolean vectors corresponding to C are affinely independent in $\mathbb{F}^n$.*

We are curious if other properties of the vectors in C correspond to implications for the interpolation degree. Even for interpolation degree two, the algebraic/matrix description becomes more opaque and less intuitive than the characterization in the above proposition. Since $\mathsf{intdeg}(C, \mathbb{F}) \leq \mathsf{VC}(V)$, any combinatorial characterization may also shed new light on the structure of sets with VC dimension two, for which our understanding is lacking [3, 34].

Acknowledgements. We thank Paul Beame, Eli Ben-Sasson, Yuval Filmus, Ariel Gabizon, Sivaramakrishnan Natarajan Ramamoorthy, Anup Rao, Mert Sağlam, and Amir Yehudayoff for helping us improve the presentation of the paper. We also thank the anonymous reviewers for their comments.

References

1 Noga Alon. Combinatorial Nullstellensatz. *Combinatorics Probability and Computing*, 8(1):7–30, 1999.

2 Noga Alon, Peter Frankl, and Vojtech Rödl. Geometrical Realization of Set Systems and Probabilistic communication Complexity. In *FOCS*, pages 277–280. IEEE, 1985.

3 Noga Alon, Shay Moran, and Amir Yehudayoff. Sign rank, VC dimension and spectral gaps. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:135, 2014. URL: `http://eccc.hpi-web.de/report/2014/135`.

4 Ian Anderson. Combinatorics of Finite sets. *Bull. Amer. Math. Soc.*, 1988.

5 Richard P. Anstee, Lajos Rónyai, and Attila Sali. Shattering news. *Graphs and Combinatorics*, 18(1):59–73, 2002. `doi:10.1007/s003730200003`.

6 J. Aspnes, R. Beigel, M. Furst, and S. Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994. `doi:10.1007/BF01215346`.

7 L Babai and P Frankl. *Linear Algebra Methods in Combinatorics*. University of Chicago, 1992.

8 Hans-Jürgen Bandelt, Victor Chepoi, Andreas W. M. Dress, and Jack H. Koolen. Combinatorics of lopsided sets. *Eur. J. Comb.*, 27(5):669–689, 2006.

9 Richard Beigel. The Polynomial Method in Circuit Complexity. In *In Proceedings of the 8th IEEE Structure in Complexity Theory Conference*. Citeseer, 1993.

10 C Berge. A Theorem Related to the Chvátal conjecture. In *Proceedings, 5th British Combinatorial Conference*, 1976.

11 Anna Bernasconi and Lavinia Egidi. Hilbert Function and Complexity Lower Bounds for Symmetric Boolean Functions. *Information and Computation*, 153(1):1–25, 1999.

12 Abhishek Bhowmick and Shachar Lovett. Nonclassical Polynomials as a Barrier to Polynomial Lower Bounds. In *30th Conference on Computational Complexity*, page 72, 2015.

13 Béla Bollobás and Imre Leader. Sums in the grid. *Discrete Mathematics*, 162(1-3):31–48, 1996. `doi:10.1016/S0012-365X(96)00303-2`.

14 Béla Bollobás and A. J. Radcliffe. Defect sauer results. *J. Comb. Theory, Ser. A*, 72(2):189–208, 1995.

15 Béla Bollobás, A. J. Radcliffe, and Leader I. Reverse kleitman inequalities. *Proc. London Math. Soc., Ser. A*, (3) 58:153–168, 1989.

16 Vašek Chvátal and Donald Knuth. *Selected Combinatorial Research Problems*. Stanford University, 1972.

17 David P Dobkin and Steven P Reiss. The Complexity of Linear Programming. *Theoretical Computer Science*, 11:1–18, 1980.

18 A.W.M. Dress. Towards a theory of holistic clustering. *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 37 Amer. Math. Soc.:271–289, 1997.

19 Zeev Dvir. On the size of Kakeya sets in finite fields. *Journal of the American Mathematical Society*, 22(4):1093–1097, 2009.

20 Per Enflo. On the nonexistence of uniform homeomorphisms betweenl p-spaces. *Arkiv för Matematik*, 8(2):103–105, 1970. `doi:10.1007/BF02589549`.

21 Bernd Gärtner and Emo Welzl. Vapnik-Chervonenkis dimension and (pseudo-) hyperplane arrangements. *Discrete & Computational Geometry*, 12(1):399–432, 1994.

22 Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000. `doi:10.1007/s004930070011`.

23 Ben Joseph Green and Terence Tao. Freiman's theorem in finite fields via extremal set theory. *Combinatorics, Probability &amp; Computing*, 18(3):335–355, 2009. `doi:10.1017/S0963548309009821`.

24 Leonid Gurvits. Linear algebraic proofs of VC-dimension based inequalities. In *Computational Learning Theory*, pages 238–250. Springer, 1997.

25 Larry Guth and Nets Hawk Katz. On the Erdos Distinct Distances Problem in the Plane. *Annals of Mathematics*, 181(1):155–190, 2015.

26 Stasys Jukna. *Extremal Combinatorics: with Applications in Computer Science.* Springer Science & Business Media, 2011.

27 Daniel J. Kleitman. Families of non-disjoint subsets. *Journal of Combinatorial Theory*, 1(1):153–155, 1966. doi:10.1016/S0021-9800(66)80012-1.

28 László Kozma and Shay Moran. Shattering, graph orientations, and connectivity. *Electr. J. Comb.*, 20(3):P44, 2013. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v20i3p44.

29 J. Lawrence. Lopsided sets and orthant-intersection by convex sets. *Pac. J. Math.*, 104(1):155–173, 1983.

30 Jiří Matoušek. *Lectures on discrete geometry*, volume 212. Springer New York, 2002.

31 Tamás Mészáros. S-extremal set systems and Gröbner Bases. Master's thesis, Budapest University of Technology and Economics, 2009.

32 Tamás Mészáros and Lajos Rónyai. Shattering-extremal set systems of VC dimension at most 2. *Electr. J. Comb.*, 21(4):P4.30, 2014. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v21i4p30.

33 S. Moran. Shattering Extremal Systems. Master's thesis, Saarland University, Saarbrücken, Germany, 2012.

34 Shay Moran, Amir Shpilka, Avi Wigderson, and Amir Yehudayoff. Teaching and compressing for low vc-dimension. In *FOCS*, 2015. URL: http://arxiv.org/abs/1502.06187.

35 Shay Moran and Manfred K. Warmuth. Labeled compression schemes for extremal classes. *CoRR*, abs/1506.00165, 2015. URL: http://arxiv.org/abs/1506.00165.

36 A. Pajor. Sous-espaces l_1^n des espaces de banach. *Travaux en Cours. Hermann, Paris*, 1985.

37 Alexander A Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes*, 41(4):333–338, 1987.

38 Zachary Remscrim. The Hilbert Function, Algebraic Extractors, and Recursive Fourier Sampling. *ECCC Technical Report TR 16-020*, 2016.

39 Micha Sharir, Adam Sheffer, and Joshua Zahl. Improved Bounds for Incidences Between Points and Circles. *Combinatorics, Probability and Computing*, 24(03):490–520, 2015.

40 R. Smolensky. On Representations by Low-degree Polynomials. In *FOCS*, 1993.

41 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82. ACM, 1987.

42 Roman Smolensky. Well-Known Bound for the VC-Dimension Made Easy. *Computational Complexity*, 6(4):299–300, 1997. doi:10.1007/BF01270383.

43 Richard P Stanley. Introduction to Hyperplane Arrangements. *Lecture notes, IAS/Park City Mathematics Institute*, 2004.

44 Terence Tao. Algebraic Combinatorial Geometry: the Polynomial Method in Arithmetic Combinatorics, Incidence Combinatorics, and Number Theory. *EMS Surveys in Mathematical Sciences*, 1(1):1–46, 2014.

Optimal Sparsification for Some Binary CSPs Using Low-Degree Polynomials*

Bart M. P. Jansen[1] **and Astrid Pieterse**[2]

1 **Eindhoven University of Technology, Eindhoven, The Netherlands**
`b.m.p.jansen@tue.nl`

2 **Eindhoven University of Technology, Eindhoven, The Netherlands**
`a.pieterse@tue.nl`

Abstract

This paper analyzes to what extent it is possible to efficiently reduce the number of clauses in NP-hard satisfiability problems, without changing the answer. Upper and lower bounds are established using the concept of kernelization. Existing results show that if $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, no efficient preprocessing algorithm can reduce n-variable instances of CNF-SAT with d literals per clause, to equivalent instances with $\mathcal{O}(n^{d-\varepsilon})$ bits for any $\varepsilon > 0$. For the Not-All-Equal SAT problem, a compression to size $\widetilde{\mathcal{O}}(n^{d-1})$ exists. We put these results in a common framework by analyzing the compressibility of binary CSPs. We characterize constraint types based on the minimum degree of multivariate polynomials whose roots correspond to the satisfying assignments, obtaining (nearly) matching upper and lower bounds in several settings. Our lower bounds show that not just the number of constraints, but also the encoding size of individual constraints plays an important role. For example, for Exact Satisfiability with unbounded clause length it is possible to efficiently reduce the number of constraints to $n+1$, yet no polynomial-time algorithm can reduce to an equivalent instance with $\mathcal{O}(n^{2-\varepsilon})$ bits for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic

Keywords and phrases constraint satisfaction problem, sparsification, satisfiability, kernelization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.71

1 Introduction

The goal of sparsification is to make an object such as a graph or logical structure less dense, without changing the outcome of a computational task of interest. Sparsification can be used to speed up the solution of NP-hard problems, by sparsifying a problem instance before solving it. The notion of kernelization, originating in the field of parameterized complexity [8, 12, 13], facilitates a rigorous study of polynomial-time preprocessing for NP-hard problems and can be used to reason about (the impossibility of) sparsification. Over the last few years, our understanding of the power of polynomial-time data reduction has increased tremendously, as documented in recent surveys [4, 16, 23, 26]. By studying the kernelization complexity of a graph problem parameterized by the number of vertices, or of a logic problem parameterized by the number of variables, we can analyze its potential for sparsification.

* This work was supported by NWO Veni grant "Frontiers in Parameterized Preprocessing" and NWO Gravitation grant "Networks". This work was done in part while the first author was visiting the Simons Institute for the Theory of Computing.

The vast majority of the currently known results in this direction are negative [10, 18, 19, 20], stating that no nontrivial sparsification is possible under plausible complexity-theoretic assumptions. For example, Dell and van Melkebeek [10] obtained such a result for CNF-Satisfiability with clauses of size at most d (d-cnf-sat), for each fixed $d \geq 3$. Assuming $\mathsf{NP} \not\subseteq \mathsf{coNP/poly}$, there is no polynomial-time algorithm that compresses any n-variable instance of d-cnf-sat to an equivalent instance with $\mathcal{O}(n^{d-\varepsilon})$ bits for $\varepsilon > 0$. Since there are $\mathcal{O}(n^d)$ possible clauses of size at most d over n variables, the trivial compression scheme that outputs a bitstring of length $\mathcal{O}(n^d)$, denoting for each possible clause whether it occurs in the instance or not, is optimal up to $n^{o(1)}$ factors.

A problem for which nontrivial polynomial-time sparsification *is* possible was recently discovered by the current authors [20]. Any n-variable instance of the Not-All-Equal CNF-Satisfiability problem with clauses of size at most d (d-nae-sat) can efficiently be compressed to an equivalent instance with $\mathcal{O}(n^{d-1})$ clauses, which can be encoded in $\mathcal{O}(n^{d-1} \log n)$ bits. The preprocessing algorithm is based on a linear-algebraic lemma by Lovász [27] to identify clauses that are implied by others, allowing a reduction from $\Theta(n^d)$ clauses to $\mathcal{O}(n^{d-1})$. This sparsification for d-nae-sat forms the starting point for this work. Since d-cnf-sat and d-nae-sat can both be seen as constraint satisfaction problems (CSPs) with a binary domain, it is natural to ask whether the positive results for d-nae-sat extend to other binary CSPs. The difference between d-cnf-sat and d-nae-sat shows that the type of constraints that one allows, affects the compressibility of the resulting CSP. The goal of this paper is to understand how the optimal compression size for a binary CSP depends on the type of legal constraints, with the aim of obtaining matching upper and lower bounds.

Before presenting our results, we give an example to illustrate our methods. Consider the NP-complete Exact d-CNF-Satisfiability (Exact d-sat) problem, which asks whether there is a truth assignment that satisfies *exactly one* literal in each clause; the clauses have size at most d. While there are $\Theta(n^d)$ different clauses that can occur in an instance with n variables, the exact nature of the problem makes it possible to reduce any instance to an equivalent one with $n + 1$ clauses. A clause such as $x_1 \vee x_3 \vee \neg x_5$ naturally corresponds to an equality constraint of the form $x_1 + x_3 + (1 - x_5) = 1$, since a 0/1-assignment to the variables satisfies exactly one literal of the clause if and only if it satisfies the equality. To find redundant clauses, transform each of the m clauses into an equality to obtain a system of equalities $A\mathbf{x} = \mathbf{b}$ where A is an $m \times n$ matrix, $\mathbf{x}$ is the column vector $(x_1, \ldots, x_n)$, and $\mathbf{b}$ is an integer column vector. Using Gaussian elimination, one can efficiently compute a basis B for the row space of the extended matrix $(A|b)$: a set of equalities such that every equality can be written as a linear combination of equalities in B. Since $(A|b)$ has $n + 1$ columns, its rank is at most $n + 1$ and the basis B contains at most $n + 1$ equalities. To perform data reduction, remove all clauses from the Exact d-sat instance whose corresponding equalities do not occur in B. If an assignment satisfies $f_1(\mathbf{x}) = b_1$ and $f_2(\mathbf{x}) = b_2$, then it also satisfies their sum $f_1(\mathbf{x}) + f_2(\mathbf{x}) = b_1 + b_2$, and any linear combination of the satisfied equalities. Since any equality not in B can be written as a linear combination of equalities in B, a truth assignment satisfying all clauses from B must necessarily also satisfy the remaining clauses, which shows the correctness of the data reduction procedure. The resulting instance can be encoded in $\mathcal{O}(n \log n)$ bits, as each of the remaining $n + 1$ clauses has $d \in O(1)$ literals.

Our results

Our positive results are generalizations of the linear-algebraic data reduction tool for binary CSPs presented above. They reveal that the $\widetilde{\mathcal{O}}(n)$-bit compression for Exact d-sat, the $\widetilde{\mathcal{O}}(n^{d-1})$-bit compression for d-nae-sat, and the $\mathcal{O}(n^d)$-bit compression for d-cnf-sat are samples of a gliding scale of problem complexity: more tightly constrained problems can be compressed better. We formalize this idea by considering a generic CSP whose constraints are of the form $f(\mathbf{x}) = 0$, where f is a bounded-degree polynomial and the constraint demands that $\mathbf{x}$ is a root of f. The example given earlier shows that EXACT d-SAT can be expressed using degree-1 polynomials. We show that d-NAE-SAT and d-CNF-SAT can be expressed using equalities of polynomial expressions of degree $d-1$ and d. We study the following problem:

d-POLYNOMIAL ROOT CSP **Parameter:** The number of variables n.
Input: A list L of polynomial equalities over variables $V = \{x_1, \ldots, x_n\}$. An equality is of the form $f(x_1, \ldots, x_n) = 0$, where f is a multivariate polynomial of degree at most d.
Question: Does there exist an assignment of the variables $\tau\colon V \to \{0,1\}$ satisfying all equalities in L?

Using a generalization of the argument presented above, the number of constraints in an instance of d-POLYNOMIAL ROOT CSP can efficiently be reduced to $\mathcal{O}(n^d)$, even when the number of variables that occur in a constraint is not restricted. The latter implies, for example, that using degree-1 polynomials one can express the EXACT SAT problem with clauses of arbitrary size. When the number of variable occurrences in a constraint can be as large as n, it may take $\Omega(n)$ bits to encode a single constraint. After reducing the number of clauses in an EXACT SAT instance to $n+1$, one may therefore still require $\Theta(n^2)$ bits to encode the instance. This turns out to be unavoidable: we prove that EXACT SAT has no sparsification of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. In general, we compress instances of d-POLYNOMIAL ROOT CSP to bitsize $\widetilde{\mathcal{O}}(n^{d+1})$ when each constraint can be encoded in $\widetilde{\mathcal{O}}(n)$ bits. We prove that no compression to size $\mathcal{O}(n^{d+1-\varepsilon})$ is possible unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. When each constraint can be encoded in $\widetilde{\mathcal{O}}(1)$ bits, the constraint reduction scheme reduces the size of an instance to $\widetilde{\mathcal{O}}(n^d)$. As we will show that d-NAE-SAT can be modeled using polynomials of degree $d-1$, this method strictly generalizes our earlier results [20] for d-NAE-SAT.

The linear-algebraic data reduction tool described above works over arbitrary fields F, allowing us to capture constraints such as "the number of satisfied literals in the clause is exactly two, when evaluated modulo 3". We therefore extend our study to the d-POLYNOMIAL ROOT CSP problem over arbitrary fields F, and obtain similar positive and negative results.

Finally, we consider binary CSPs whose constraints are formed by *inequalities*, rather than equalities, of degree-d polynomials. This leads to the following generic problem:

d-POLYNOMIAL NON-ROOT CSP OVER F **Parameter:** The number of variables n.
Input: A list L of polynomial inequalities over variables $V = \{x_1, \ldots, x_n\}$. An inequality is of the form $f(x_1, \ldots, x_n) \neq 0$, where f is a multivariate polynomial of degree $\leq d$.
Question: Does there exist an assignment of the variables $\tau\colon V \to \{0,1\}$ satisfying all inequalities in L?

We present upper and lower bounds for problems of this type. When the polynomials are evaluated over a structure that is not a field, the situation changes significantly. For example, CSPs with constraints of the type "the number of satisfied literals in the clause is 1 or 2, when evaluated modulo 6" behave differently than the corresponding problem modulo 5, or modulo 7, because the integers modulo 6 do not form a field. Both our upper- and lower bound techniques fail when defining constraints with respect to composite moduli. We present connections to different areas of theoretical computer science where the distinction between prime and composite moduli plays a big role. More concretely, we show that obtaining polynomial sparsification upper bounds for d-Polynomial non-root CSP over the integers modulo a composite, would resolve a long-standing problem concerning the representation of the or-function using low-degree polynomials (cf. [2, 3, 29]).

Related work

Schaefer's Theorem [28] is a classic result relating the complexity of a binary CSP to the type of allowed constraints, separating the NP-complete from the polynomial-time solvable cases. A characterization of the kernelization complexity of min-ones CSPs parameterized by the number of variables was presented by Kratsch and Wahlström [25]. There are several parameterized complexity results for CSPs [7, 9, 24].

2 Preliminaries

A *parameterized problem* $\mathcal{Q}$ is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. Let $\mathcal{Q}, \mathcal{Q}' \subseteq \Sigma^* \times \mathbb{N}$ be parameterized problems and let $h\colon \mathbb{N} \to \mathbb{N}$ be a computable function. A *generalized kernel for $\mathcal{Q}$ into $\mathcal{Q}'$ of size $h(k)$* is an algorithm that, on input $(x,k) \in \Sigma^* \times \mathbb{N}$, takes time polynomial in $|x| + k$ and outputs an instance (x',k') such that:

1. $|x'|$ and k' are bounded by $h(k)$, and
2. $(x',k') \in \mathcal{Q}'$ if and only if $(x,k) \in \mathcal{Q}$.

The algorithm is a *kernel* for $\mathcal{Q}$ if $\mathcal{Q} = \mathcal{Q}'$. It is a *polynomial (generalized) kernel* if $h(k)$ is a polynomial. Since a polynomial-time reduction to an equivalent sparse instance yields a generalized kernel, we use lower bounds for the sizes of generalized kernels to prove the non-existence of sparsification algorithms.

A *linear-parameter transformation* from a parameterized problem $\mathcal{Q}$ to a parameterized problem $\mathcal{Q}'$ is a polynomial-time algorithm that transforms any instance (x,k) of $\mathcal{Q}$ into an equivalent instance (x',k') of $\mathcal{Q}'$ such that $k' \in \mathcal{O}(k)$. It is easy to see (cf. [6]) that the existence of a linear-parameter transformation from $\mathcal{Q}$ to $\mathcal{Q}'$, together with a (generalized) kernel of size $\mathcal{O}(k^d)$ for $\mathcal{Q}'$, yields a generalized kernel of size $\mathcal{O}(k^d)$ for $\mathcal{Q}$. By contraposition, the existence of such a transformation implies that when $\mathcal{Q}$ does not have generalized kernels of size $\mathcal{O}(k^{d-\varepsilon})$, then $\mathcal{Q}'$ does not have generalized kernels of size $\mathcal{O}(k^{d-\varepsilon})$ either.

We use the framework of cross-composition [5] to establish kernelization lower bounds, requiring the definitions of polynomial equivalence relations [5, Def. 3.1] and or-cross-compositions [5, Def. 3.3].

▶ **Theorem 1** ([5, Theorem 6]). *Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let d, ε be positive reals. If L is NP-hard under Karp reductions, has an* or*-cross-composition into $\mathcal{Q}$ with cost $f(t) = t^{1/d+o(1)}$, where t denotes the number of instances, and $\mathcal{Q}$ has a polynomial (generalized) kernelization with size bound $\mathcal{O}(k^{d-\varepsilon})$, then* NP $\subseteq$ coNP/poly.

For $d \in \mathbb{N}$ we will refer to an or-cross-composition of cost $f(t) = t^{1/d}\log(t)$ as a *degree-d cross-composition*. By Theorem 1, a degree-d cross-composition can be used to rule out generalized kernels of size $\mathcal{O}(k^{d-\varepsilon})$. Note that when studying sparsification, we use the number of vertices or variables in the instance (which is usually denoted by n) as the parameter value (which is usually denoted by k).

When interpreting truth assignments as elements of a field, we equate the value *true* with the 1 element in the field (multiplicative identity), and the value *false* with the 0 element (additive identity). Consequently, for a boolean variable x its negation $\neg x$ corresponds to $(1 - x)$. We let $\mathbb{Z}/m\mathbb{Z}$ denote the integers modulo m, which form a field if m is a prime number. The *degree* of a multivariate polynomial is the maximum degree of its monomials. Let $f(x_1,\ldots,x_d)$ be a d-variate polynomial over a field F. The *root set* of f is the algebraic variety $\{(e_1,\ldots,e_d)\in F^d \mid f(e_1,\ldots,e_d)=0\}$. For a field F and a finite set $S\subseteq F$ of elements, the univariate polynomial $f(x) := \prod_{s\in S}(x-s)$ over F of degree $|S|$ has root set exactly S. We say that a field F is *efficient* if the field operations and Gaussian elimination can be done in polynomial time in the size of a reasonable input encoding. The field of rational numbers $\mathbb{Q}$, and all finite fields, are efficient. We use $[n]$ to denote $\{1,\ldots,n\}$. The $\widetilde{\mathcal{O}}$-notation suppresses polylogarithmic factors: $\widetilde{\mathcal{O}}(n)=\mathcal{O}(n\log^c n)$ for a constant c. For statements marked with a (★), the proof can be found in the full version [21].

3 Kernel upper bounds

3.1 Polynomial root CSP

We start by showing how to reduce the number of constraints in instances of d-Polynomial root CSP, by extending the argument presented in the introduction.

▶ **Theorem 2.** *There is a polynomial-time algorithm that, given an instance (L,V) of d-*Polynomial root CSP *over an efficient field F, outputs an equivalent instance (L',V) with at most n^d+1 constraints such that $L'\subseteq L$.*

Proof. Given a list L of polynomial equalities over variables V for d-Polynomial root CSP, we use linear algebra to find redundant constraints. Observe that $(x_i)^c = x_i$ for all 0/1-assignments and $c\in\mathbb{N}_+$. As constraints are evaluated over 0/1-assignments, we may assume without loss of generality that the monomials in each of the polynomials are multilinear: each monomial consists of a coefficient from F multiplied by distinct variables.

Create a matrix A with $|L|$ rows and a column for every multilinear monomial of degree at most d over variables from V. Let position $a_{i,j}$ in A be the coefficient of the monomial corresponding to column j in the polynomial equality corresponding to row i.

Compute a basis B of the row space of matrix A, for example using Gaussian elimination [17], and let L' consist of the equalities in L whose corresponding row appears in the basis. Since $L'\subseteq L$, it follows that if the original instance has a satisfying assignment, the reduced instance has a satisfying assignment as well. The crucial part of the correctness proof is to establish the converse.

▶ **Claim 3.** *If an assignment $\tau\colon V\to\{0,1\}$ of the variables in V satisfies the equalities in L', then it satisfies all equalities in L.*

Proof. Consider any equality $(f(\mathbf{x})=0)\in L\setminus L'$, since equalities in L' are trivially satisfied, and assume it corresponds to the i'th matrix row. Let $f_j(\mathbf{x})$ be the polynomial represented in the j'th row of matrix A for $j\in[|L|]$. Without loss of generality, let the basis of A correspond to its first m rows $\mathbf{a}_1,\ldots,\mathbf{a}_m$. We then have $i>m$, and by the definition of basis there exist $\beta_1,\ldots,\beta_m\in F$ such that $\mathbf{a}_i=\sum_{j=1}^m \beta_j\mathbf{a}_j$. Let $\mathbf{t}$ be the column vector containing, for each multilinear monomial of degree $\leq d$ in variables $x_1,\ldots,x_n$, the evaluation under τ. For example, for monomial x_1x_3 it contains $\tau(x_1)\cdot\tau(x_3)$. By using the same order of monomials as in the construction of A, we obtain for all $j\in[|L|]$ that $f_j(\tau(x_1),\ldots,\tau(x_n))=\mathbf{a}_j\mathbf{t}$, the inner product of $\mathbf{a}_j$ and $\mathbf{t}$. It follows that $\mathbf{a}_j\mathbf{t}=0$ for all $j\in[m]$, since satisfying L' implies $f_j(\tau(x_1),\ldots,\tau(x_n))=0$. To conclude the proof, note that

$$f_i(\mathbf{x})=\mathbf{a}_i\mathbf{t}=\sum_{j=1}^m(\beta_j\mathbf{a}_j)\mathbf{t}=\sum_{j=1}^m\beta_j(\mathbf{a}_j\mathbf{t})=\sum_{j=1}^m\beta_j\cdot 0=0. \qquad ◀$$

▶ **Claim 4.** *The number of constraints in the resulting kernel is bounded by $n^d + 1$.*

Proof. The size of a basis of any matrix over a field equals its rank, which is bounded by the number of columns. As there is a column for each multilinear monomial of degree at most d, there are at most $\sum_{i=0}^{d} \binom{n}{i}$ constraints in the basis. Now observe that $\sum_{i=1}^{d} \binom{n}{i} \leq n^d$. The left side counts nonempty subsets of $[n]$ of size at most d, each of which can be mapped to a distinct d-tuple by repeating an element. Since there are n^d d-tuples, the claim follows. ◀

This concludes the proof of Theorem 2. ◀

When each constraint can be encoded in $\widetilde{\mathcal{O}}(n)$ bits, for example when each polynomial can be represented as an arithmetic circuit of size $\mathcal{O}(n)$, Theorem 2 gives a kernelization of size $\widetilde{\mathcal{O}}(n^{d+1})$. When constraints can be encoded in $\widetilde{\mathcal{O}}(1)$ bits, which may occur when constraints have constant arity, we obtain kernels of bitsize $\widetilde{\mathcal{O}}(n^d)$. For explicit examples consider the following problem, where optionally a prime p may be chosen.

Generalized d-Sat (mod p) **Parameter:** The number of variables n
Input: A set of clauses $\mathcal{C}$ over variables $V := \{x_1, \ldots, x_n\}$, and for each clause a set $S_i \subset \mathbb{N} \cup \{0\}$ with $|S_i| \leq d$. Each clause is a set of distinct literals of the form x_i or $\neg x_i$.
Question: Does there exist a truth assignment for the variables V such that the number of satisfied literals in clause i lies in $S_i \pmod p$ for all i?

▶ **Corollary 5.** Generalized d-Sat *and* Generalized d-Sat mod p *both have a kernel with $n^d + 1$ clauses that can be encoded in $\mathcal{O}(n^{d+1} \log n)$ bits.*

Proof. To reduce the number of clauses using Theorem 2, we only have to provide a polynomial of degree at most d to represent each constraint. Consider a clause involving k variables $x_{i_1}, \ldots, x_{i_k}$. Let $t_j = x_{i_j}$ if variable x_{i_j} occurs positively in the clause, and let $t_j = (1 - x_{i_j})$ if the variable occurs negatively. Then the number of satisfied literals in the clause is given by the degree-1 polynomial $f(x_{i_1}, \ldots, x_{i_k}) := \sum_{i=1}^{k} t_i$. Let $F(x)$ be a polynomial with root set $S_j \pmod p$ of degree at most $|S_j|$. We obtain $F(f(\mathbf{x})) \equiv 0 \pmod p$ if and only if $\mathbf{x}$ satisfies the clause. Note that the degree of $F(f(\mathbf{x}))$ is at most $|S_j| \leq d$.

Applying Theorem 2 to the resulting instance of d-Polynomial root CSP identifies a subset of at most $n^d + 1$ constraints which preserve the answer to the Sat problem. Each clause contains at most $2n$ literals, which can be encoded in $\mathcal{O}(\log n)$ bits each. Additionally, for each clause we need to store the set S_i of at most d integers, which have value at most $2n$ in relevant inputs. As d is a constant, the instance can be encoded in $\mathcal{O}(n^{d+1} \log n)$ bits. ◀

Corollary 5 yields a new way to get a nontrivial compression for d-nae-sat, which is conceptually simpler than the existing approach which requires an unintuitive lemma by Lovász [27]. The new approach gives the same size bound as given earlier [20].

▶ **Corollary 6.** d-nae-sat *has a kernel with $n^{d-1} + 1$ clauses and bitsize $\mathcal{O}(n^{d-1} \log n)$.*

Proof. A clause of size $k \leq d$ is not-all-equal satisfied if and only if the number of satisfied literals lies in $S := \{1, \ldots, k-1\}$. Using Corollary 5 we can reduce the number of clauses to $n^{d-1} + 1$. Each clause has $d \in O(1)$ variables and can thus be encoded in $\mathcal{O}(\log n)$ bits. ◀

3.2 Polynomial non-root CSP

In this section we consider d-Polynomial non-root CSP. In Section 4.2 we will show that, over the field of rational numbers, the problem cannot be compressed to size polynomial in n, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. We therefore consider the field $\mathbb{Z}/p\mathbb{Z}$ of integers modulo a prime p.

▶ **Theorem 7.** *There is a polynomial-time algorithm that, given an instance (L, V) of d-*Polynomial non-root CSP *over $\mathbb{Z}/p\mathbb{Z}$, outputs an equivalent instance (L', V) with $\mathcal{O}(n^{d(p-1)})$ constraints such that $L' \subseteq L$.*

Proof. Suppose we are given a list of polynomial inequalities L over variables V. Observe that an inequality $f(\mathbf{x}) \not\equiv 0 \pmod p$ is equivalent to $f(\mathbf{x}) \in \{1, \ldots, p-1\} \pmod p$.

Let $F\colon \mathbb{Z}/p\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$ be a polynomial of degree $p-1$ with root set $\{1, \ldots, p-1\}$ modulo p, which exists since $\mathbb{Z}/p\mathbb{Z}$ is a field. Then $f(\mathbf{x}) \not\equiv 0 \pmod p$ can equivalently be stated as $F(f(\mathbf{x})) \equiv 0 \pmod p$. It is easy to see that $F(f(\mathbf{x}))$ is a polynomial of degree at most $d(p-1)$. Therefore, L can be written as an instance of $d(p-1)$-Polynomial root CSP by replacing every polynomial f by $F \circ f$. By Theorem 2, the proof follows. ◀

In Section 4.2 we will establish a nearly-matching lower bound counterpart to Theorem 7.

4 Kernel lower bounds

4.1 Polynomial root CSP

We now turn our attention to lower bounds, starting with d-Polynomial root CSP over $\mathbb{Q}$. We start by proving that Exact Red-Blue Dominating Set does not have generalized kernels of bitsize $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. The same lower bound for 1-Polynomial root CSP will follow by a linear-parameter transformation. We then show how to generalize this result to d-Polynomial root CSP. As a starting problem for the cross-composition we will use the NP-hard Red-Blue Dominating Set (rbds) [11, 22].

Red-Blue Dominating Set (rbds) **Parameter:** The number of vertices n
Input: A bipartite graph $G = (R \cup B, E)$ containing red (R) and blue (B) vertices, and an integer k.
Question: Does there exist a set $D \subseteq R$ with $|D| \leq k$ such that every vertex in B has at least one neighbor in D?

Exact Red Blue Dominating Set (erbds) is defined similarly, except that every vertex in B must have *exactly one* neighbor in D. Furthermore we will not bound the size of such a set, but merely ask for the existence of any erbds.

▶ **Theorem 8.** Exact Red-Blue Dominating Set *parameterized by the number of vertices n does not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Proof. We will prove this result by giving a degree-2 cross-composition from rbds to erbds. We start by giving a polynomial equivalence relation $\mathcal{R}$ on inputs of rbds. Let two instances of rbds be equivalent under $\mathcal{R}$ if they have the same number of red vertices, the same number of blue vertices, and the same maximum size of a rbds. It is easy to check that $\mathcal{R}$ is a polynomial equivalence relation.

Assume we are given t instances of rbds, labeled $X_{i,j}$ for $i, j \in [\sqrt{t}]$, from the same equivalence class of $\mathcal{R}$. If the number of instances given is not a square, we duplicate one of the input instances until a square number is reached. Since this changes the number of

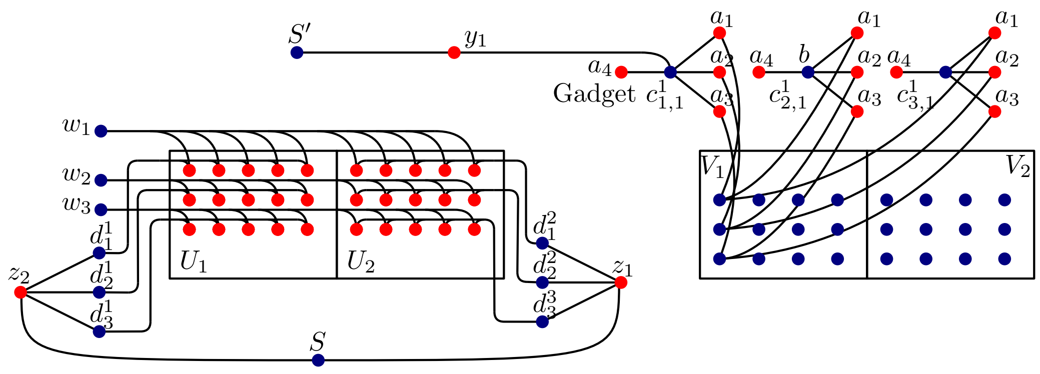


Figure 1 The graph G' created in the proof of Theorem 8, for $k = 3$, $m_R = 5$, $m_B = 4$, and $t = 4$. Edges between U and V are left out for simplicity. Of the 24 gadgets in C only $c^1_{1,1,}$, $c^1_{2,1}$, and $c^1_{3,1}$ are shown. Vertex y_2 is left out.

inputs by at most a factor four, this does not influence the cross-composition. Instance $X_{i,j}$ consists of graph $G_{i,j}$ with a set of red vertices $R_{i,j}$ and blue vertices $B_{i,j}$. Call the number of red vertices in every instance m_R, the number of blue vertices m_B, and the required size of the dominating set k. For each instance enumerate the red vertices as $r_1, \dots, r_{m_R}$ and the blue vertices as $b_1, \dots, b_{m_B}$, arbitrarily. Create instance G' for ERBDS by the following steps. Figure 1 shows a sketch of G'.

1. Create $\sqrt{t}$ sets $U_1, \dots, U_{\sqrt{t}}$ each consisting of $k \cdot m_R$ red vertices, such that for all $\ell \in [\sqrt{t}]$ $U_\ell := \{u^\ell_{i,j} \mid i \in [k], j \in [m_R]\}$.
2. Similarly create $\sqrt{t}$ sets $V_1, \dots, V_{\sqrt{t}}$, each consisting of $k \cdot m_B$ blue vertices, and define $V_\ell := \{v^\ell_{i,j} \mid i \in [k], j \in [m_B]\}$ for all $\ell \in [\sqrt{t}]$.
3. For each $i \in [k]$ add the edge from $u^\ell_{i,j}$ to $v^{\ell'}_{i,j'}$ if $\{r_j, b_{j'}\}$ is an edge in instance $X_{\ell,\ell'}$ with $\ell, \ell' \in [\sqrt{t}], j \in [m_R], j' \in [m_B]$.

By steps 1 to 3, the graph induced by the vertices in $U_\ell \cup V_{\ell'}$ consists of k vertex-disjoint copies of $G_{\ell,\ell'}$. The next steps are used to ensure that there are exactly k vertices from U in any ERBDS, which must all belong to the same set U_ℓ.

4. Create k blue vertices $W := \{w_i \mid i \in [k]\}$ and connect all vertices $\{u^\ell_{i,j} \mid j \in [m_R], \ell \in [\sqrt{t}]\}$ to w_i for $i \in [k]$.
5. Create blue vertices d^ℓ_i for $\ell \in [\sqrt{t}]$ and $i \in [k]$. Connect vertex d^ℓ_i to the vertices $u^\ell_{i,j}$ with $j \in [m_R]$. Add blue vertex S and red vertices $Z := \{z_j \mid j \in [\sqrt{t}]\}$ and connect z_j to d^ℓ_i for $i \in [k]$ and $\ell \neq j \in [\sqrt{t}]$. Connect all vertices in Z to vertex S.

The next steps ensure that some of the blue vertices in one set V_ℓ need to be dominated by vertices from U, while all other vertices in V can be dominated "for free".

6. Add sets of gadgets C_ℓ for $\ell \in [\sqrt{t}]$. Each set consists of $m_B \cdot k$ selector gadgets $c^\ell_{i,j}$. A selector gadget consists of $k+1$ red vertices labeled $a_1, \dots, a_{k+1}$ that are all connected to a blue vertex b that is private to the gadget. Furthermore, in gadget $c^\ell_{i,j}$, the vertex a_x for $x \in [k]$ is connected to $v^\ell_{x,j}$ for $j \in [m_B]$, $\ell \in [\sqrt{t}]$ and $i \in [k]$. By this construction an ERBDS uses at most one red vertex from each gadget, to dominate one vertex from V.
7. Add red vertices $Y := y_1, \dots, y_{\sqrt{t}}$ and connect y_ℓ to the blue vertices of gadgets $c^\ell_{1,j}$ for all $j \in [m_B]$, $\ell \in [\sqrt{t}]$. Connect $y_1, \dots, y_{\sqrt{t}}$ to the new blue vertex S'.

This concludes the construction of graph G', with red vertices ($U \cup Y \cup Z \cup$ vertices labeled $a_1, \dots, a_{k+1}$ in C), and blue vertices ($V \cup D \cup \{S, S'\} \cup$ vertices labeled b in C).

▶ **Claim 9.** *For any* ERBDS *E of G', there exists an index $\ell \in [\sqrt{t}]$ such that $U_x \cap E = \emptyset$ for all $x \neq \ell \in [\sqrt{t}]$ and $|E \cap \{u^{\ell}_{i,j} \mid j \in [m_R]\}| = 1$ for all $i \in k$.*

Proof. By Step 5, blue vertex S has neighborhood $\{z_\ell \mid \ \ell \in [\sqrt{t}]\}$. Exactly one of these vertices is contained in E; let this be z_ℓ. The neighborhood of z_ℓ contains $\{d^j_i \mid i \in [k], j \in [\sqrt{t}] \setminus \{\ell\}\}$. Thereby, no other neighbors from vertices in this set can be in E, implying no vertices from U_i for $i \neq \ell \in [\sqrt{t}]$ can be in E. In other words, $U_i \cap E = \emptyset$ for all $i \neq \ell \in [\sqrt{t}]$.

By Step 4, the neighborhood of blue vertex w_i for $i \in [k]$ is exactly $\{u^x_{i,j} \mid x \in [\sqrt{t}], j \in [m_R]\}$. It follows that exactly one vertex in this set is in E for all i. By the previous argument the vertex cannot be from U_x for $x \neq \ell$, hence it is from U_ℓ. ◀

▶ **Claim 10.** *For any* ERBDS *E of G', there exists ℓ such that $E \cap c^{\ell}_{1,j} = \emptyset$ for all $j \in [m_B]$.*

Proof. By Step 7, blue vertex S' has neighborhood $\{y_\ell \mid \ell \in [\sqrt{t}]\}$. Exactly one of these vertices is contained in E; let this be y_ℓ. It is connected to the blue vertex of all gadgets $c^\ell_{1,j}$ for $j \in [m_B]$. Since all red vertices in a gadget $c^\ell_{1,j}$ for $j \in [m_B]$ have a blue neighbor b that is also adjacent to $y_\ell \in E$, the red vertices in these gadgets are not present in E. ◀

▶ **Claim 11.** *For any* ERBDS *E of G', there exists an index ℓ such that for every $j \in [m_B]$ at least one of the vertices in $\{v^\ell_{i,j} \mid i \in [k]\}$ has a neighbor in $E \cap U$.*

Proof. By Claim 10 there exists $\ell \in [\sqrt{t}]$ such that $E \cap c^\ell_{1,j} = \emptyset$ for all $j \in [m_B]$. Consider an arbitrary $j \in [m_B]$. The k vertices in $\{v^\ell_{i,j} \mid i \in [k]\}$ are connected to k gadgets $c^\ell_{1,j}, c^\ell_{2,j}, \ldots, c^\ell_{k,j}$, and to some vertices in U. From each gadget, at most one red vertex is in E, since the red vertices have a common blue neighbor. Any red gadget vertex is connected to only one vertex in V. Since no vertex of gadget $c^\ell_{1,j}$ is in E, at most $k-1$ of the vertices in $\{v^\ell_{i,j} \mid i \in [k]\}$ have a neighbor in $E \cap C_\ell$. Consequently, at least one of these vertices has a neighbor in $E \cap U$ for each $j \in [m_B]$. ◀

▶ **Claim 12.** *If G' has an* ERBDS, *then some input $X_{i,j}$ has a* RBDS *of size at most k.*

Proof. Assume G' has an ERBDS, say E. By Claim 11, there exists $\ell_2 \in [\sqrt{t}]$, such that for every $j \in [m_B]$ at least one of the vertices in $\{v^\ell_{i,j} \mid i \in [k]\}$ has a neighbor in $E \cap U$. By Claim 9, there exists $\ell_1 \in [\sqrt{t}]$ with $U_i \cap E = \emptyset$ for all $i \neq \ell_1$, so these neighbors lie in U_{ℓ_1}.

We now construct a RBDS E' for instance X_{ℓ_1,ℓ_2}. For each $j \in [m_R]$, add r_j to E' if $E \cap \{u^{\ell_1}_{i,j} \mid i \in [k]\} \neq \emptyset$. By Claim 9, it follows that E' has size at most k, as required. It remains to show that every vertex in B_{ℓ_1,ℓ_2} has a neighbor in E'. If some vertex b_j from B_{ℓ_1,ℓ_2} does not have a neighbor in E', then none of the vertices $\{v^{\ell_2}_{i,j} \mid i \in [k]\}$ have a neighbor in $E \cap U_{\ell_1}$. This contradicts Claim 11. Hence E' is an RBDS of size at most k for B_{ℓ_1,ℓ_2}. ◀

▶ **Claim 13.** *If some input instance has a* RBDS *of size at most k, then G' has an* ERBDS.

Proof. Suppose instance X_{ℓ_1,ℓ_2} has a RBDS E' of size k consisting of vertices $r_{i_1}, \ldots, r_{i_k} \subseteq R_{\ell_1,\ell_2}$. We construct an ERBDS E for G'. Start by choosing vertices $u^{\ell_1}_{x,i_x}$ for $x \in [k]$, so for every vertex in E' we pick one vertex in the ERBDS for G'. Furthermore we choose the red vertices z_{ℓ_1} and y_{ℓ_2} to be in E. To exactly dominate the blue vertices in V, we use the gadgets in C as follows. For $\ell \neq \ell_2 \in [\sqrt{t}]$, add red vertex a_x of gadget $c^\ell_{x,j}$ if vertex $v^\ell_{x,j}$ does not yet have a neighbor in E, for $j \in [m_R]$. Else, add vertex a_{k+1} of gadget $c^\ell_{x,j}$ to E, in order to exactly dominate the blue vertex of this gadget.

To exactly dominate the vertices in V_{ℓ_2} we apply a similar procedure, except that gadget $c^\ell_{1,j}$ cannot be used since its blue vertex b is already dominated by y_{ℓ_2}. Since E' is a RBDS of instance X_{ℓ_1,ℓ_2}, for each $j \in [m_B]$ at least one vertex from set $\{v^{\ell_2}_{i,j} \mid i \in [k]\}$ has a neighbor in $E \cap U$. As such, the $k-1$ remaining gadgets can be used to each dominate one of the $k-1$ remaining vertices in this set, if they do not already have a neighbor in $E \cap U$. If no red vertex of a gadget is needed to dominate, we choose vertex a_{k+1} of the gadget in E to dominate the blue vertex in the gadget.

It is straight-forward to verify that this results in an ERBDS for G'. ◀

From Claims 12 and 13 it follows that graph G' has an ERBDS if and only if at least one of the input instances has a RBDS of size at most k. The graph G' has $\mathcal{O}(\sqrt{t} \cdot (m_R + m_B)^3)$ vertices, which is suitably bounded for a cross-composition. By Theorem 1, it follows that ERBDS parameterized by the number of vertices n does not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$. ◀

Using Theorem 8 we provide lower bounds for constraint satisfaction problems.

▶ **Corollary 14.** *The problems* EXACT SATISFIABILITY *and* 1-POLYNOMIAL ROOT CSP *over $\mathbb{Q}$, parameterized by the number of variables n, do not have a generalized kernel of size $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Proof. By Theorem 8 and the discussion in Section 2, it suffices to give linear-parameter transformations from ERBDS parameterized by the number of vertices to the two mentioned problems. Consider an instance $G = (R \cup B, E)$ of ERBDS. Create a binary variable x_r for each $r \in R$. For each blue vertex $b \in B$ create a clause of the form $\bigvee_{r \in N(b)} x_r$ (to build an instance of EXACT SAT), or create a constraint $\sum_{r \in N(b)} x_r = 1$ (to build an instance of CSP). The resulting instance has a satisfying 0/1-assignment if and only if G has an ERBDS. Since the number of variables is $|R| \leq n$, these are valid linear-parameter transformations. ◀

▶ **Theorem 15.** *d-*POLYNOMIAL ROOT CSP *over $\mathbb{Q}$ parameterized by the number of variables n does not have a generalized kernel of size $\mathcal{O}(n^{d+1-\varepsilon})$ for any $\varepsilon > 0$, unless* $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

Proof. The case $d = 1$ is covered by Corollary 14; we consider $d \geq 2$ and give a degree-$(d+1)$ cross-composition from RBDS, re-using some parts of the proof of Theorem 8. Suppose we are given $t = r^{d+1}$ instances of RBDS from the same equivalence class of $\mathcal{R}$, all having m_R red vertices and m_B blue vertices. By a similar padding argument as before, we may assume r is an integer. Split the inputs into groups of size r^2 and apply the cross-composition of Theorem 8 to each group, followed by the linear-parameter transformation in Corollary 14. We obtain r^{d-1} instances of 1-POLYNOMIAL ROOT CSP with $O(r \cdot \mathrm{poly}(m_R + m_B))$ variables each, such that the answer to each composed instance is the logical OR of the answers to the RBDS instances in its group. Label the instances resulting from the group compositions $X_{i_1,\ldots,i_{d-1}}$ with $i_1, \ldots, i_{d-1} \in [r]$. They all use the same number of variables; label the variables in each instance as $x_1, \ldots, x_q$. Create an instance L' of d-POLYNOMIAL ROOT CSP as follows:

1. Create variables $x_1, \ldots, x_q$. Create sets $Y_1, \ldots, Y_{d-1}$ of r variables each, where $Y_i := \{y^i_j \mid j \in [r]\}$. Add the requirement $\sum_{j \in [r]} y^i_j = 1$ to L' for each $i \in [d-1]$.
2. Let the list of equations of instance $X_{i_1,\ldots,i_{d-1}}$ be $L_{i_1,\ldots,i_{d-1}}$. For every equality $f(\mathbf{x}) = 1$ in $L_{i_1,\ldots,i_{d-1}}$ with $i_1, \ldots, i_d \in [r]$, add the following equality to L':

$$f(\mathbf{x}) \cdot \prod_{z \in [d-1]} y^z_{i_z} = \prod_{z \in [d-1]} y^z_{i_z}.$$

The polynomial equalities have degree $\leq d$ since $f(\mathbf{x})$ has degree 1. The number of variables is $q + (d-1) \cdot r \in \mathcal{O}(r \cdot d \cdot \mathrm{poly}(m_R + m_B)) \in \mathcal{O}(t^{1/(d+1)} \mathrm{poly}(m_R + m_B))$. It remains to show that L' is satisfiable if and only if one of the input instances has an ERBDS. Since Theorem 8 gives a correct cross-composition, it is sufficient to show that L' is satisfiable if and only if one of the r^{d-1} instances of 1-Polynomial root CSP has a solution.

($\Rightarrow$) Suppose L' is satisfied by some assignment. Then from each Y_i for $i \in [d-1]$, exactly one variable is set to 1. So suppose variables $y^z_{i_z}$ are set to 1 for $z \in [d-1], i_z \in [r]$. Then from instance $X_{i_1,\ldots,i_{d-1}}$, all polynomial equations are copied to L' and multiplied by 1 on both sides. Hence they are satisfied by the assignment to $\mathbf{x}$.

($\Leftarrow$) Suppose instance $X_{i_1,\ldots,i_{d-1}}$ of 1-Polynomial root CSP has a satisfying assignment. Set the $\mathbf{x}$-variables according to this assignment. Furthermore, set variables $y^z_{i_z}$ for $z \in [d-1]$ to 1, set all other variables to 0. Thereby the sum of variables in each set Y_i is 1, as required. Furthermore, any equation added in Step 2 of the construction is satisfied in the following way. If it belongs to instance $X_{i_1,\ldots,i_{d-1}}$, it is satisfied by definition. Equations belonging to any other instance are trivially satisfied since both sides are multiplied by zero. ◀

Observe that the polynomials constructed in Theorem 15 have a simple form: each polynomial is a product of $(d-1)$ Y-variables multiplied by a sum of distinct variables from $\mathbf{x}$. Each polynomial can therefore be encoded in $\widetilde{\mathcal{O}}(n)$ bits, where n is the number of variables in the constructed CSP. The sparsification of Theorem 2 therefore encodes such instances in $\widetilde{\mathcal{O}}(n^{d+1})$ bits. The lower bound shows that this is optimal up to $n^{o(1)}$ factors.

We expect the lower bound of Theorem 15 to extend to arbitrary finite fields of prime order, except for the case $d = 1$ over $\mathbb{Z}/2\mathbb{Z}$, which is polynomial-time solvable [28].

4.2 Polynomial non-root CSP

We start our lower bound discussion for d-Polynomial non-root CSP by considering polynomials over $\mathbb{Q}$. 1-Polynomial non-root CSP over $\mathbb{Q}$ does not have a generalized kernel of size bounded by any polynomial in n, unless NP $\subseteq$ coNP/poly. This follows from the fact that CNF-Satisfiability parameterized by the number of variables does not have a kernel of size polynomial in n unless NP $\subseteq$ coNP/poly [10, 14], together with the fact that a clause such as $(x_1 \vee \neg x_3 \vee x_4)$ is satisfied by a 0/1-assignment if and only if $x_1 + (1-x_3) + x_4 \neq 0$ over $\mathbb{Q}$. In the remainder of the section we investigate the behavior over finite fields.

In Theorem 7 we provided a kernel for d-Polynomial non-root CSP over $\mathbb{Z}/p\mathbb{Z}$ for primes p. It is natural to ask whether similar results can be obtained when working with polynomials modulo an arbitrary integer m. When m is composite, our kernelization fails. We can show that this is not a shortcoming of our proof strategy, but a necessity due to the fact that constraints expressed by equalities of degree-d polynomials modulo composite numbers can model more complex constraints than degree-d polynomials modulo a prime. For example, it is known (cf. [1, §2]) that there is a degree-3 polynomial f over the integers modulo 6 which represents a logical or of size 27 in the following way:

$$f(x_1, \ldots, x_{27}) \not\equiv 0 \pmod 6 \Leftrightarrow (x_1 \vee \ldots \vee x_{27}). \tag{1}$$

By this expressibility of a size-27 or by a polynomial of degree 3 over $\mathbb{Z}/6\mathbb{Z}$ using the same variables, it is easy to give a linear-parameter transformation from 27-cnf-sat to 3-Polynomial non-root CSP (mod 6). Using known lower bounds for d-cnf-sat [10, Theorem 1], this implies the latter problem has no kernel of $\mathcal{O}(n^{27-\varepsilon})$ bits, unless NP $\subseteq$ coNP/poly. Plugging in the degree of 3 and modulus 6 into the bound of Theorem 7 would give a reduction to $\mathcal{O}(n^{3\cdot(6-1)}) = \mathcal{O}(n^{15})$ constraints and would contradict the lower bound. The example therefore shows that the problem is more complex for composite moduli.

For more general non-primes, we can prove a lower bound using a general construction by Bhowmick *et al.* [3] of low-degree polynomials representing or in the sense of Equation 1.

▶ **Theorem 16** (★). *Let m be a non-prime with a prime factorization consisting of r distinct primes, such that $m = \prod_{i \in [r]} p_i$. Then d-*Polynomial non-root CSP (mod m) *parameterized by the number of variables n does not have a generalized kernel of size $\mathcal{O}(n^{(d^r)/2-\varepsilon})$ for any $\varepsilon > 0$, unless* NP ⊆ coNP/poly.

In case m does not have a prime factorization in which all primes are distinct, it is possible to obtain weaker a lower bound using a result by Barrington *et al.* [2], which proves that there exists a polynomial of degree $\mathcal{O}(\ell N^{1/r})$ that represents a logical OR when taken modulo m. Here ℓ is the largest prime factor of m. For prime moduli, we provide a lower bound almost matching the upper bound in Section 3.2.

▶ **Theorem 17** (★). *Let p be a prime. d-*Polynomial non-root CSP (mod p) *parameterized by the number of variables n does not have a generalized kernel of size $\mathcal{O}(n^{d(p-1)-\varepsilon})$ for any $\varepsilon > 0$, unless* NP ⊆ coNP/poly.

5 Conclusion

We have given upper and lower bounds on the kernelization complexity of binary CSPs that can be represented by polynomial (in)equalities, obtaining tight sparsification bounds in several cases. Our main conceptual contribution is to analyze constraints on binary variables based on the minimum degree of multivariate polynomials whose roots, or non-roots, capture the satisfying assignments. The ultimate goal of this line of research is to characterize the optimal sparsification size of a binary CSP based on easily accessible properties of the constraint language. To reach this goal, several significant hurdles have to be overcome.

For d-Polynomial non-root CSP (mod 6), we do not know of any way to reduce the number of constraints to polynomial in n. This difficulty is connected to longstanding questions regarding the minimum degree of a multivariate polynomial modulo 6 that represents the OR-function of n variables in the sense of Equation 1. As exploited in the construction of Theorem 16, if the OR-function with $g(d)$ inputs can be represented by polynomials of degree d, then d-Polynomial non-root CSP cannot be compressed to size $\mathcal{O}(n^{g(d)-\varepsilon})$ unless NP ⊆ coNP/poly. By contraposition, a kernelization with size bound $\widetilde{\mathcal{O}}(n^{h(d)})$ implies a lower bound of $h^{-1}(d)$ on the degree of a polynomial representing an OR of arity $h(d)$, assuming NP ⊈ coNP/poly. Kernel bounds where $h(d)$ is polynomially bounded in d, would therefore establish inverse polynomial lower bounds on the degree of polynomials representing an n-variable OR modulo 6. However, the current-best degree lower bound [29] is only $\Omega(\log n)$, which has not been improved in nearly two decades (cf. [3, §1.4]).

When it comes to CSPs whose constraints are of the form "the number of satisfied literals in the clause belongs to set S", many cases remain unsolved. We can prove (★) using the Green-Tao theorem [15] that for constraints of the form "the number of satisfied literals is a prime number", no generalized kernel of size polynomial in n exists unless NP ⊆ coNP/poly. On the other hand, Corollary 5 gives good compressions for problems of the type "the number of satisfied literals in the clause is a multiple of three". Is sparsification possible when a constraint requires the number of satisfied literals to be a square, for example?

A simple example of a CSP whose kernelization complexity is currently unclear has constraints of the form "the number of satisfied literals is one or two, modulo six". The approach of Theorem 2 fails, since there is no polynomial modulo six with root set $\{1, 2\}$.

Finally, we mention that all our results extend to the setting of min-ones and max-ones CSPs, in which one has to find a satisfying assignment that sets at least, or at most, a given number of variables to true. For example, our results easily imply that Exact Hitting Set parameterized by the number of variables n has a sparsification of size $\mathcal{O}(n^2)$, which cannot be improved to $\mathcal{O}(n^{2-\varepsilon})$ unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

References

1 David A. Mix Barrington. Some problems involving Razborov-Smolensky polynomials. In *Proceedings of the London Mathematical Society Symposium on Boolean Function Complexity*, pages 109–128. Cambridge University Press, 1992. doi:10.1017/CBO9780511526633.010.

2 David A. Mix Barrington, Richard Beigel, and Steven Rudich. Representing boolean functions as polynomials modulo composite numbers. *Computational Complexity*, 4(4):367–382, 1994. doi:10.1007/BF01263424.

3 Abhishek Bhowmick and Shachar Lovett. Nonclassical Polynomials as a Barrier to Polynomial Lower Bounds. In *Proc. 30th CCC*, volume 33 of *LIPIcs*, pages 72–87, 2015. doi:10.4230/LIPIcs.CCC.2015.72.

4 Hans L. Bodlaender. Kernelization, exponential lower bounds. In *Encyclopedia of Algorithms*. Springer, 2015. doi:10.1007/978-3-642-27848-8_521-1.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM J. Discrete Math.*, 28(1):277–305, 2014. doi:10.1137/120880240.

6 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.

7 Andrei A. Bulatov and Dániel Marx. Constraint satisfaction parameterized by solution size. *SIAM J. Comput.*, 43(2):573–616, 2014. doi:10.1137/120882160.

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 Holger Dell, Eun Jung Kim, Michael Lampis, Valia Mitsou, and Tobias Mömke. Complexity and approximability of parameterized MAX-CSPs. In *Proc. 10th IPEC*, volume 43 of *LIPIcs*, pages 294–306, 2015. doi:10.4230/LIPIcs.IPEC.2015.294.

10 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. *J. ACM*, 61(4):23:1–23:27, 2014. doi:10.1145/2629620.

11 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Transactions on Algorithms*, 11(2):13, 2014. doi:10.1145/2650261.

12 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

13 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

14 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011. doi:10.1016/j.jcss.2010.06.007.

15 Ben Green and Terence Tao. The primes contain arbitrarily long arithmetic progressions. *Annals of Mathematics*, 167(2):481–547, 2008. doi:10.4007/annals.2008.167.481.

16 Gregory Gutin. Kernelization: Constraint satisfaction problems parameterized above average. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*. Springer, 2015. doi:10.1007/978-3-642-27848-8_524-1.

17 Leslie Hogben. *Handbook of Linear Algebra, Second Edition*. Chapman and Hall/CRC, 2014.

18 Bart M. P. Jansen. On sparsification for computing treewidth. *Algorithmica*, 71(3):605–635, 2015. doi:10.1007/s00453-014-9924-2.

19 Bart M. P. Jansen. Constrained bipartite vertex cover: The easy kernel is essentially tight. In *Proc. 33rd STACS*, volume 47 of *LIPIcs*, pages 45:1–45:13. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.45.

20 Bart M. P. Jansen and Astrid Pieterse. Sparsification upper and lower bounds for graphs problems and not-all-equal SAT. In *Proc. 10th IPEC*, volume 43 of *LIPIcs*, pages 163–174, 2015. doi:10.4230/LIPIcs.IPEC.2015.163.

21 Bart M.P. Jansen and Astrid Pieterse. Optimal sparsification for some binary CSPs using low-degree polynomials. *CoRR*, abs/1606.03233v1, 2016. arXiv:1606.03233v1.

22 R. M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

23 Stefan Kratsch. Recent developments in kernelization: A survey. *Bulletin of the EATCS*, 113:58–97, 2014.

24 Stefan Kratsch, Dániel Marx, and Magnus Wahlström. Parameterized complexity and kernelizability of max ones and exact ones problems. *TOCT*, 8(1):1, 2016. doi:10.1145/2858787.

25 Stefan Kratsch and Magnus Wahlström. Preprocessing of min ones problems: A dichotomy. In *Proc. 37th ICALP*, volume 6198 of *Lecture Notes in Computer Science*, pages 653–665, 2010. doi:10.1007/978-3-642-14165-2_55.

26 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. Kernelization – preprocessing with a guarantee. In *The Multivariate Algorithmic Revolution and Beyond – Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*, pages 129–161, 2012. doi:10.1007/978-3-642-30891-8_10.

27 Lásló Lovász. Chromatic number of hypergraphs and linear algebra. In *Studia Scientiarum Mathematicarum Hungarica 11*, pages 113–114, 1976.

28 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th ACM Symposium on Theory of Computing*, pages 216–226, 1978. doi:10.1145/800133.804350.

29 Gábor Tardos and David A. Mix Barrington. A lower bound on the mod 6 degree of the OR function. *Computational Complexity*, 7(2):99–108, 1998. doi:10.1007/PL00001597.

Fully Dynamic Data Structure for LCE Queries in Compressed Space

Takaaki Nishimoto[1], Tomohiro I[2], Shunsuke Inenaga[3], Hideo Bannai[4], and Masayuki Takeda[5]

1 **Department of Informatics, Kyushu University, Japan**
takaaki.nishimoto@inf.kyushu-u.ac.jp
2 **Kyushu Institute of Technology, Japan**
tomohiro@ai.kyutech.ac.jp
3 **Department of Informatics, Kyushu University, Japan**
inenaga@inf.kyushu-u.ac.jp
4 **Department of Informatics, Kyushu University, Japan**
bannai@inf.kyushu-u.ac.jp
5 **Department of Informatics, Kyushu University, Japan**
takeda@inf.kyushu-u.ac.jp

Abstract

A Longest Common Extension (LCE) query on a text T of length N asks for the length of the longest common prefix of suffixes starting at given two positions. We show that the signature encoding $\mathcal{G}$ of size $w = O(\min(z \log N \log^* M, N))$ [Mehlhorn et al., Algorithmica 17(2):183-198, 1997] of T, which can be seen as a compressed representation of T, has a capability to support LCE queries in $O(\log N + \log \ell \log^* M)$ time, where ℓ is the answer to the query, z is the size of the Lempel-Ziv77 (LZ77) factorization of T, and $M \geq 4N$ is an integer that can be handled in constant time under word RAM model. In compressed space, this is the fastest deterministic LCE data structure in many cases. Moreover, $\mathcal{G}$ can be enhanced to support efficient update operations: After processing $\mathcal{G}$ in $O(wf_{\mathcal{A}})$ time, we can insert/delete any (sub)string of length y into/from an arbitrary position of T in $O((y + \log N \log^* M)f_{\mathcal{A}})$ time, where $f_{\mathcal{A}} = O(\min\{\frac{\log\log M \log\log w}{\log\log\log M}, \sqrt{\frac{\log w}{\log\log w}}\})$. This yields the first fully dynamic LCE data structure working in compressed space. We also present efficient construction algorithms from various types of inputs: We can construct $\mathcal{G}$ in $O(Nf_{\mathcal{A}})$ time from uncompressed string T; in $O(n \log\log(n \log^* M) \log N \log^* M)$ time from grammar-compressed string T represented by a straight-line program of size n; and in $O(zf_{\mathcal{A}} \log N \log^* M)$ time from LZ77-compressed string T with z factors. On top of the above contributions, we show several applications of our data structures which improve previous best known results on grammar-compressed string processing.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Dynamic Texts, Longest Common Extension (LCE) Queries, Straight-line Program

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.72

1 Introduction

A *Longest Common Extension (LCE)* query on a text T of length N asks to compute the length of the longest common prefix of suffixes starting at given two positions. This fundamental query appears at the heart of many string processing problems (see text book [11] for example), and hence, efficient data structures to answer LCE queries gain a great attention.

A classic solution is to use a data structure for lowest common ancestor queries [4] on the suffix tree of T. Although this achieves constant query time, the $\Theta(N)$ space needed for the data structure is too large to apply it to large scale data. Hence, recent work focuses on reducing space usage at the expense of query time. For example, time-space trade-offs of LCE data structure have been extensively studied [7, 24].

Another direction to reduce space is to utilize a compressed structure of T, which is advantageous when T is highly compressible. There are several LCE data structures working on grammar-compressed string T represented by a straight-line program (SLP) of size n. The best known deterministic LCE data structure is due to I et al. [13], which supports LCE queries in $O(h \log N)$ time, and occupies $O(n^2)$ space, where h is the height of the derivation tree of a given SLP. Their data structure can be built in $O(hn^2)$ time directly from the SLP. Bille et al. [5] showed a Monte Carlo randomized data structure which supports LCE queries in $O(\log N \log \ell)$ time, where ℓ is the output of the LCE query. Their data structure requires only $O(n)$ space, but requires $O(N)$ time to construct. Very recently, Bille et al. [6] showed a faster Monte Carlo randomized data structure of $O(n)$ space which supports LCE queries in $O(\log N + \log^2 \ell)$ time. The preprocessing time of this new data structure is not given in [6]. Note that, given the LZ77-compression of size z of T, we can convert it into an SLP of size $n = O(z \log \frac{N}{z})$ [22] and then apply the above results.

In this paper, we focus on the *signature encoding* $\mathcal{G}$ of T, which can be seen as a grammar compression of T, and show that $\mathcal{G}$ can support LCE queries efficiently. The signature encoding was proposed by Mehlhorn et al. for equality testing on a dynamic set of strings [19]. Alstrup et al. used signature encodings combined with their own data structure called anchors to present a pattern matching algorithm on a dynamic set of strings [2, 1]. In their paper, they also showed that signature encodings can support longest common prefix (LCP) and longest common suffix (LCS) queries on a dynamic set of strings. Their algorithm is randomized as it uses a hash table for maintaining the dictionary of $\mathcal{G}$. Very recently, Gawrychowski et al. improved the results by pursuing advantages of randomized approach other than the hash table [10]. It should be noted that the algorithms in [2, 1, 10] can support LCE queries by combining split operations and LCP queries although it is not explicitly mentioned. However, they did not focus on the fact that signature encodings can work in compressed space. In [9], LCE data structures on edit sensitive parsing, a variant of signature encoding, was used for sparse suffix sorting, but again, they did not focus on working in compressed space.

Our contributions are stated by the following theorems, where $M \geq 4N$ is an integer that can be handled in constant time under word RAM model. More specifically, $M = 4N$ if T is static, and $M/4$ is the upper bound of the length of T if we consider updating T dynamically. In dynamic case, N (resp. w) always denotes the current size of T (resp. $\mathcal{G}$). Also, $f_{\mathcal{A}}$ denotes the time for predecessor/successor queries on a set of w integers from an M-element universe, which is $f_{\mathcal{A}} = O(\min\{\frac{\log\log M \log\log w}{\log\log\log M}, \sqrt{\frac{\log w}{\log\log w}}\})$ by the best known data structure [3].

▶ **Theorem 1** (LCE queries). *Let $\mathcal{G}$ denote the signature encoding of size $w = O(\min(z \log N \log^* M, N))$ for a string T of length N. Then $\mathcal{G}$ supports LCE queries on T in $O(\log N + \log \ell \log^* M)$ time, where ℓ is the answer to the query, and z is the size of the LZ77 factorization of T.*

▶ **Theorem 2** (Updates). *After processing $\mathcal{G}$ in $O(w f_{\mathcal{A}})$ time, we can insert/delete any (sub)string Y of length y into/from an arbitrary position of T in $O((y + \log N \log^* M) f_{\mathcal{A}})$ time. If Y is given as a substring of T, we can support insertion in $O(f_{\mathcal{A}} \log N \log^* M)$ time.*

▶ **Theorem 3** (Construction). *Let T be a string of length N, Z be LZ77 factorization without self reference of size z representing T, and $\mathcal{S}$ be an SLP of size n generating T. Then, we can construct the signature encoding $\mathcal{G}$ for T in (1a) in $O(Nf_{\mathcal{A}})$ time and $O(w)$ working space from T, (1b) in $O(N)$ time and working space from T, (2) in $O(zf_{\mathcal{A}} \log N \log^* M)$ time and $O(w)$ working space from Z, (3a) in $O(nf_{\mathcal{A}} \log N \log^* M)$ time and $O(w)$ working space from $\mathcal{S}$, and (3b) in $O(n \log\log(n \log^* M) \log N \log^* M)$ time and $O(n \log^* M + w)$ working space from $\mathcal{S}$.*

The remarks on our contributions are listed in the following:

- We achieve an algorithm for the fastest deterministic LCE queries on SLPs, which even permits faster LCE queries than the randomized data structure of Bille et al. [6] when $\log^* M = o(\log \ell)$ which in many cases is true.
- We present the first fully dynamic LCE data structure working in compressed space.
- Different from the work in [2, 1, 10], we mainly focus on maintaining a single text T in compressed $O(w)$ space. For this reason we opt for supporting insertion/deletion as edit operations rather than split/concatenate on a dynamic set of strings. However, the difference is not much essential; our insert operations specified by a substring of an existing string can work as split/concatenate, and conversely, split/concatenate can simulate insert. Our contribution here is to clarify how to collect garbage being produced during edit operations, as directly indicated by a support of delete operations.
- The results (2) and (3a) of Theorem 3 immediately follow from the update operations considered in [2, 1], but others are nontrivial.
- Direct construction of $\mathcal{G}$ from SLPs is important for applications in compressed string processing, where the task is to process a given compressed representation of string(s) without explicit decompression. In particular, we use the result (3b) of Theorem 3 to show several applications which improve previous best known results. Note that the time complexity of the result (3b) can be written as $O(n \log\log n \log N \log^* M)$ when $\log^* M = O(n)$ which in many cases is true, and always true in static case because $\log^* M = O(\log^* N) = O(\log N) = O(n)$.

Proofs and examples omitted due to lack of space are in a full version of this paper [21].

2 Preliminaries

2.1 Strings

Let Σ be an ordered alphabet. An element of Σ^* is called a string. For string $w = xyz$, x, y and z are called a prefix, substring, and suffix of w, respectively. The length of string w is denoted by $|w|$. The empty string ε is a string of length 0. Let $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. For any $1 \leq i \leq |w|$, $w[i]$ denotes the i-th character of w. For any $1 \leq i \leq j \leq |w|$, $w[i..j]$ denotes the substring of w that begins at position i and ends at position j. Let $w[i..] = w[i..|w|]$ and $w[..i] = w[1..i]$ for any $1 \leq i \leq |w|$. For any string w, let w^R denote the reversed string of w, that is, $w^R = w[|w|] \cdots w[2]w[1]$. For any strings w and u, let $\mathsf{LCP}(w, u)$ (resp. $\mathsf{LCS}(w, u)$) denote the length of the longest common prefix (resp. suffix) of w and u. Given two strings s_1, s_2 and two integers i, j, let $\mathsf{LCE}(s_1, s_2, i, j)$ denote a query which returns $\mathsf{LCP}(s_1[i..|s_1|], s_2[j..|s_2|])$. Our model of computation is the unit-cost word RAM with machine word size of $\Omega(\log_2 M)$ bits, and space complexities will be evaluated by the number of machine words. Bit-oriented evaluation of space complexities can be obtained with a $\log_2 M$ multiplicative factor.

▶ **Definition 4** (Lempel-Ziv77 factorization [25]). The Lempel-Ziv77 (LZ77) factorization of a string s without self-references is a sequence $f_1, \ldots, f_z$ of non-empty substrings of s such that $s = f_1 \cdots f_z$, $f_1 = s[1]$, and for $1 < i \leq z$, if the character $s[|f_1..f_{i-1}| + 1]$ does not occur in $s[|f_1..f_{i-1}|]$, then $f_i = s[|f_1..f_{i-1}| + 1]$, otherwise f_i is the longest prefix of $f_i \cdots f_z$ which occurs in $f_1 \cdots f_{i-1}$. The size of the LZ77 factorization $f_1, \ldots, f_z$ of string s is the number z of factors in the factorization.

2.2 Context free grammars as compressed representation of strings

Straight-line programs. A *straight-line program* (*SLP*) is a context free grammar in the Chomsky normal form that generates a single string. Formally, an SLP that generates T is a quadruple $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{D}, S)$, such that Σ is an ordered alphabet of terminal characters; $\mathcal{V} = \{X_1, \ldots, X_n\}$ is a set of positive integers, called *variables*; $\mathcal{D} = \{X_i \rightarrow \mathit{expr}_i\}_{i=1}^{n}$ is a set of *deterministic productions* (or *assignments*) with each expr_i being either of form $X_\ell X_r$ $(1 \leq \ell, r < i)$, or a single character $a \in \Sigma$; and $S := X_n \in \mathcal{V}$ is the start symbol which derives the string T. We also assume that the grammar neither contains *redundant* variables (i.e., there is at most one assignment whose righthand side is *expr*) nor *useless* variables (i.e., every variable appears at least once in the derivation tree of $\mathcal{G}$). The *size* of the SLP $\mathcal{G}$ is the number n of productions in $\mathcal{D}$. In the extreme cases the length N of the string T can be as large as 2^{n-1}, however, it is always the case that $n \geq \log_2 N$.

Let $\mathit{val} : \mathcal{V} \rightarrow \Sigma^+$ be the function which returns the string derived by an input variable. If $s = \mathit{val}(X)$ for $X \in \mathcal{V}$, then we say that the variable X *represents* string s. For any variable sequence $y \in \mathcal{V}^+$, let $\mathit{val}^+(y) = \mathit{val}(y[1]) \cdots \mathit{val}(y[|y|])$.

Run-length straight-line programs. We define *run-length SLPs* (*RLSLPs*), as an extension to SLPs, which allow *run-length encodings* in the righthand sides of productions, i.e., $\mathcal{D}$ might contain a production $X \rightarrow \hat{X}^k \in \mathcal{V} \times \mathcal{N}$. The *size* of the RLSLP is still the number of productions in $\mathcal{D}$ as each production can be encoded in constant space. Let $\mathit{Assgn}_\mathcal{G}$ be the function such that $\mathit{Assgn}_\mathcal{G}(X_i) = \mathit{expr}_i$ iff $X_i \rightarrow \mathit{expr}_i \in \mathcal{D}$. Also, let $\mathit{Assgn}_\mathcal{G}^{-1}$ denote the reverse function of $\mathit{Assgn}_\mathcal{G}$. When clear from the context, we write $\mathit{Assgn}_\mathcal{G}$ and $\mathit{Assgn}_\mathcal{G}^{-1}$ as Assgn and Assgn^{-1}, respectively.

Representation of RLSLPs. For an RLSLP $\mathcal{G}$ of size w, we can consider a DAG of size w as a compact representation of the derivation trees of variables in $\mathcal{G}$. Each node represents a variable X in $\mathcal{V}$ and store $|\mathit{val}(X)|$ and out-going edges represent the assignments in $\mathcal{D}$: For an assignment $X_i \rightarrow X_\ell X_r \in \mathcal{D}$, there exist two out-going edges from X_i to its ordered children X_ℓ and X_r; and for $X \rightarrow \hat{X}^k \in \mathcal{D}$, there is a single edge from X to $\hat{X}$ with the multiplicative factor k.

3 Signature encoding

Here, we recall the *signature encoding* first proposed by Mehlhorn et al. [19]. Its core technique is *locally consistent parsing* defined as follows:

▶ **Lemma 5** (Locally consistent parsing [19, 1]). *Let W be a positive integer. There exists a function $f : [0..W]^{\log^* W + 11} \rightarrow \{0, 1\}$ such that, for any $p \in [1..W]^n$ with $n \geq 2$ and $p[i] \neq p[i+1]$ for any $1 \leq i < n$, the bit sequence d defined by $d[i] = f(\tilde{p}[i - \Delta_L], \ldots, \tilde{p}[i + \Delta_R])$ for $1 \leq i \leq n$, satisfies: $d[1] = 1$; $d[n] = 0$; $d[i] + d[i+1] \leq 1$ for $1 \leq i < n$; and $d[i] + d[i+1] + d[i+2] + d[i+3] \geq 1$ for any $1 \leq i < n - 3$; where $\Delta_L = \log^* W + 6$, $\Delta_R = 4$,* *and $\tilde{p}[j] = p[j]$ for all $1 \leq j \leq n$, $\tilde{p}[j] = 0$ otherwise. Furthermore, we can compute d in $O(n)$ time using a precomputed table of size $o(\log W)$, which can be computed in $o(\log W)$ time.*

For the bit sequence d of Lemma 5, we define the function $\mathit{Eblock}_d(p)$ that decomposes an integer sequence p according to d: $\mathit{Eblock}_d(p)$ decomposes p into a sequence $q_1, \ldots, q_j$ of substrings called *blocks* of p, such that $p = q_1 \cdots q_j$ and q_i is in the decomposition iff $d[|q_1 \cdots q_{i-1}| + 1] = 1$ for any $1 \leq i \leq j$. Note that each block is of length from two to four by the property of d, i.e., $2 \leq |q_i| \leq 4$ for any $1 \leq i \leq j$. Let $|\mathit{Eblock}_d(p)| = j$ and let $\mathit{Eblock}_d(s)[i] = q_i$. We omit d and write $\mathit{Eblock}(p)$ when it is clear from the context, and we use implicitly the bit sequence created by Lemma 5 as d.

We complementarily use run-length encoding to get a sequence to which *Eblock* can be applied. Formally, for a string s, let $\mathit{Epow}(s)$ be the function which groups each maximal run of same characters a as a^k, where k is the length of the run. $\mathit{Epow}(s)$ can be computed in $O(|s|)$ time. Let $|\mathit{Epow}(s)|$ denote the number of maximal runs of same characters in s and let $\mathit{Epow}(s)[i]$ denote i-th maximal run in s.

The signature encoding is the RLSLP $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{D}, S)$, where the assignments in $\mathcal{D}$ are determined by recursively applying *Eblock* and *Epow* to T until a single integer S is obtained. We call each variable of the signature encoding a *signature*, and use e (for example, $e_i \rightarrow e_\ell e_r \in \mathcal{D}$) instead of X to distinguish from general RLSLPs.

For a formal description, let $E := \Sigma \cup \mathcal{V}^2 \cup \mathcal{V}^3 \cup \mathcal{V}^4 \cup (\mathcal{V} \times \mathcal{N})$ and let $\mathit{Sig} : E \rightarrow \mathcal{V}$ be the function such that: $\mathit{Sig}(x) = e$ if $(e \rightarrow x) \in \mathcal{D}$; $\mathit{Sig}(x) = \mathit{Sig}(\mathit{Sig}(x[1..|x| - 1])x[|x|])$ if $x \in \mathcal{V}^3 \cup \mathcal{V}^4$; or otherwise undefined. Namely, the function *Sig* returns, if any, the lefthand side of the corresponding production of x by recursively applying the Assgn^{-1} function from left to right. For any $p \in E^*$, let $\mathit{Sig}^+(p) = \mathit{Sig}(p[1]) \cdots \mathit{Sig}(p[|p|])$.

The signature encoding of string T is defined by the following *Shrink* and *Pow* functions: $\mathit{Shrink}^T_t = \mathit{Sig}^+(T)$ for $t = 0$, and $\mathit{Shrink}^T_t = \mathit{Sig}^+(\mathit{Eblock}(\mathit{Pow}^T_{t-1}))$ for $0 < t \leq h$; and $\mathit{Pow}^T_t = \mathit{Sig}^+(\mathit{Epow}(\mathit{Shrink}^T_t))$ for $0 \leq t \leq h$; where h is the minimum integer satisfying $|\mathit{Pow}^T_h| = 1$. Then, the start symbol of the signature encoding is $S = \mathit{Pow}^T_h$. We say that a node is in *level* t in the derivation tree of S if the node is produced by Shrink^T_t or Pow^T_t. The height of the derivation tree of the signature encoding of T is $O(h) = O(\log |T|)$. For any $T \in \Sigma^+$, let $\mathit{id}(T) = \mathit{Pow}^T_h = S$, i.e., the integer S is the signature of T.

In this paper, we implement signature encodings by the DAG of RLSLP introduced in Section 2.

4 Compressed LCE data structure using signature encodings

In this section, we show Theorem 1.

Space requirement of the signature encoding. It is clear from the definition of the signature encoding $\mathcal{G}$ of T that the size of $\mathcal{G}$ is less than $4N \leq M$, and hence, all signatures are in $[1..M - 1]$. Moreover, the next lemma shows that $\mathcal{G}$ requires only *compressed space*:

▶ **Lemma 6** ([23]). *The size w of the signature encoding of T of length N is $O(z \log N \log^* M)$, where z is the number of factors in the LZ77 factorization without self-reference of T.*

Common sequences of signatures to all occurrences of same substrings. Here, we recall the most important property of the signature encoding, which ensures the existence of common signatures to all occurrences of same substrings by the following lemma.

▶ **Lemma 7** (common sequences [23]). *Let $\mathcal{G}$ be a signature encoding for a string T. Every substring P in T is represented by a signature sequence $Uniq(P)$ in $\mathcal{G}$ for a string P.*

$Uniq(P)$, which we call the *common sequence* of P, is defined by the following.

▶ **Definition 8.** For a string P, let

$$XShrink^P_t = \begin{cases} Sig^+(P) & \text{for } t = 0, \\ Sig^+(Eblock_d(XPow^P_{t-1})[|L^P_t|..|XPow^P_{t-1}| - |R^P_t|]) & \text{for } 0 < t \leq h^P, \end{cases}$$

$$XPow^P_t = Sig^+(Epow(XShrink^P_t[|\hat{L}^P_t| + 1..|XShrink^P_t| - |\hat{R}^P_t|])|) \text{ for } 0 \leq t < h^P,$$

- L^P_t is the shortest prefix of $XPow^P_{t-1}$ of length at least Δ_L such that $d[|L^P_t| + 1] = 1$,
- R^P_t is the shortest suffix of $XPow^P_{t-1}$ of length at least Δ_R+1 such that $d[|d|-|R^P_t|+1] = 1$,
- $\hat{L}^P_t$ is the longest prefix of $XShrink^P_t$ such that $|Epow(\hat{L}^P_t)| = 1$,
- $\hat{R}^P_t$ is the longest suffix of $XShrink^P_t$ such that $|Epow(\hat{R}^P_t)| = 1$, and
- h^P is the minimum integer such that $|Epow(XShrink^P_{h^P})| \leq \Delta_L + \Delta_R + 9$.

Note that $\Delta_L \leq |L^P_t| \leq \Delta_L + 3$ and $\Delta_R + 1 \leq |R^P_t| \leq \Delta_R + 4$ hold by the definition. Hence $|XShrink^P_{t+1}| > 0$ holds if $|Epow(XShrink^P_t)| > \Delta_L + \Delta_R + 9$. Then,

$$Uniq(P) = \hat{L}^P_0 L^P_0 \cdots \hat{L}^P_{h^P-1} L^P_{h^P-1} XShrink^P_{h^P} R^P_{h^P-1} \hat{R}^P_{h^P-1} \cdots R^P_0 \hat{R}^P_0.$$

We give an intuitive description of Lemma 7. Recall the locally consistent parsing of Lemma 5. Each i-th bit of bit sequence d of Lemma 5 for a given string s is determined by $s[i - \Delta_L..i + \Delta_R]$. Hence, for two positions i, j such that $P = s[i..i + k - 1] = s[j..j + k - 1]$ for some k, $d[i + \Delta_L..i + k - 1 - \Delta_R] = d[j + \Delta_L..j + k - 1 - \Delta_R]$ holds, namely, "internal" bit sequences of the same substring of s are equal. Since each level of the signature encoding uses the bit sequence, all occurrences of same substrings in a string share same internal signature sequences, and this goes up level by level. $XShrink^P_t$ and $XPow^P_t$ represent signature sequences obtained from only internal signature sequences of $XPow^T_{t-1}$ and $XShrink^T_t$, respectively. This means that $XShrink^P_t$ and $XPow^P_t$ are always created over P. From such common signatures we take as short signature sequence as possible for $Uniq(P)$: Since $val^+(Pow^P_{t-1}) = val^+(L^P_{t-1} XShrink^P_t R^P_{t-1})$ and $val^+(Shrink^P_t) = val^+(\hat{L}^P_t XPow^P_t \hat{R}^P_t)$ hold, $|Epow(Uniq(P))| = O(\log |P| \log^* M)$ and $val^+(Uniq(P)) = P$ hold. Hence Lemma 7 holds [1].

The number of ancestors of nodes corresponding to $Uniq(P)$ is upper bounded by:

▶ **Lemma 9.** *Let $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{D}, S)$ be a signature encoding for a string T, P be a string, and let $\mathcal{T}$ be the derivation tree of a signature $e \in \mathcal{V}$. Consider an occurrence of P in s, and the induced subtree X of $\mathcal{T}$ whose root is the root of $\mathcal{T}$ and whose leaves are the parents of the nodes representing $Uniq(P)$, where $s = val(e)$. Then X contains $O(\log^* M)$ nodes for every level and $O(\log |s| + \log |P| \log^* M)$ nodes in total.*

LCE queries. In the next lemma, we show a more general result than Theorem 1, which states that the signature encoding supports (both forward and backward) LCE queries on a given arbitrary pair of signatures. Theorem 1 immediately follows from Lemma 10.

▶ **Lemma 10.** *Using a signature encoding $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{D}, S)$ for a string T, we can support queries $\mathsf{LCE}(s_1, s_2, i, j)$ and $\mathsf{LCE}(s^R_1, s^R_2, i, j)$ in $O(\log |s_1| + \log |s_2| + \log \ell \log^* M)$ time for given two signatures $e_1, e_2 \in \mathcal{V}$ and two integers $1 \leq i \leq |s_1|$, $1 \leq j \leq |s_2|$, where $s_1 = val(e_1)$, $s_2 = val(e_2)$ and ℓ is the answer to the LCE query.*

[1] The common sequences are conceptually equivalent to the *cores* [17] which are defined for the *edit sensitive parsing* of a text, a kind of locally consistent parsing of the text.

Proof. We focus on $\mathsf{LCE}(s_1, s_2, i, j)$ as $\mathsf{LCE}(s_1^R, s_2^R, i, j)$ is supported similarly.

Let P denote the longest common prefix of $s_1[i..]$ and $s_2[j..]$. Our algorithm simultaneously traverses two derivation trees rooted at e_1 and e_2 and computes P by matching the common signatures greedily from left to right. Recall that s_1 and s_2 are substrings of T. Since the both substrings P occurring at position i in $val(e_1)$ and at position j in $val(e_2)$ are represented by $Uniq(P)$ in the signature encoding by Lemma 7, we can compute P by at least finding the common sequence of nodes which represents $Uniq(P)$, and hence, we only have to traverse ancestors of such nodes. By Lemma 9, the number of nodes we traverse, which dominates the time complexity, is upper bounded by $O(\log |s_1| + \log |s_2| + Epow(Uniq(P))) = O(\log |s_1| + \log |s_2| + \log \ell \log^* M)$. ◀

5 Updates

In this section, we show Theorem 2. Formally, we consider a *dynamic signature encoding* $\mathcal{G}$ of T, which allows for efficient updates of $\mathcal{G}$ in compressed space according to the following operations: $INSERT(Y, i)$ inserts a string Y into T at position i, i.e., $T \leftarrow T[..i-1]YT[i..]$; $INSERT'(j, y, i)$ inserts $T[j..j+y-1]$ into T at position i, i.e., $T \leftarrow T[..i-1]T[j..j+y-1]T[i..]$; and $DELETE(j, y)$ deletes a substring of length y starting at j, i.e., $T \leftarrow T[..j-1]T[j+y..]$.

During updates we recompute $Shrink_t^T$ and Pow_t^T for some part of new T (note that the most part is unchanged thanks to the virtue of signature encodings, Lemma 9). When we need a signature for $expr$, we look up the signature assigned to $expr$ (i.e., compute $Assign^{-1}(expr)$) and use it if such exists. If $Assign^{-1}(expr)$ is undefined we create a new signature, which is an integer that is currently not used as signatures (say $e_{new} = \min([1..M] \setminus \mathcal{V})$), and add $e_{new} \rightarrow expr$ to $\mathcal{D}$. Also, updates may produce a useless signature whose parents in the DAG are all removed. We remove such useless signatures from $\mathcal{G}$ during updates.

Note that the corresponding nodes and edges of the DAG can be added/removed in constant time per addition/removal of an assignment. In addition to the DAG, we need dynamic data structures to conduct the following operations efficiently: (A) computing $Assgn^{-1}(\cdot)$, (B) computing $\min([1..M] \setminus \mathcal{V})$, and (C) checking if a signature e is useless.

For (A), we use Beame and Fich's data structure [3] that can support predecessor/successor queries on a dynamic set of integers.[2] For example, we consider Beame and Fich's data structure maintaining a set of integers $\{e_\ell M^2 + e_r M + e \mid e \rightarrow e_\ell e_r \in \mathcal{D}\}$ in $O(w)$ space. Then we can implement $Assgn^{-1}(e_\ell e_r)$ by computing the successor q of $e_\ell M^2 + e_r M$, i.e., $e = q \bmod M$ if $\lfloor q/M \rfloor = e_\ell M + e_r$, and otherwise $Assgn^{-1}(e_\ell e_r)$ is undefined. Queries as well as update operations can be done in deterministic $O(f_\mathcal{A})$ time, where $f_\mathcal{A} = O\left(\min\left\{\frac{\log\log M \log\log w}{\log\log\log M}, \sqrt{\frac{\log w}{\log\log w}}\right\}\right)$.

For (B), we again use Beame and Fich's data structure to maintain the set of maximal intervals such that every element in the intervals is signature. Formally, the intervals are maintained by a set of integers $\{e_i M + e_j \mid [e_i..e_j] \subseteq \mathcal{V}, e_i - 1 \notin \mathcal{V}, e_j + 1 \notin \mathcal{V}\}$ in $O(w)$ space. Then we can know the minimum integer currently not in $\mathcal{V}$ by computing the successor of 0.

For (C), we let every signature $e \in \mathcal{V}$ have a counter to count the number of parents of e in the DAG. Then we can know that a signature is useless if the counter is 0.

Lemma 11 shows that we can efficiently compute $Uniq(P)$ for a substring P of T.

[2] Alstrup et al. [1] used hashing for this purpose. However, since we are interested in the worst case time complexities, we use the data structure [3] in place of hashing.

▶ **Lemma 11.** *Using a signature encoding $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{D}, S)$ of size w, given a signature $e \in \mathcal{V}$ (and its corresponding node in the DAG) and two integers j and y, we can compute $Epow(Uniq(s[j..j+y-1]))$ in $O(\log|s| + \log y \log^* M)$ time, where $s = val(e)$.*

Proof of Theorem 2. It is easy to see that, given the static signature encoding of T, we can construct data structures (A)-(C) in $O(wf_{\mathcal{A}})$ time. After constructing these, we can add/remove an assignment in $O(f_{\mathcal{A}})$ time.

Let $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{D}, S)$ be the signature encoding before the update operation. We support $DELETE(j, y)$ as follows: (1) Compute the new start variable $S' = id(T[..j-1]T[j+y..])$ by recomputing the new signature encoding from $Uniq(T[..j-1])$ and $Uniq(T[j+y..])$. Although we need a part of d to recompute $Eblock_d(Pow_t^{T[..j-1]T[j+y..]})$ for every level t, the input size to compute the part of d is $O(\log^* M)$ by Lemma 5. Hence these can be done in $O(f_{\mathcal{A}} \log N \log^* M)$ time by Lemmas 11 and 9. (2) Remove all useless signatures Z from $\mathcal{G}$. Note that if a signature is useless, then all the signatures along the path from S to it are also useless. Hence, we can remove all useless signatures efficiently by depth-first search starting from S, which takes $O(f_{\mathcal{A}}|Z|)$ time, where $|Z| = O(y + \log N \log^* M)$ by Lemma 9.

Similarly, we can support $INSERT(Y, i)$ in $O(f_{\mathcal{A}}(y + \log N \log^* M))$ time by creating the new start variable S' from $Uniq(T[..i-1])$, $Uniq(Y)$ and $Uniq(T[i..])$. Note that we can naively compute $Uniq(Y)$ in $O(f_{\mathcal{A}}y)$ time. For $INSERT'(j, y, i)$, we can avoid $O(f_{\mathcal{A}}y)$ time by computing $Uniq(T[j..j+y-1])$ using Lemma 11. ◀

6 Construction

In this section, we give proofs of Theorem 3, but we omit proofs of the results (2) and (3a) as they are straightforward from the previous work [2, 1].

6.1 Theorem 3 (1a)

Proof of Theorem 3 (1a). Note that we can naively compute $id(T)$ for a given string T in $O(Nf_{\mathcal{A}})$ time and $O(N)$ working space. In order to reduce the working space, we consider factorizing T into blocks of size B and processing them incrementally: Starting with the empty signature encoding $\mathcal{G}$, we can compute $id(T)$ in $O(\frac{N}{B} f_{\mathcal{A}}(\log N \log^* M + B))$ time and $O(w + B)$ working space by using $INSERT(T[(i-1)B+1..iB], (i-1)B+1)$ for $i = 1, \ldots, \frac{N}{B}$ in increasing order. Hence our proof is finished by choosing $B = \log N \log^* M$. ◀

6.2 Theorem 3 (1b)

We compute signatures level by level, i.e., construct $Shrink_0^T, Pow_0^T, \ldots, Shrink_h^T, Pow_h^T$ incrementally. For each level, we create signatures by sorting signature blocks (or run-length encoded signatures) to which we give signatures, as shown by the next two lemmas.

▶ **Lemma 12.** *Given $Eblock(Pow_{t-1}^T)$ for $0 < t \leq h$, we can compute $Shrink_t^T$ in $O((b-a) + |Pow_{t-1}^T|)$ time and space, where b is the maximum integer in Pow_{t-1}^T and a is the minimum integer in Pow_{t-1}^T.*

Proof. Since we assign signatures to signature blocks and run-length signatures in the derivation tree of S in the order they appear in the signature encoding. $Pow_{t-1}^T[i] - a$ fits in an entry of a bucket of size $b - a$ for each element of $Pow_{t-1}^T[i]$ of Pow_{t-1}^T. Also, the length of each block is at most four. Hence we can sort all the blocks of $Eblock(Pow_{t-1}^T)$ by bucket sort in $O((b-a) + |Pow_{t-1}^T|)$ time and space. Since Sig is an injection and since we process the levels in increasing order, for any two different levels $0 \leq t' < t \leq h$, no elements of $Shrink^T_{t-1}$ appear in $Shrink^T_{t'-1}$, and hence no elements of Pow^T_{t-1} appear in $Pow^T_{t'-1}$. Thus, we can determine a new signature for each block in $Eblock(Pow^T_{t-1})$, *without* searching existing signatures in the lower levels. This completes the proof. ◀

▶ **Lemma 13.** *Given $Epow(Shrink^T_t)$, we can compute Pow^T_t in $O(x + (b - a) + |Epow(Shrink^T_t)|)$ time and space, where x is the maximum length of runs in $Epow(Shrink^T_t)$, b is the maximum integer in Pow^T_{t-1}, and a is the minimum integer in Pow^T_{t-1}.*

Proof. We first sort all the elements of $Epow(Shrink^T_t)$ by bucket sort in $O(b - a + |Epow(Shrink^T_t)|)$ time and space, ignoring the powers of runs. Then, for each integer r appearing in $Shrink^T_t$, we sort the runs of r's by bucket sort with a bucket of size x. This takes a total of $O(x + |Epow(Shrink^T_t)|)$ time and space for all integers appearing in $Shrink^T_t$. The rest is the same as the proof of Lemma 12. ◀

Proof of Theorem 3 (1b). Since the size of the derivation tree of $id(T)$ is $O(N)$, by Lemmas 5, 12, and 13, we can compute a DAG of $\mathcal{G}$ for T in $O(N)$ time and space. ◀

6.3 Theorem 3 (3b)

In this section, we sometimes abbreviate $val(X)$ as X for $X \in \mathcal{S}$. For example, $Shrink^X_t$ and Pow^X_t represents $Shrink^{val(X)}_t$ and $Pow^{val(X)}_t$ respectively.

Our algorithm computes signatures level by level, i.e., constructs incrementally $Shrink^{X_n}_0$, $Pow^{X_n}_0$, $\ldots$, $Shrink^{X_n}_h$, $Pow^{X_n}_h$. Like the algorithm described in Section 6.2, we can create signatures by sorting blocks of signatures or run-length encoded signatures in the same level. The main difference is that we now utilize the structure of the SLP, which allows us to do the task efficiently in $O(n \log^* M + w)$ working space. In particular, although $|Shrink^{X_n}_t|, |Pow^{X_n}_t| = O(N)$ for $0 \le t \le h$, they can be represented in $O(n \log^* M)$ space.

In so doing, we introduce some additional notations relating to $XShrink^P_t$ and $XPow^P_t$ in Definition 8. By Lemma 7, there exist $\hat{z}^{(P_1,P_2)}_t$ and $z^{(P_1,P_2)}_t$ for any string $P = P_1P_2$ such that the following equation holds: $XShrink^P_t = \hat{y}^{P_1}_t \hat{z}^{(P_1,P_2)}_t \hat{y}^{P_2}_t$ for $0 < t \le h^P$, and $XPow^P_t = y^{P_1}_t z^{(P_1,P_2)}_t y^{P_2}_t$ for $0 \le t < h^P$, where we define $\hat{y}^P_t$ and y^P_t for a string P as:

$$\hat{y}^P_t = \begin{cases} XShrink^P_t & \text{for } 0 < t \le h^P, \\ \varepsilon & \text{for } t > h^P, \end{cases} \qquad y^P_t = \begin{cases} XPow^P_t & \text{for } 0 \le t < h^P, \\ \varepsilon & \text{for } t \ge h^P. \end{cases}$$

For any variable $X_i \to X_\ell X_r$, we denote $\hat{z}^{X_i}_t = \hat{z}^{(val(X_\ell),val(X_r))}_t$ (for $0 < t \le h^{val(X_i)}$) and $z^{X_i}_t = z^{(val(X_\ell),val(X_r))}_t$ (for $0 \le t < h^{val(X_i)}$). Note that $|z^{X_i}_t|, |\hat{z}^{X_i}_t| = O(\log^* M)$ because $z^{X_i}_t$ is created on $\hat{R}^{X_\ell}_t \hat{z}^{X_i}_t \hat{L}^{X_r}_t$, similarly, $\hat{z}^{X_i}_t$ is created on $R^{X_\ell}_{t-1} z^{X_i}_{t-1} L^{X_r}_{t-1}$. We can use $\hat{z}^{X_1}_t, \ldots, \hat{z}^{X_n}_t$ (resp. $z^{X_1}_t, \ldots, z^{X_n}_t$) as a compressed representation of $XShrink^{X_n}_t$ (resp. $XPow^{X_n}_t$) based on the SLP: Intuitively, $\hat{z}^{X_n}_t$ (resp. $z^{X_n}_t$) covers the middle part of $XShrink^{X_n}_t$ (resp. $XPow^{X_n}_t$) and the remaining part is recovered by investigating the left/right child recursively (see also Fig. 1). Hence, with the DAG structure of the SLP, $XShrink^{X_n}_t$ and $XPow^{X_n}_t$ can be represented in $O(n \log^* M)$ space.

In addition, we define $\hat{A}^P_t$, $\hat{B}^P_t$, A^P_t and B^P_t as follows: For $0 < t \le h^P$, $\hat{A}^P_t$ (resp. $\hat{B}^P_t$) is a prefix (resp. suffix) of $Shrink^P_t$ which consists of signatures of $A^P_{t-1} L^P_{t-1}$ (resp. $R^P_{t-1} B^P_{t-1}$); and for $0 \le t < h^P$, A^P_t (resp. B^P_t) is a prefix (resp. suffix) of Pow^P_t which consists of signatures of $\hat{A}^P_t \hat{L}^P_t$ (resp. $\hat{R}^P_t \hat{B}^P_t$). By the definition, $Shrink^P_t = \hat{A}^P_t XShrink^P_t \hat{B}^P_t$ for $0 \le t \le h^P$, and $Pow^P_t = A^P_t XPow^P_t B^P_t$ for $0 \le t < h^P$. See Fig. 2 for the illustration.

Since $Shrink^{X_n}_t = \hat{A}^{X_n}_t XShrink^{X_n}_t \hat{B}^{X_n}_t$ for $0 < t \le h^{X_n}$, we use $\hat{\Lambda}_t = (\hat{z}^{X_1}_t, \ldots, \hat{z}^{X_n}_t, \hat{A}^{X_n}_t, \hat{B}^{X_n}_t)$ as a compressed representation of $Shrink^{X_n}_t$ of size $O(n \log^* M)$. Similarly, for $0 \le$

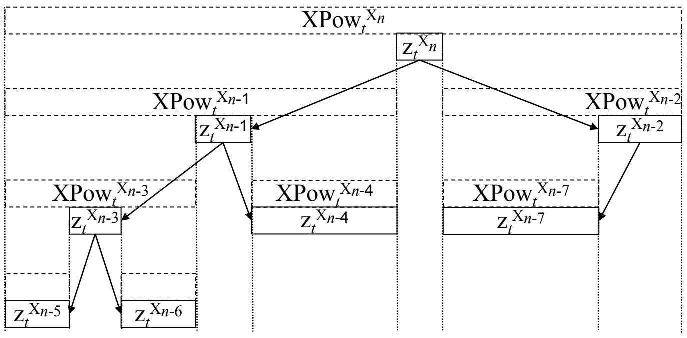


Figure 1 $XPow_t^{X_n}$ can be represented by $z_t^{X_1}, \ldots, z_t^{X_n}$. In this example, $XPow_t^{X_n} = z_t^{X_{n-5}} z_t^{X_{n-3}} z_t^{X_{n-6}} z_t^{X_{n-1}} z_t^{X_{n-4}} z_t^{X_n} z_t^{X_{n-7}} z_t^{X_{n-2}}$.

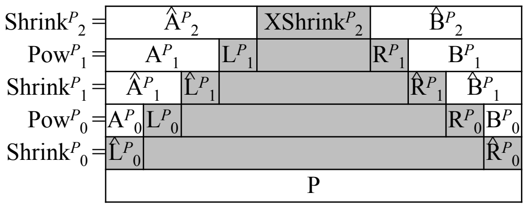


Figure 2 An abstract image of $Shrink_t^P$ and Pow_t^P for a string P. For $0 \leq t < h^P$, $A_t^P L_t^P$ (resp. $R_t^P B_t^P$) is encoded into $\hat{A}_{t+1}^P$ (resp. $\hat{B}_{t+1}^P$). Similarly, for $0 < t < h^P$, $\hat{A}_t^P \hat{L}_t^P$ (resp. $\hat{R}_t^P \hat{B}_t^P$) is encoded into A_t^P (resp. B_t^P).

$t < h^{X_n}$, we use $\Lambda_t = (z_t^{X_1}, \ldots, z_t^{X_n}, A_t^{X_n}, B_t^{X_n})$ as a compressed representation of $Pow_t^{X_n}$ of size $O(n \log^* M)$.

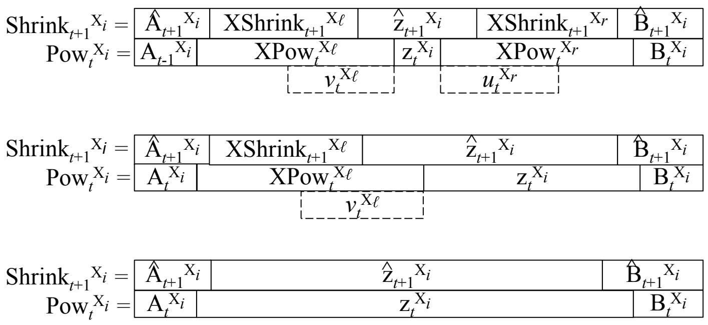
Our algorithm computes incrementally $\Lambda_0, \hat{\Lambda}_1, \ldots, \hat{\Lambda}_{h^{X_n}}$. Given $\hat{\Lambda}_{h^{X_n}}$, we can easily get $Pow_{h^{X_n}}^{X_n}$ of size $O(\log^* M)$ in $O(n \log^* M)$ time, and then $id(val(X_n))$ in $O(\log^* M)$ time from $Pow_{h^{X_n}}^{X_n}$. Hence, in the following three lemmas, we show how to compute $\Lambda_0, \hat{\Lambda}_1, \ldots, \hat{\Lambda}_{h^{X_n}}$.

▶ **Lemma 14.** *Given an SLP of size n, we can compute Λ_0 in $O(n \log\log(n \log^* M) \log^* M)$ time and $O(n \log^* M)$ space.*

Proof. We first compute, for all variables X_i, $Epow(XShrink_0^{X_i})$ if $|Epow(XShrink_0^{X_i})| \leq \Delta_L + \Delta_R + 9$, otherwise $Epow(\hat{L}_0^{X_i})$ and $Epow(\hat{R}_0^{X_i})$. The information can be computed in $O(n \log^* M)$ time and space in a bottom-up manner, i.e., by processing variables in increasing order. For $X_i \to X_\ell X_r$, if both $|Epow(XShrink_0^{X_\ell})|$ and $|Epow(XShrink_0^{X_r})|$ are no greater than $\Delta_L + \Delta_R + 9$, we can compute $Epow(XShrink_0^{X_i})$ in $O(\log^* M)$ time by naively concatenating $Epow(XShrink_0^{X_\ell})$ and $Epow(XShrink_0^{X_r})$. Otherwise $|Epow(XShrink_0^{X_i})| > \Delta_L + \Delta_R + 9$ must hold, and $Epow(\hat{L}_0^{X_i})$ and $Epow(\hat{R}_0^{X_i})$ can be computed in $O(1)$ time from the information for X_ℓ and X_r.

The run-length encoded signatures represented by $z_0^{X_i}$ can be obtained by using the above information for X_ℓ and X_r in $O(\log^* M)$ time: $z_0^{X_i}$ is created over run-length encoded signatures $Epow(XShrink_0^{X_\ell})$ (or $Epow(\hat{R}_0^{X_\ell})$) followed by $Epow(XShrink_0^{X_r})$ (or $Epow(\hat{R}_0^{X_r})$). Also, by definition $A_0^{X_n}$ and $B_0^{X_n}$ represents $Epow(\hat{L}_0^{X_n})$ and $Epow(\hat{R}_0^{X_n})$, respectively.

Hence, we can compute in $O(n \log^* M)$ time $O(n \log^* M)$ run-length encoded signatures to which we give signatures. We determine signatures by sorting the run-length encoded signatures as Lemma 13. However, in contrast to Lemma 13, we do not use bucket sort for sorting the powers of runs because the maximum length of runs could be as large as N and we cannot afford $O(N)$ space for buckets. Instead, we use the sorting algorithm of Han [12]



■ **Figure 3** Abstract images of the needed signature sequence $v_t^{X_\ell} z_t^{X_i} u_t^{X_r}$ ($v_t^{X_\ell}$ and $u_t^{X_r}$ are not shown when they are empty) for computing $\hat{z}_{t+1}^{X_i}$ in three situations: Top for $0 \le t < h^{X_\ell}, h^{X_r}$; middle for $h^{X_r} \le t < h^{X_\ell}$; and bottom for $h^{X_\ell}, h^{X_r} \le t < h^{X_i}$.

which sorts x integers in $O(x \log\log x)$ time and $O(x)$ space. Hence, we can compute Λ_0 in $O(n \log\log(n \log^* M) \log^* M)$ time and $O(n \log^* M)$ space. ◀

▶ **Lemma 15.** *Given $\hat{\Lambda}_t$, we can compute Λ_t in $O(n \log\log(n \log^* M) \log^* M)$ time and $O(n \log^* M)$ space.*

Proof. The computation is similar to that of Lemma 14 except that we also use $\hat{\Lambda}_t$. ◀

▶ **Lemma 16.** *Given Λ_t, we can compute $\hat{\Lambda}_{t+1}$ in $O(n \log^* M)$ time and $O(n \log^* M)$ space.*

Proof. In order to compute $\hat{z}_{t+1}^{X_i}$ for a variable $X_i \to X_\ell X_r$, we need a signature sequence on which $\hat{z}_{t+1}^{X_i}$ is created, as well as its context, i.e., Δ_L signatures to the left and Δ_R to the right. To be precise, the needed signature sequence is $v_t^{X_\ell} z_t^{X_i} u_t^{X_r}$, where $u_t^{X_j}$ (resp. $v_t^{X_j}$) denotes a prefix (resp. suffix) of $y_t^{X_j}$ of length $\Delta_L + \Delta_R + 4$ for any variable X_j (see also Figure 3). Also, we need $A_t u_t^{X_n}$ and $v_t^{X_n} B_t$ to create $\hat{A}_{t+1}^{X_n}$ and $\hat{B}_{t+1}^{X_n}$, respectively.

Note that by Definition 8, $|z_t^X| > \Delta_L + \Delta_R + 9$ if $z_t^X \neq \varepsilon$. Then, we can compute $u_t^{X_i}$ for all variables X_i in $O(n \log^* M)$ time and space by processing variables in increasing order on the basis of the following fact: $u_t^{X_i} = u_t^{X_\ell}$ if $z_t^{X_\ell} \neq \varepsilon$, otherwise $u_t^{X_i}$ is the prefix of $z_t^{X_i}$ of length $\Delta_L + \Delta_R + 4$. Similarly $v_t^{X_i}$ for all variables X_i can be computed in $O(n \log^* M)$ time and space.

Using $u_t^{X_i}$ and $v_t^{X_i}$ for all variables X_i, we can obtain $O(n \log^* M)$ blocks of signatures to which we give signatures. We determine signatures by sorting the blocks by bucket sort as in Lemma 12 in $O(n \log^* M)$ time. Hence, we can get $\hat{\Lambda}_{t+1}$ in $O(n \log^* M)$ time and space. ◀

Proof of Theorem 3 (3b). Using Lemmas 14, 15 and 16, we can get $\hat{\Lambda}_{h^{X_n}}$ in $O(n \log\log (n \log^* M) \log N \log^* M)$ time by computing $\Lambda_0, \hat{\Lambda}_1, \dots, \hat{\Lambda}_{h^{X_n}}$ incrementally. Note that during the computation we only have to keep Λ_t (or $\hat{\Lambda}_t$) for the current t and the assignments of $\mathcal{G}$. Hence the working space is $O(n \log^* M + w)$. By processing $\hat{\Lambda}_{h^{X_n}}$ in $O(n \log^* M)$ time, we can get the DAG of $\mathcal{G}$ of size $O(w)$. ◀

7 Applications

Theorem 17 is an application to text compression. Theorems 19-23 are applications to compressed string processing, where the task is to process a given compressed representation of string(s) without explicit decompression. We believe that only a few applications are listed here, considering the importance of LCE queries. As one example of unlisted applications, there is a paper [14] in which our LCE data structure was used to improve an algorithm of computing the Lyndon factorization of a string represented by a given SLP.

▶ **Theorem 17.** (1) *Given a dynamic signature encoding $\mathcal{G}$ for $\mathcal{G} = (\Sigma, \mathcal{V}, \mathcal{D}, S)$ of size w which generates T, we can compute an SLP $\mathcal{S}$ of size $O(w \log |T|)$ generating T in $O(w \log |T|)$ time.* (2) *Let us conduct a single INSERT or DELETE operation on the string T generated by the SLP of (1). Let y be the length of the substring to be inserted or deleted, and let T' be the resulting string. During the above operation on the string, we can update, in $O((y + \log |T'| \log^* M)(f_{\mathcal{A}} + \log |T'|))$ time, the SLP of (1) to an SLP $\mathcal{S}'$ of size $O(w' \log |T'|)$ which generates T', where w' is the size of updated $\mathcal{G}$ which generates T'.*

We can get the next lemma using Theorem 3 (3b) and Theorem 2:

▶ **Lemma 18.** *Given an SLP of size n representing a string of length N, we can sort the variables of the SLP in lexicographical order in $O(n \log n \log N \log^* N)$ time and $O(n \log^* N + w)$ working space.*

Lemma 18 has an application to an SLP-based index of Claude and Navarro [8]. In the paper, they showed how to construct their index in $O(n \log n)$ time if the lexicographic order of variables of a given SLP is already computed. However, in order to sort variables they almost decompressed the string, and hence, needs $\Omega(N)$ time and $\Omega(N \log |\Sigma|)$ bits of working space. Now, Lemma 18 improves the sorting part yielding the next theorem.

▶ **Theorem 19.** *Given an SLP of size n representing a string of length N, we can construct the SLP-based index of [8] in $O(n \log n \log N \log^* N)$ time and $O(n \log^* N + w)$ working space.*

▶ **Theorem 20.** *Given an SLP $\mathcal{S}$ of size n generating a string T of length N, we can construct, in $O(n \log\log n \log N \log^* N)$ time, a data structure which occupies $O(n \log N \log^* N)$ space and supports $\mathsf{LCP}(val(X_i), val(X_j))$ and $\mathsf{LCS}(val(X_i), val(X_j))$ queries for variables X_i, X_j in $O(\log N)$ time. The $\mathsf{LCP}(val(X_i), val(X_j))$ and $\mathsf{LCS}(val(X_i), val(X_j))$ query times can be improved to $O(1)$ using $O(n \log n \log N \log^* N)$ preprocessing time.*

▶ **Theorem 21.** *Given an SLP $\mathcal{S}$ of size n generating a string T of length N, there is a data structure which occupies $O(w + n)$ space and supports queries $\mathsf{LCE}(val(X_i), val(X_j), a, b)$ for variables X_i, X_j, $1 \leq a \leq |X_i|$ and $1 \leq b \leq |X_j|$ in $O(\log N + \log \ell \log^* N)$ time, where $w = O(z \log N \log^* N)$. The data structure can be constructed in $O(n \log\log n \log N \log^* N)$ preprocessing time and $O(n \log^* N + w)$ working space, where $z \leq n$ is the size of the LZ77 factorization of T and ℓ is the answer of LCE query.*

Let h be the height of the derivation tree of a given SLP $\mathcal{S}$. Note that $h \geq \log N$. Matsubara et al. [18] showed an $O(nh(n + h \log N))$-time $O(n(n + \log N))$-space algorithm to compute an $O(n \log N)$-size representation of all palindromes in the string. Their algorithm uses a data structure which supports in $O(h^2)$ time, LCE queries of a special form $\mathsf{LCE}(val(X_i), val(X_j), 1, p_j)$ [20]. This data structure takes $O(n^2)$ space and can be constructed in $O(n^2 h)$ time [16]. Using Theorem 21, we obtain a faster algorithm, as follows:

▶ **Theorem 22.** *Given an SLP of size n generating a string of length N, we can compute an $O(n \log N)$-size representation of all palindromes in the string in $O(n \log^2 N \log^* N)$ time and $O(n \log^* N + w)$ space.*

Our data structures also solve *the grammar compressed dictionary matching problem* [15].

▶ **Theorem 23.** *Given a DSLP $\langle \mathcal{S}, m\rangle$ of size n that represents a dictionary $\Pi_{\langle \mathcal{S}, m\rangle}$ for m patterns of total length N, we can preprocess the DSLP in $O((n \log\log n + m \log m) \log N \log^* N)$ time and $O(n \log N \log^* N)$ space so that, given any text T in a streaming fashion, we can detect all occ occurrences of the patterns in T in $O(|T| \log m \log N \log^* N + occ)$ time.*

It was shown in [15] that we can construct in $O(n^4 \log n)$ time a data structure of size $O(n^2 \log N)$ which finds all occurrences of the patterns in T in $O(|T|(h+m))$ time, where h is the height of the derivation tree of DSLP $\langle \mathcal{S}, m\rangle$. Note that our data structure of Theorem 23 is always smaller, and runs faster when $h = \omega(\log m \log N \log^* N)$.

References

1 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Dynamic pattern matching. Technical report, Department of Computer Science, University of Copenhagen, 1998.

2 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *Proc. SODA 2000*, pages 819–828, 2000.

3 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.

4 M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *J. Algorithms*, 57(2):75–94, 2005.

5 P. Bille, P. H. Cording, I. L. Gørtz, B. Sach, H. W. Vildhøj, and Søren Vind. Fingerprints in compressed strings. In *Proc. WADS 2013*, pages 146–157, 2013.

6 Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, and Inge Li Gørtz. Finger search, random access, and longest common extensions in grammar-compressed strings. *CoRR*, abs/1507.02853, 2015. URL: http://arxiv.org/abs/1507.02853.

7 Philip Bille, Inge Li Gørtz, Mathias Bæk Tejs Knudsen, Moshe Lewenstein, and Hjalte Wedel Vildhøj. Longest common extensions in sublinear space. In Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, editors, *Combinatorial Pattern Matching – 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 – July 1, 2015, Proceedings*, volume 9133 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2015. doi:10.1007/978-3-319-19929-0_6.

8 Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fundamenta Informaticae*, 111(3):313–337, 2011.

9 Johannes Fischer, Tomohiro I, and Dominik Köppl. Deterministic sparse suffix sorting on rewritable texts. In *LATIN 2016: Theoretical Informatics – 12th Latin American Symposium, Ensenada, Mexico, April 11-15, 2016, Proceedings*, pages 483–496, 2016.

10 Pawel Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal dynamic strings. *CoRR*, abs/1511.02612, 2015. URL: http://arxiv.org/abs/1511.02612.

11 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

12 Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *Proc. STOC 2002*, pages 602–608, 2002.

13 Tomohiro I, Wataru Matsubara, Kouji Shimohira, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, Kazuyuki Narisawa, and Ayumi Shinohara. Detecting regularities on grammar-compressed strings. *Inf. Comput.*, 240:74–89, 2015.

14 Tomohiro I, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Lyndon factorization algorithms for SLP and LZ78 compressed text. *Theoretical Computer Science*, 2016. in press. doi:10.1016/j.tcs.2016.03.005.

15 Tomohiro I, Takaaki Nishimoto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Compressed automata for dictionary matching. *Theor. Comput. Sci.*, 578:30–41, 2015. doi:10.1016/j.tcs.2015.01.019.

16 Yury Lifshits. Processing compressed texts: A tractability border. In *Proc. CPM 2007*, volume 4580 of *LNCS*, pages 228–240, 2007.

17 S. Maruyama, M. Nakahara, N. Kishiue, and H. Sakamoto. ESP-index: A compressed index based on edit-sensitive parsing. *J. Discrete Algorithms*, 18:100–112, 2013.

18 W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theor. Comput. Sci.*, 410(8–10):900–913, 2009.

19 Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997.

20 M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. CPM 1997*, pages 1–11, 1997.

21 Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. *CoRR*, abs/1605.01488, 2016. URL: http://arxiv.org/abs/1605.01488.

22 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1–3):211–222, 2003.

23 S. C Sahinalp and Uzi Vishkin. Data compression using locally consistent parsing. *TechnicM report, University of Maryland Department of Computer Science*, 1995.

24 Yuka Tanimura, Tomohiro I, Hideo Bannai, Shunsuke Inenaga, Simon J. Puglisi, and Masayuki Takeda. Deterministic sub-linear space LCE data structures with efficient construction. In *Proc. CPM 2016*, 2016. to appear.

25 J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–349, 1977.

Undecidability of Two-Dimensional Robot Games

Reino Niskanen[1], Igor Potapov[2], and Julien Reichert[3]

1 Department of Computer Science, University of Liverpool, UK
r.niskanen@liverpool.ac.uk
2 Department of Computer Science, University of Liverpool, UK
potapov@liverpool.ac.uk
3 LSV, ENS Cachan, France
reichert@crans.org

Abstract

Robot game is a two-player vector addition game played on the integer lattice $\mathbb{Z}^n$. Both players have sets of vectors and in each turn the vector chosen by a player is added to the current configuration vector of the game. One of the players, called Eve, tries to play the game from the initial configuration to the origin while the other player, Adam, tries to avoid the origin. The problem is to decide whether or not Eve has a winning strategy. In this paper we prove undecidability of the robot game in dimension two answering the question formulated by Doyen and Rabinovich in 2011 and closing the gap between undecidable and decidable cases.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation

Keywords and phrases reachability games, vector addition game, decidability, winning strategy

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.73

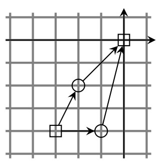
1 Introduction

In the modern world the reliability of a software code and verification of the correct functionality of complex technological devices require the analysis of various interactive processes and open systems, where it is important to take into account the effects of uncontrollable adversaries, such as environment or malicious users. Computational games provide a good framework to model interactive processes and the extensions of classical reachability problems to game schemes, studied in different contexts and settings, have recently garnered considerable interest [2, 3, 5, 6, 8, 10, 19].

In this paper we study two-player games where the main problem is to decide which of the players wins based on a given set of eligible moves, a computational environment and reachability objectives. Following early results for games on VASS (Vector Addition Systems with States)[1] [1, 6], Doyen and Rabinovich formulated an open problem about the simplest version of games (*robot games*) for which the decidability was unknown [9]. Robot games are two-player games played by updating a vector of n integer counters. Each of the players, called Adam and Eve, has a finite set of vectors in $\mathbb{Z}^n$. A play starts from a given initial vector $\mathbf{x}_0 \in \mathbb{Z}^n$, and proceeds in rounds. During each round, first Adam adds a vector from his set, followed by Eve doing the same. Eve wins when, after her turn, the vector is the zero vector. A simple example of the game is illustrated in Figure 1.

We say that Eve has a winning strategy if she eventually can reach the zero vector regardless of the moves Adam plays. As a consequence of [15], robot games are determined,

[1] A game is played on a graph with states of player 1 and states of player 2, with $\mathbb{N}^2$ as the vector space.



Adam's moves: $\{(1,2),(2,0)\}$

Eve's moves: $\{(2,2),(1,4)\}$

Figure 1 An example of a robot game.

that is, Adam has a winning strategy if Eve does not. Thus a winning strategy gives a way for a player to win, regardless of the way the opponent plays. Previously, it has been proved that deciding the winner in one-dimensional robot games, where integers are given in binary, is EXPTIME-complete [4].

In this paper we consider the open problem of deciding the winner of robot games for dimension $n = 2$ and show that it is undecidable to check which of the players has a winning strategy in a two-dimensional robot game, i.e., in a very restricted fragment of counter reachability games with stateless players playing in integer grid $\mathbb{Z}^2$. The basis of proofs are 2-counter Minsky machines (2CM) for which the halting problem is undecidable. For a 2-counter machine, we construct a game where Eve has to simulate the machine and Adam verifies that Eve does not cheat. The intuition is that the counters of the machine are multiplied by constants and represented by two-dimensional vectors. Additionally, the states of the machine are encoded in the least significant digits of the vectors. We analyse all the possible deviations from simulating the counter machine and show that the opponent has a winning strategy in that case. The biggest challenge is to ensure that all possible ways to cheat can be caught without introducing new ways to cheat for the other player.

We prove the main theorem by considering the undecidable problem of determining whether a 2CM $\mathcal{M}$ reaches a configuration where both counters are zero. In Section 3, we construct a robot game with states that follows the computation of $\mathcal{M}$. To simulate zero checks present in 2CM, Adam has a move allowing him to check whether a counter is positive or not leading, deterministically, either to his victory with a correct guess or to his loss otherwise. In the fourth section, we map the states and state transitions into integers and embed them into the least significant digits in vectors of a two-dimensional robot game. Our proof uses two successive reductions making the proof shorter and more intuitive in contrast to a direct reduction from 2CM that would lead to a longer proof with significantly more cases to consider. All proofs omitted due to length constrain can be found in [17].

Apart from the solution of the open problem, the main contribution of this paper is a collection of new, original encodings and constructions that allow simulating zero-checks and state space of a universal machine within a minimalistic two-dimensional system of two non-deterministic stateless players.

Previous research: Robot games are subfamily of counter reachability games where the game is played on a graph with vertices partitioned between players. It has been proved that deciding the winner in two-dimensional counter reachability games is undecidable [19]. Our result can be seen as strengthening of this as our arena is a graph without self-loops and with one vertex for each player, i.e., both players are stateless.

In [2] and [6], VASS games, where the game is played on a graph and counters are always positive, were considered. It was proven that already in two dimensions it is undecidable who wins if Eve's goal is to reach a particular vertex with counter $(0,0)$. On the other hand, if it can be any vertex, then the problem is $(k-1)$-EXPTIME for a game with k counters. Later, the result was improved to PTIME for $k = 2$ [7]. In [19], the possible counter values were extended to all integers and it was proven that the problem remains undecidable. Hunter considered the variants of games, where updates on the counters are done in binary, and showed that one-dimensional games are EXPSPACE-complete [13]. While these games have reachability objectives, it is also possible to extend the objectives of the games to energy constrains [10] or parity constrains [3, 8].

The proofs of undecidability of VASS games and counter reachability in two dimensions in [1, 6] use the state structure of the game to embed the state structure of a 2-counter machine. In this sense, our result on robot games with states is comparable, as Eve simulates the state transitions of a 2-counter machine with her underlying automaton. On the other hand, the stateless game is essentially different as we have to represent state transitions with integers. When simulating a two-counter machine, it is possible for Eve to make a wrong move and then Adam is able to ensure his victory from this point onward. In robot games, Eve's state is dependent only on her previous moves, while in VASS games or counter reachability games, Adam's moves effect which state Eve enters. Because of this, Adam's cheat catching ability is implemented in a different way.

2 Notation and definitions

We denote the set of all integers by $\mathbb{Z}$ and the set of all non-negative integers by $\mathbb{N}$. By 0_n we denote a n-dimensional zero vector.

A *counter reachability game* (CRG) consists of a directed graph $G = (V, F)$, where the set of vertices is partitioned into two parts, V_1 and V_2, each edge $e \in F \subseteq V \times \mathbb{Z}^n \times V$ is labelled with vectors in $\mathbb{Z}^n$. A *configuration* of the game is $(v, \mathbf{x})$, a successive configuration is $(v', \mathbf{x} + \mathbf{x}')$, where an edge $(v, \mathbf{x}', v') \in E$ is chosen by player 1 if $v \in V_1$ or by player 2 if $v \in V_2$. A *play* is a sequence of successive configurations. The goal of the first player, called *Eve*, is to reach the *final configuration* $(v_f, 0_n)$ for some $v_f \in V$ from a given initial configuration $(v_0, \mathbf{x}_0)$, while the goal of the second player, called *Adam*, is to keep Eve from reaching $(v_f, 0_n)$. A *strategy* for a player is a function that maps a configuration to an edge that can be applied. We say that Eve has a *winning strategy* if she can reach the final configuration regardless of the strategies of Adam. The determinacy result of [15] holds for counter reachability games and thus Adam has a winning strategy if Eve does not have a winning strategy. In the figures we use ◯ for Eve's states and □ for Adam's states.

A *robot game* (RG) [9] is a special case of the counter reachability games, where the graph consists of only two vertices, v_0 of Adam and v of Eve. The goal of the game is the configuration $(v_0, 0_n)$. That is, a robot game consists of two players, *Eve* and *Adam*, having a set of vectors E, A over $\mathbb{Z}^n$, respectively, and an *initial vector* $\mathbf{x}_0$. Starting from $\mathbf{x}_0$ players add a vector from their respective sets to the current configuration of the game in turns. As in counter reachability games, Eve tries to reach the origin while Adam tries to keep Eve from reaching the origin. The decision problem concerning robot games is, for a given robot game (A, E) and $\mathbf{x}_0$, to decide whether Eve has a winning strategy to reach 0_n from $\mathbf{x}_0$. The problem is EXPTIME-complete in dimension one [4] and was open for dimension two.

An extension of robot games where players have control states is called *robot games with states* (RGS). We consider only the games where Adam's state structure is trivial, i.e., he has only one state and all moves are self-loops. RGS consists of (A, E) where A is a finite subset of $\mathbb{Z}^n$ that Adam can apply during his turn and E is a finite subset of $V \times \mathbb{Z}^n \times V$ of Eve. The configuration is now a pair $(s, \mathbf{v})$ consisting of Eve's control state s and a counter vector $\mathbf{v} \in \mathbb{Z}^n$. Eve updates her control state when she makes a move: in the configuration $(s, \mathbf{v})$, for any vector $\mathbf{v}$, only moves of the form $(s, \mathbf{x}, t)$ are enabled, and with one such move the new configuration is $(t, \mathbf{v} + \mathbf{x})$. Eve wins if, and only if, after her turn, the configuration is $(s, (0, 0))$ for any $s \in V$. The decision problem associated with robot games with states asks whether Eve has a winning strategy from a given configuration.

A *Minsky machine*, introduced in [16], is a simple computation model that is crucial in our proof. A *deterministic two-counter Minsky machine* (2CM) is a pair (Q, T), where Q is a finite set of states and $T \subseteq Q \times \{c_{i++}, c_{i--}, c_i{=}{=}0 \mid i = 1, 2\} \times Q$ is a finite set of labelled transitions to increment, decrement or test for zero one of the counters. In a deterministic two-counter Minsky machine, the set Q contains an initial state s_0 and a sink state $\perp$, such that there is no outgoing transition from $\perp$. Moreover, from all $s \in Q \setminus \{\perp\}$, either there is only one outgoing transition with the label c_{1++} or c_{2++}, or there are exactly two outgoing transitions with respective labels c_{1--} and $c_1{=}{=}0$, or c_{2--} and $c_2{=}{=}0$. A configuration of a 2CM is a pair $(s, (y, z)) \in Q \times \mathbb{N}^2$, representing a state and a pair of counter values. The run of a 2CM is a finite or infinite sequence of configurations that starts from $(s_0, (0, 0))$ and follows the transitions of the machine incrementing and decrementing the counters according to the labels. As usual, a transition with a label $c_i{=}{=}0$ can only be taken when the counter i is zero and a transition with a label c_{i--} can only be taken when the counter i is positive.

Note that there is only one possible run in a deterministic two-counter Minsky machine. Indeed, when there are two outgoing transitions, only one of them can be executed, depending on the value of the counter that the transitions update or test for zero. The halting problem of 2CM is to decide, given a 2CM, whether the run reaches a configuration with state $\perp$, in other words whether the run halts. This problem is known to be undecidable for deterministic two-counter machines [16]. Another well-known undecidable problem for 2CM is whether a configuration where both counters are zero is reachable. The undecidability follows from the halting problem by modifying a 2CM to ensure that both counters are zero only in the halting state; see for example [18] for a proof.

▶ **Theorem 1.** *Let (Q, T) be a deterministic two-counter machine. It is undecidable whether in the run of (Q, T), a configuration in $Q \times \{(0, 0)\} \setminus \{(s_0, (0, 0))\}$ appears.*

We can assume that the first move of a 2CM is an increment of either c_1 or c_2. Indeed, otherwise the problem is trivial as the second configuration is in $Q \times \{(0, 0)\}$.

3 Robot games with states in two dimensions

In this section we prove that the decision problem for robot games with states is undecidable. We show that for each two-counter machine, there exists a corresponding robot game with states where Eve has a winning strategy if and only if the machine reaches a configuration where both counters are zero. To simulate zero checks present in two-counter machines, Adam has a move allowing him to check whether a counter is positive or not.

▶ **Theorem 2.** *Let (Q, T) be a two-counter machine. There exists a two-dimensional robot game with states (A, E) where Eve has a winning strategy if and only if (Q, T) reaches a configuration in $Q \times \{(0, 0)\}$.*

The idea is that in the robot game with states, Eve simulates the computation of the 2CM while Adam does not interfere with the computation. If one of the players deviates from the computation, the opponent has a winning strategy from that point on.

Essentially, there are four ways the game can progress. These ways are depicted in the Figure 2. Three of the outcomes have a predetermined winner which does not depend on the 2CM. In the last case where Eve correctly simulates the 2CM and Adam does not interfere

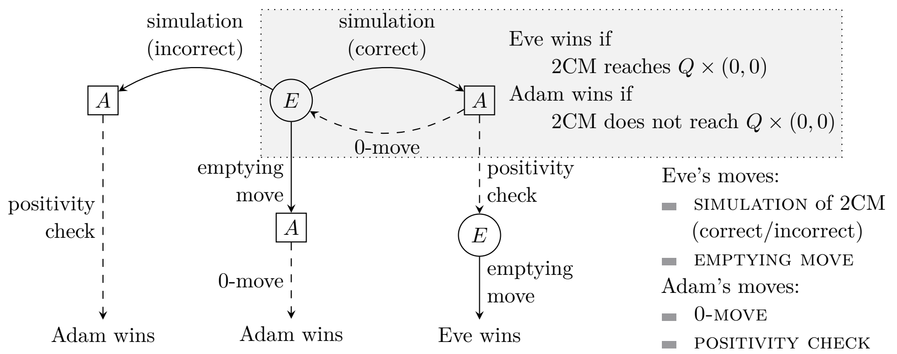


Figure 2 Progress of 2RGS.

(plays only a 0-MOVE), the winner depends on whether the 2CM reaches $(s, (0, 0))$ for some $s \in Q$ or not.

- If Eve's move corresponds to the SIMULATION of the 2CM and Adam replies with a 0-MOVE (a move that does not modify the counters), then iteratively applying only this turn-based interaction, Eve wins if and only if the 2CM reaches $(s, (0, 0))$ for some $s \in Q$ (Lemma 3).
- If Eve's move incorrectly simulates the 2CM, then Adam has a winning strategy from this moment on, starting with a POSITIVITY CHECK that makes Eve's target unreachable (Lemma 4).
- On the other hand, if Adam plays his POSITIVITY CHECK following a correct simulating move of Eve, then Eve has a winning strategy from this moment on, starting with an EMPTYING MOVE allowing Eve to empty both counters and reach $(0, 0)$ (Lemma 5).
- Finally, if Eve plays an EMPTYING MOVE instead of a SIMULATING MOVE, in that case Adam has a winning strategy starting by playing his 0-MOVE (Lemma 6).

Before presenting the detailed constructions of Eve's and Adam's state spaces, we consider a simple modification to a 2CM, making it non-deterministic. For any 2CM (Q, T), we construct a 2CM (Q', T') where Q' is Q with additional information on positivity of the both counters and T' is like T with guards ensuring that the extra information in states of Q' correspond to the actual values of the counters. We denote the states of Q' by s_{ab} where $s \in Q$ and $a, b \in \{0, +\}$ are flags indicating whether the value of a counter is positive or equal to 0, i.e., a (b) is $+$ if the first (second) counter is positive or 0 if the counter is zero. The transition set T' consists of the following sets

$$\begin{aligned}
&\{(s_{ab}, c_1{+}{+}, t_{+b}) \mid (s, c_1{+}{+}, t) \in T, a, b \in \{0, +\}\}, \{(s_{ab}, c_2{+}{+}, t_{a+}) \mid (s, c_2{+}{+}, t) \in T, a, b \in \{0, +\}\},\\
&\{(s_{+b}, c_1{-}{-}, t_{ab}) \mid (s, c_1{-}{-}, t) \in T, a, b \in \{0, +\}\}, \{(s_{a+}, c_2{-}{-}, t_{ab}) \mid (s, c_2{-}{-}, t) \in T, a, b \in \{0, +\}\},\\
&\{(s_{0b}, c_1{=}{=}0, t_{0b}) \mid (s, c_1{=}{=}0, t) \in T, b \in \{0, +\}\}, \{(s_{a0}, c_2{=}{=}0, t_{a0}) \mid (s, c_2{=}{=}0, t) \in T, a \in \{0, +\}\}.
\end{aligned}$$

Now, after decrementing counters from a state with $+$ flag, a state will be changed to a state with $+$ or 0 flag depending on the current counter value.

counter value	flag		flag	
$c_i > 1$	$+$	$\rightarrow$	$+$	**correct flag**
$c_i > 1$	$+$	$\rightarrow$	0	wrong flag
$c_i = 1$	$+$	$\rightarrow$	$+$	wrong flag
$c_i = 1$	$+$	$\rightarrow$	0	**correct flag**

At the moment we assume that the machine moves to a state with the correct flag (correct simulation) and does not move to incorrect flag (incorrect simulation). Later in the robot

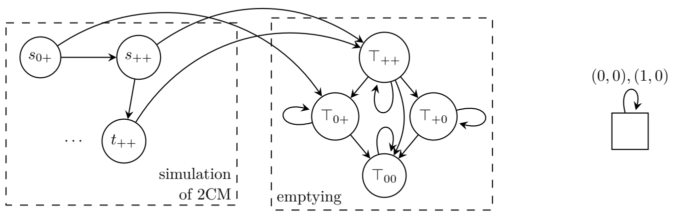


Figure 3 An illustration of state transitions of Eve and Adam.

game with states, Adam will act as guards (i.e., checks whether $c_i > 1$ or $c_i = 1$) using his POSITIVITY CHECK if Eve picks a wrong transition resulting in a state with the wrong flag.

Now we present the moves of the players. Eve's states are the states of Q', corresponding to the simulation of the 2CM, together with emptying states $\{\top_{00}, \top_{+0}, \top_{0+}, \top_{++}\}$, associated with EMPTYING MOVES. The moves of Eve correspond to transitions in T' where incrementing and decrementing of the first counter is by 4 rather than by 1. We call these moves SIMULATING MOVES.

Transition with c_1	Eve's move
$(s, c_1{+}{+}, t)$	$(s, (4,0), t)$
$(s, c_1{-}{-}, t)$	$(s, (-4,0), t)$
$(s, c_1{=}{=}0, t)$	$(s, (0,0), t)$

Transition with c_2	Eve's move
$(s, c_2{+}{+}, t)$	$(s, (0,1), t)$
$(s, c_2{-}{-}, t)$	$(s, (0,-1), t)$
$(s, c_2{=}{=}0, t)$	$(s, (0,0), t)$

The other type of moves, EMPTYING MOVES, are related to the new states and are used to empty the counters. Note that there is hierarchy in the emptying states — Eve cannot move from a state with 0 to a state with +. Let us define the emptying partition of Eve's automaton where for every possible move of Adam there is a cancelling move with additional decrementing of the counters eventually leading to the sink state $\top_{00}$.

- $\{(\top_{++}, (-4-e, -1), t) \mid e \in \{0,1\}, t \in \{\top_{++}, \top_{+0}, \top_{0+}, \top_{00}\}\}$;
- $\{(\top_{+0}, (-4-e, 0), t) \mid e \in \{0,1\}, t \in \{\top_{+0}, \top_{00}\}\}$;
- $\{(\top_{0+}, (-e, -1), t) \mid e \in \{0,1\}, t \in \{\top_{0+}, \top_{00}\}\}$;
- $\{(\top_{00}, (-e, 0), \top_{00}) \mid e \in \{0,1\}\}$.

Finally, we define transitions connecting the simulating partition of Eve's automaton with the emptying partition. For each state $s_{ab} \in Q'$, Eve has a transition $(s_{ab}, (-1,0), \top_{ab})$.

Adam is stateless, i.e., he has one state and his moves are self-loops. There are two types of moves: the 0-MOVE, $(0,0)$, with which Adam agrees that Eve simulated the 2CM correctly and the POSITIVITY CHECK, $(1,0)$, with which Adam checks whether a flag matches the counter (i.e., Eve simulated incorrectly). Control states of the players are depicted in Figure 3.

To avoid Eve winning trivially every play in the robot game with states, we do not use $(s'_{00}, (0,0))$ as an initial configuration, but instead consider the configuration that is reached in (Q', T') after one step of the run of the machine. We write the configuration after one step as $(s_{\overline{a}\overline{b}}, (y,z))$ and we define $\overline{a} = +, \overline{b} = 0$ if $y = 1$ and $\overline{a} = 0, \overline{b} = +$ if $y = 0$. The initial configuration in the robot game with states is then $(s_{\overline{a}\overline{b}}, (4y, z))$. The effect of simulating moves, emptying moves and positivity check modulo four is depicted in Figure 4.

Next we prove which player has a winning strategy in the scenarios presented previously.

▶ **Lemma 3.** *In a sequence where Adam plays only the* 0-MOVE *and Eve plays only correct* SIMULATING MOVES*, Adam wins if the 2-counter machine does not reach a configuration with zeros in both counters and Eve wins otherwise.*

Proof. It easy to see that correct moves of Eve simulate the 2CM and that a configuration $(s, (0,0))$ of the 2CM is reachable if and only if it is reachable in 2RGS. ◀

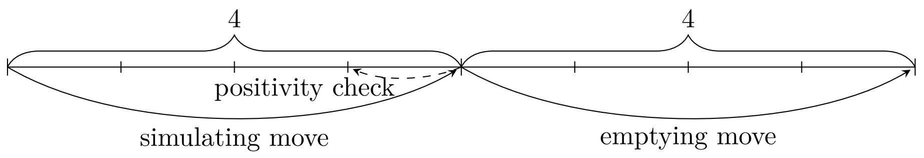


Figure 4 An illustration of changes in an interval when simulating or emptying moves of Eve or positivity check of Adam is applied.

▶ **Lemma 4.** *If Eve plays an incorrect move, i.e., after her turn a flag does not match the counter value (i.e., the flag is + while the counter is 0 or vice versa), then Adam has a winning strategy starting with the* POSITIVITY CHECK.

Proof. Assume that Eve made a mistake regarding the positivity of the first counter. As noted previously, there are two ways she can make a mistake. Either the configuration is $(s_{0b}, (4x, y))$, where $x \geq 1$ or $(s_{+b}, (0, y))$. In both cases Adam plays his POSITIVITY CHECK which changes the parity of the first counter. That is, after Adam's turn, the first counter is $1 \pmod 4$. It is easy to see that if Eve does not change the parity of the counter back to zero with her following turn, then Adam has a winning strategy. Indeed in this case, he will play his POSITIVITY CHECK if and only if the first counter is not $3 \pmod 4$. Eve cannot make the counter 0, as she cannot even make it $0 \pmod 4$. Thus Eve has to play a move adding -1 to the first counter. The only move for that is $(s_{ab}, (-1, 0), \top_{ab})$ which takes Eve to an emptying state. In the first case the emptying state is $\top_{0b}$ and all the transitions from it do not modify the first counter, i.e., Eve cannot reach $(0, 0)$. In the second case the emptying state is $\top_{+b}$ where the next transition subtracts 4 from the first counter making it negative and there are no moves that increment the counters. Again, Eve cannot reach $(0, 0)$. The case where Eve makes a mistake with the second counter is proven analogously and, in fact, Adam's strategy is the same. ◀

▶ **Lemma 5.** *Assume that Eve plays only correct* SIMULATING MOVES *before Adam plays the* POSITIVITY CHECK *for the first time. If Adam plays the* POSITIVITY CHECK, *then Eve has a winning strategy starting with an* EMPTYING MOVE.

Proof. Similarly as in the previous proof, if Eve does not play an EMPTYING MOVE, then Adam has a winning strategy. Now, the configuration is $(s_{ab}, (4x + 1, y))$ after Adam's turn and Eve plays $(s_{ab}, (-1, 0), \top_{ab})$. From that point onward, Eve can empty the counters ensuring that the first counter is $0 \pmod 4$ and that the flags match the positivity of the counters. That is, every time Adam plays his POSITIVITY CHECK, Eve plays an EMPTYING MOVE subtracting one from the first counter. Eventually, Eve will reach the configuration $(\top_{00}, (0, 0))$ and win the game. ◀

▶ **Lemma 6.** *If Adam plays only the* 0-MOVE *and Eve plays an* EMPTYING MOVE. *Adam has a winning strategy starting with the* 0-MOVE.

Proof. After Eve's move, the first counter is $3 \pmod 4$. As in proof of Lemma 4, Adam ensures that the first counter stays non-zero modulo four and wins the game. ◀

Proof of Theorem 2. Let (A, E) be the robot game with states constructed in this section. Assume first that (Q, T) reaches a configuration in $Q \times \{(0, 0)\}$. Now by Lemma 3, Eve's winning strategy is to respond with the correct SIMULATING MOVES if Adam plays the 0-MOVE, and if Adam plays a POSITIVITY CHECK, then Eve has a sequence of moves described in Lemma 5 that leads to the configuration $(\top_{00}, (0, 0))$.

Assume then that (Q, T) never reaches a configuration in $Q \times \{(0,0)\}$. We show that Eve does not have a winning strategy. If Adam plays only the 0-MOVE, then, by Lemma 3, Eve does not win by responding with just the correct SIMULATING MOVES. Alternatively, if at some point, she plays either an incorrect SIMULATING MOVE or an EMPTYING MOVE, then by Lemmas 4 and 6, respectively, Adam has winning strategies making sure that a configuration with counter values $(0,0)$ is not reachable. As we analysed all the possible moves of Eve, we have shown that Eve does not have a winning strategy. ◀

By Theorems 1 and 2, we have the following corollary regarding decidability of 2-dimensional robot games with states.

▶ **Corollary 7.** *Let (A, E) be a robot game with states and $\mathbf{x}_0$ be the initial vector. It is undecidable whether Eve has a winning strategy to reach $(0,0)$ from $\mathbf{x}_0$. In particular, Adam is stateless and does not modify the second counter.*

4 Stateless robot games in two dimensions

In this section we prove the main result that it is undecidable whether Eve has a winning strategy in a two-dimensional robot game. We prove the claim by constructing a robot game that simulates a robot game with states. In some ways the construction is similar to the construction of a game with states in the previous section as can be seen in similarities of figures 2 and 5. On the other hand, the construction of the stateless game is more complex as the information on two counters, states and state transitions has to be embedded into two-dimensional vectors.

▶ **Theorem 8.** *Let (A_1, E_1) be a 2-dimensional robot game with states where Adam is stateless and does not modify the second counter. There exists a two-dimensional robot game (A, E) where Eve has a winning strategy if and only if Eve has a winning strategy in (A_1, E_1).*

Similarly to the construction of Section 3, the idea is that in the robot game, Eve and Adam simulate a play of the 2RGS. If one of the players deviates from the play, the opponent has a winning strategy from that point onward. In Figure 5, we present a schematic similar to Figure 2 depicting the possible ways two-dimensional robot games can go. Three of the outcomes have a predetermined winner which does not depend on the 2RGS. In the last case where Eve and Adam correctly simulate the 2RGS, the winner depends on the winner of the 2RGS, i.e., whether Eve has a winning strategy to reach $(s, (0,0))$, for any state s, or not.

- If Eve's move corresponds to a move in a play of the 2RGS, that we call a REGULAR MOVE, and Adam replies with his REGULAR MOVE, then iteratively applying only this turn-based interaction, Eve has a winning strategy if and only if she has a winning strategy in the corresponding 2RGS (Lemma 10).
- If Eve's move incorrectly simulates the 2RGS, then Adam has a winning strategy from this moment on starting with a STATE-CHECK that makes Eve's target unreachable (Lemma 11).
- On the other hand, if Adam plays his STATE-CHECK following a correct REGULAR MOVE of Eve, then Eve has a winning strategy from this moment on starting with a STATE-DEFENCE MOVE allowing Eve to empty both counters and reach $(0,0)$ (Lemma 12).
- Finally, if Eve plays a STATE-DEFENCE MOVE instead of a REGULAR MOVE, in that case Adam has a winning strategy starting by playing his REGULAR MOVE (Lemma 13).

Intuitively, we encode the states as powers of 8 such that the coefficient of 8^i is 1 if and only if Eve's state in robot games with states is s_i. When the state changes from

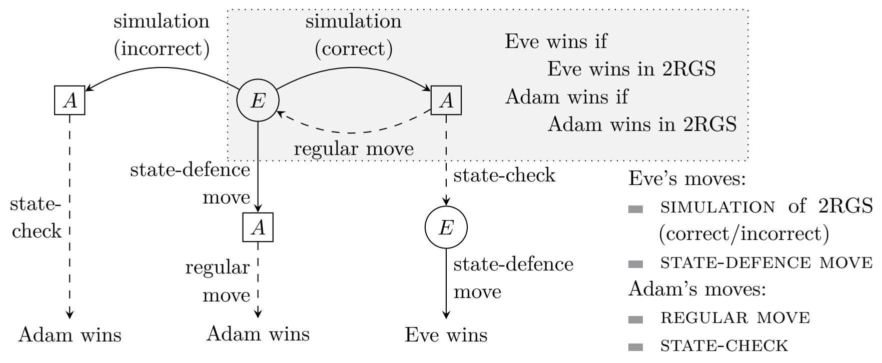


Figure 5 Progress of 2RG.

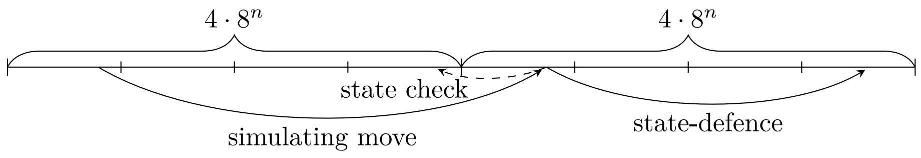


Figure 6 An illustration of changes in interval when simulating or state-defence moves of Eve or state check of Adam is applied.

s_i to s_j, $-8^i + 8^j$ is added to the second counter. Let $(s_i, (x, y))$ be a configuration in a two-dimensional robot game with m states. Let us represent the state s_i with m-dimensional characteristic vector $\mathbf{s_i} = (s_1, \dots, s_m)$ where s_i is 1 and $s_j = 0$ for all $j \neq i$. We can now map $\mathbf{s_i}$ to an integer defined by the sum $\sum_{k=1}^{m} s_k 8^k$. A transition from s_i to s_j can be simulated by adding $\sum_{k=1}^{m} s_k 8^k$, where $s_k = -1$ if $k = i$, $s_k = 1$ if $k = j$, and zero otherwise. We represent states as coefficients of powers of eight because we need the extra space smaller bases do not possess.

It is easy to see that this is not enough as incorrect transitions can result in a correct configuration. For example, if the configuration of the 2RGS is $(s_i, (x, y))$ and moves corresponding to $(s_j, (a, b), s_k)$ and $(s_k, (c, d), s_j)$ are used, the resulting configuration corresponds to $(s_i, (x + a + c, y + b + d))$. Another way to cheat is to use carries as incrementing the coefficient of 8^i eight times is indistinguishable from incrementing the coefficient of 8^{i+1} once. Both types of cheating can be countered with Adam's STATE-CHECKS.

We now show how we embed the states and state transitions into the second counter of the game. Similarly to how in the previous section we created additional space in the first counter by multiplying the moves modifying the first counter by four, we multiply the second counter by $4 \cdot 8^n$, where $n = m + 7$ and m is the number of states, creating enough space to store all the needed information of the underlying automaton. The multiplication by $4 \cdot 8^n$ rather than just 8^n has two purposes. The first one is similar to multiplying the first counter by four in the Section 3. Namely, certain moves will move between different intervals modulo $4 \cdot 8^n$ ensuring the correct response from the opponent. This is illustrated in Figure 6. The second purpose is to ensure that above described cheating with carries is not possible. A configuration in $Q \times \mathbb{Z}^2$ is mapped to a vector in $\mathbb{Z}^2$ by $(s_i, (c_1, c_2)) \mapsto (c_1, c_2 \cdot 4 \cdot 8^n + 8^i)$.

Before presenting the detailed constructions of Eve's and Adam's moves, we note that we can assume that the 2RGS has the information on the positivity of the counters and players have to update the information correctly. Indeed, this was done in the previous section by using flags 0 and $+$. Recall that because of this, the first counter is incremented and decremented by 4. By this assumption, we can denote the states of Eve by s_{ab} as before. We also assume that Eve's automaton is without self-loops as they would allow Eve to modify the counters without modifying coefficients of the states. Let Q be the set of states of Eve in 2RGS. We create an emptying gadget for Eve similar to the one constructed in the previous reduction. To avoid self-loops, there are seven emptying states, $\{\top_{ab}, \top'_{ab} \mid a, b \in \{0, +\}\} \setminus \{\top'_{00}\}$. The state $\top'_{00}$ is not needed as $\top_{00}$ will not have any moves from it. The moves in the emptying gadget are as in the emptying gadget constructed in Section 3 but instead of self-loops, the transitions are between primed and unprimed versions of the states.

- $\{(\top_{++}, (-4, -1) - \alpha, t) \mid \alpha \in A_1, t \in \{\top'_{++}, \top_{+0}, \top'_{+0}, \top_{0+}, \top'_{0+}, \top_{00}\}\}$;
  $\{(\top'_{++}, (-4, -1) - \alpha, t) \mid \alpha \in A_1, t \in \{\top'_{++}, \top_{+0}, \top'_{+0}, \top_{0+}, \top'_{0+}, \top_{00}\}\}$;
- $\{(\top_{+0}, (-4, 0) - \alpha, t) \mid \alpha \in A_1, t \in \{\top'_{+0}, \top_{00}\}\}$;
  $\{(\top'_{+0}, (-4, 0) - \alpha, t) \mid \alpha \in A_1, t \in \{\top_{+0}, \top_{00}\}\}$;
- $\{(\top_{0+}, (0, -1) - \alpha, t) \mid \alpha \in A_1, t \in \{\top'_{+0}, \top_{00}\}\}$;
  $\{(\top'_{0+}, (0, -1) - \alpha, t) \mid \alpha \in A_1, t \in \{\top_{+0}, \top_{00}\}\}$;

We denote $\mathcal{T} = \{\top_{++}, \top'_{++}, \top_{0+}, \top'_{0+}, \top_{+0}, \top'_{+0}\}$. We think of elements of $Q \cup \mathcal{T} \cup \{\top_{00}\}$ as integers in $\{0, \ldots, n-1\}$ such that $\top_{00} = 0, \top'_{0+} = n-6, \top_{0+} = n-5, \top'_{+0} = n-4, \top_{+0} = n-3, \top'_{++} = n-2, \top_{++} = n-1$. We give names for update vectors that we often use:

$$\begin{aligned}
&\textsc{Add}(1, x) := (x, 0); && \textsc{Move}(j, k) := (0, -8^j + 8^k), \text{ for } 0 \le j, k \le n-1;\\
&\textsc{Add}(2, x) := (0, 4x \cdot 8^n); && \textsc{Check}(i) := (0, -5 \cdot 8^i - 8^n), \text{ for } n-6 \le i \le n-1.
\end{aligned}$$

The initial vector of the robot game is $\textsc{Add}(1, x) + \textsc{Add}(2, y) + \textsc{Move}(\top_{00}, s)$, that is, $(x, 4y \cdot 8^n + 8^s - 8^0)$, where $(s, (x, y))$ is the initial configuration in the robot game with states. In the next example we illustrate how the update vectors modify the counters.

▶ **Example 9.** Let (A_1, E_1) be a two-dimensional robot game with states where Eve has two states, $s = 1$ and $t = 2$, and the initial configuration $(s, (1, 0))$. Next we present a set of configurations in 2RG obtained from the corresponding initial configuration when we apply $\textsc{Add}(1, -1)$, $\textsc{Add}(2, 1)$, $\textsc{Move}(s, t)$, $\textsc{Check}(8)$ in succession:

$$\begin{aligned}
&(\overbrace{1, 0 \cdot 4 \cdot 8^9}^{\text{2RGS counters}} + \overbrace{0 \cdot 8^8 + 0 \cdot 8^7 + 0 \cdot 8^6 + 0 \cdot 8^5 + 0 \cdot 8^4 + 0 \cdot 8^3}^{\mathcal{T}} + \overbrace{0 \cdot 8^2 + 1 \cdot 8^1}^{\text{states of 2RGS}} \overbrace{-1 \cdot 8^0}^{\top_{00}}) \xrightarrow{\textsc{Add}(1,-1)}\\
&(0, 0 \cdot 4 \cdot 8^9 + 0 \cdot 8^8 + 0 \cdot 8^7 + 0 \cdot 8^6 + 0 \cdot 8^5 + 0 \cdot 8^4 + 0 \cdot 8^3 + 0 \cdot 8^2 + 1 \cdot 8^1 - 1 \cdot 8^0) \xrightarrow{\textsc{Add}(2,1)}\\
&(0, 4 \cdot 8^9 + 0 \cdot 8^8 + 0 \cdot 8^7 + 0 \cdot 8^6 + 0 \cdot 8^5 + 0 \cdot 8^4 + 0 \cdot 8^3 + 0 \cdot 8^2 + 1 \cdot 8^1 - 1 \cdot 8^0) \xrightarrow{\textsc{Move}(s,t)}\\
&(0, 4 \cdot 8^9 + 0 \cdot 8^8 + 0 \cdot 8^7 + 0 \cdot 8^6 + 0 \cdot 8^5 + 0 \cdot 8^4 + 0 \cdot 8^3 + 1 \cdot 8^2 + 0 \cdot 8^1 - 1 \cdot 8^0) \xrightarrow{\textsc{Check}(8)}\\
&(0, 3 \cdot 8^9 - 5 \cdot 8^8 + 0 \cdot 8^7 + 0 \cdot 8^6 + 0 \cdot 8^5 + 0 \cdot 8^4 + 0 \cdot 8^3 + 1 \cdot 8^2 + 0 \cdot 8^1 - 1 \cdot 8^0).
\end{aligned}$$

Now we present the moves of the players. Adam has two types of moves: REGULAR MOVES that correspond to the moves in the 2RGS and STATE-CHECK MOVES, $\{\textsc{Check}(i) \mid i \in \mathcal{T}\}$. The moves of Eve correspond to moves in E_1 where incrementing and decrementing of the second counter is by $4 \cdot 8^n$ rather than by 1. Let $(s, (x, y), t) \in E_1$, then $\textsc{Add}(1, x) + \textsc{Add}(2, y) + \textsc{Move}(s, t) = (x, 4y \cdot 8^n - 8^s + 8^t) \in E$. We call these moves REGULAR MOVES. We also need a move for Eve to finish the simulation by removing any values corresponding to the automaton if the state is s_{00}. That is, we add moves $\{\textsc{Move}(s_{00}, \top_{00}) - \alpha \mid \alpha \in A_1\}$. The other type of moves, STATE-DEFENCE MOVES, are used to empty the counters. As in the previous construction, Eve will be able to cancel every Adam's move and decrement the counters at the same time.

Finally, we define moves connecting the simulating partition of Eve's automaton with the emptying partition. For each state $s_{ab} \in Q$ where a, b are not both zero, Eve has a move $\{\textsc{Move}(s_{ab}, k) - \textsc{Check}(i) \mid (a, b) \in \{0, +\}^2 \setminus \{(0, 0)\}, k \in \{\top_{ab}, \top'_{ab}\}, k \neq i, i \in \mathcal{T}\}$. For s_{00}, Eve has a move $\{\textsc{Move}(s_{00}, \top_{00}) - \textsc{Check}(i) \mid i \in \mathcal{T}\}$.

Adam's move	Eve's move
$\alpha \in A_1$	$\{\textsc{Add}(1, -4) + \textsc{Add}(2, -1) - \textsc{Move}(j, k) - \alpha \mid j, k \in \{\top_{++}, \top'_{++}\}, j \neq k\}$ $\{\textsc{Add}(1, -4) - \textsc{Move}(j, k) - \alpha \mid j, k \in \{\top_{+0}, \top'_{+0}\}, j \neq k\}$ $\{\textsc{Add}(2, -1) - \textsc{Move}(j, k) - \alpha \mid j, k \in \{\top_{0+}, \top'_{0+}\}, j \neq k\}$
	$\{\textsc{Add}(1, -4) + \textsc{Add}(2, -1) + \textsc{Move}(j, k) - \alpha \mid j \in \{\top_{++}, \top'_{++}\}, k \in \mathcal{T}, j \neq k\}$
	$\{\textsc{Add}(1, -4) + \textsc{Add}(2, -1) + \textsc{Move}(j, 1) - \alpha \mid j \in \{\top_{++}, \top'_{++}\}\}$
	$\{\textsc{Add}(1, -4) + \textsc{Move}(j, 1) - \alpha \mid j \in \{\top_{+0}, \top'_{+0}\}\}$
	$\{\textsc{Add}(2, -1) + \textsc{Move}(j, 1) - \alpha \mid j \in \{\top_{0+}, \top'_{0+}\}\}$
$\textsc{Check}(i)$	$\{\textsc{Add}((1, -4e_1) + \textsc{Add}(2, -e_2) - \textsc{Check}(i) \mid e_1, e_2 \in \{0, 1\}\}$
	$\{\textsc{Add}(1, -4) + \textsc{Add}(2, 1) + \textsc{Move}(j, k) - \textsc{Check}(i) \mid i, j \neq k,$ $j \in \{\top_{++}, \top'_{++}\}, k \in \mathcal{T}\}$
	$\{\textsc{Add}(1, -4) + \textsc{Add}(2, -1) + \textsc{Move}(j, 1) - \textsc{Check}(i) \mid j \in \{\top_{++}, \top'_{++}\}\}$
	$\{\textsc{Add}(1, -4) + \textsc{Move}(j, 1) - \textsc{Check}(i) \mid j \in \{\top_{+0}, \top'_{+0}\}\}$
	$\{\textsc{Add}(2, -1) + \textsc{Move}(j, 1) - \textsc{Check}(i) \mid j \in \{\top_{0+}, \top'_{0+}\}\}$

Next we prove which player has a winning strategy in the scenarios presented previously.

▶ **Lemma 10.** *If both players only play* REGULAR MOVES *and Eve plays only correct* REGULAR MOVES*, then Eve has a winning strategy if and only if she has a winning strategy in two-dimensional robot games with states.*

Proof. It easy to see that REGULAR MOVES of the players simulate the 2RGS and that Eve has a winning strategy to reach a configuration $(s_{00}, (0, 0))$ of the 2RGS if and only if she has a winning strategy to reach the vector $(0, 0 \cdot 4 \cdot 8^n + 8^{s_{00}} - 8^{\top_{00}})$ in 2RG after which Eve wins by playing $\textsc{Move}(s_{00}, \top_{00}) - \alpha$, where α is the REGULAR MOVE played by Adam. ◀

▶ **Lemma 11.** *If Eve plays an incorrect move, i.e., after her turn the coefficient of some 8^s is -1 or the coefficient of $8^{\top_{00}}$ is zero, then Adam has a winning strategy starting with a* STATE-CHECK*.*

Proof. First, we prove that Eve loses if a coefficient corresponding to a state of 2RGS is negative after one of her turns. A coefficient corresponding to a state of 2RGS can only be increased, namely incremented, by Eve's REGULAR MOVES. Hence, if one of the coefficients becomes negative, then Adam wins by playing a STATE-CHECK move. The reasoning is now similar to the usage of the POSITIVITY CHECK in Lemma 4. We consider the second counter modulo $4 \cdot 8^n$. Before Adam's STATE-CHECK, the configuration is in $[0, 8^n) \pmod{4 \cdot 8^n}$ and after the check in $[3 \cdot 8^n, 4 \cdot 8^n) \pmod{4 \cdot 8^n}$. If Eve does not play a STATE-DEFENCE MOVE (a move containing a $\textsc{Check}(i)$), then Adam has a winning strategy by playing a STATE-CHECK if the second counter is not in $[3 \cdot 8^n, 4 \cdot 8^n) \pmod{4 \cdot 8^n}$ and a REGULAR MOVE otherwise (recall that Adam's REGULAR MOVES do not modify the second counter). Thus Eve has to play a STATE-DEFENCE MOVE which does not make the negative coefficient non-negative. Now at least one of the coefficients in $\mathcal{T}$ is non-zero, say i. Adam will play $\textsc{Check}(i)$ forcing Eve to play a move containing $-\textsc{Check}(i)$ which will make another coefficient in $\mathcal{T}$ non-zero. As long as Adam keeps playing the correct STATE-CHECK, Eve cannot make all the coefficients zero and thus cannot win.

The second case where a coefficient of some state in $\mathcal{T}$ is negative has been proven above. For the final case, where the coefficient of $8^{\top_{00}}$ is zero, we consider the next move of Eve. During her next turn, Eve has to play a move containing $\textsc{Move}(s,t)$ making the coefficient of 8^s negative, which has been covered previously. ◀

▶ **Lemma 12.** *Assume that Eve plays only correct* REGULAR MOVES *before Adam plays a* STATE-CHECK *for the first time. If Adam plays a* STATE-CHECK, *then Eve has a winning strategy starting with a* STATE-DEFENCE MOVE.

Proof. Similarly as in the previous proof, if Eve does not play a STATE-DEFENCE MOVE, then Adam has a winning strategy. Now, Eve plays the STATE-DEFENCE MOVE $\textsc{Move}(s_{ab},k) - \textsc{Check}(i)$ where s_{ab} is the non-zero coefficient, $\textsc{Check}(i)$ is the STATE-DEFENCE MOVE Adam played and $k \in \{\top_{ab}, \top'_{ab}\}$, $k \neq i$. From that point onward, Eve can empty the counters ensuring as she has emptying moves with an opposite move of Adam. Eventually, Eve will reach the configuration $(0,0)$ and win the game. ◀

▶ **Lemma 13.** *If Adam plays only* REGULAR MOVES *and Eve plays a* STATE-DEFENCE MOVE, *then Adam has a winning strategy starting with a* REGULAR MOVE.

Proof. Since all STATE-DEFENCE MOVES subtract -8^n from the second counter, after Eve's move, the counter is in $[8^n, 2\cdot 8^n) \pmod{4\cdot 8^n}$. As in proof of Lemma 11, Adam ensures that the second counter does not return to the interval $[0, 8^n) \pmod{4\cdot 8^n}$. ◀

Proof of Theorem 8. Let (A,E) be the robot game constructed in this section. Assume first that Eve has a winning strategy in (A_1,E_1). Now, Eve's winning strategy in two-dimensional robot games is to follow the strategy of (A_1,E_1) as long as Adam plays REGULAR MOVES which is a winning strategy by Lemma 10. If Adam plays a STATE-CHECK, then Eve responds according to the winning strategy of Lemma 12.

Assume then that Adam has a winning strategy in (A_1,E_1) and Eve has a winning strategy in (A,E). If Adam plays only REGULAR MOVES, then by Lemma 10, Eve does not win by playing just the correct the correct SIMULATING MOVES. That is, Eve has to, at some point, either play an incorrect SIMULATION MOVE or play a STATE-DEFENCE MOVE. By Lemmas 11 and 13, Adam has winning strategies for both cases. As we analysed all the possible moves of Eve, we have shown that Eve does not have a winning strategy. ◀

▶ **Corollary 14.** *Let (A,E) be a two-dimensional robot game and an initial vector $\mathbf{x}_0$. It is undecidable whether Eve has a winning strategy to reach $(0,0)$ from $\mathbf{x}_0$.*

Corollary 14 follows from Corollary 7 and Theorem 8. It is possible to apply it to matrix games introduced in [11] to show undecidability in $\mathbb{Z}^{3\times 3}$.

Final remarks: The construction of robot games with states was first presented in the PhD thesis of one of the authors, [18], where it was also proved that robot games in dimension three are undecidable. The undecidability of 2RG is proved by a new technique of embedding state transitions of a 2CM into integers. It would be interesting to see whether the same approach can be applied to other automata and games, such as stateless VASS games.

Korec showed in [14] that there exists a universal Minsky machine with 32 instructions. The natural question of a universal game arises: Is it possible to construct a fixed robot game simulating a universal 2CM? This game would have fixed moves and only the initial vector would affect the result. In [12], it was proven that two-dimensional robot games where both players have two moves are decidable in polynomial time. Consider the machine with

32 instructions. We can construct a robot game from it and count the number of moves. Thus it is undecidable whether Eve has a winning strategy in a two-dimensional robot game where Eve has at least 2083 moves and Adam has 8 moves.

References

1 Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d'Orso. Deciding monotonic games. In *Proceedings of CSL 2003*, volume 2803 of *LNCS*, pages 1–14, 2003. doi:10.1007/978-3-540-45220-1_1.

2 Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d'Orso. Monotonic and downward closed games. *J. Log. Comput.*, 18(1):153–169, 2008. doi:10.1093/logcom/exm062.

3 Parosh Aziz Abdulla, Richard Mayr, Arnaud Sangnier, and Jeremy Sproston. Solving parity games on integer vectors. In *Proceedings of CONCUR 2013*, volume 8052 of *LNCS*, pages 106–120, 2013. doi:10.1007/978-3-642-40184-8_9.

4 Arjun Arul and Julien Reichert. The complexity of robot games on the integer line. In *Proceedings of QAPL 2013*, volume 117 of *EPTCS*, pages 132–148, 2013. doi:10.4204/EPTCS.117.9.

5 Tomás Brázdil, Václav Brozek, and Kousha Etessami. One-counter stochastic games. In *In proceedings of FSTTCS 2010*, volume 8 of *LIPIcs*, pages 108–119, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.108.

6 Tomáš Brázdil, Petr Jančar, and Antonín Kučera. Reachability games on extended vector addition systems with states. In *Proceedings of ICALP 2010*, volume 6199 of *LNCS*, pages 478–489, 2010. doi:10.1007/978-3-642-14162-1_40.

7 Jakub Chaloupka. Z-reachability problem for games on 2-dimensional vector addition systems with states is in P. *Fundam. Inform.*, 123(1):15–42, 2013. doi:10.3233/FI-2013-798.

8 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. doi:10.1016/j.tcs.2012.07.038.

9 Laurent Doyen and Alexander Rabinovich. Robot games. Personal website, 2011. Technical Report LSV-13-02, LSV, ENS Cachan, 2013. URL: http://www.lsv.ens-cachan.fr/Publis/RAPPORTS%5FLSV/PDF/rr-lsv-2013-02.pdf.

10 Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jirí Srba. Energy games in multiweighted automata. In *Proceedings of ICTAC 2011*, volume 6916 of *LNCS*, pages 95–115, 2011. doi:10.1007/978-3-642-23283-1_9.

11 Vesa Halava, Tero Harju, Reino Niskanen, and Igor Potapov. Weighted automata on infinite words in the context of Attacker-Defender games. In *Proceedings of CiE 2015*, volume 9136 of *LNCS*, pages 206–215, 2015. doi:10.1007/978-3-319-20028-6_21.

12 Vesa Halava, Reino Niskanen, and Igor Potapov. On robot games of degree two. In *Proceedings of LATA 2015*, volume 8977 of *LNCS*, pages 224–236, 2015. doi:10.1007/978-3-319-15579-1_17.

13 Paul Hunter. Reachability in succinct one-counter games. In *Proceedings of RP 2015*, volume 9328 of *LNCS*, pages 37–49, 2015. doi:10.1007/978-3-319-24537-9_5.

14 Ivan Korec. Small universal register machines. *Theor. Comput. Sci.*, 168(2):267–301, 1996. doi:10.1016/S0304-3975(96)00080-1.

15 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975. doi:10.2307/1971035.

16 Marvin L Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., 1967.

17 Reino Niskanen, Igor Potapov, and Julien Reichert. Undecidability of two-dimensional robot games. *CoRR*, abs/1604.08779, 2016. URL: http://arxiv.org/abs/1604.08779.

18 Julien Reichert. *Reachability Games with Counters: Decidability and Algorithms.* Doctoral thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, 2015.

19 Julien Reichert. On the complexity of counter reachability games. *Fundam. Inform.*, 143(3-4):415–436, 2016. doi:10.3233/FI-2016-1320.

Algebraic Independence over Positive Characteristic: New Criterion and Applications to Locally Low Algebraic Rank Circuits

Anurag Pandey[1], Nitin Saxena[2], and Amit Sinhababu[3]

1 **MPI for Informatics & Saarland University, Department of Computer Science, Saarbrücken, Germany**
apandey@mpi-inf.mpg.de
2 **Department of CSE, Indian Institute of Technology Kanpur, Kanpur, India**
nitin@cse.iitk.ac.in
3 **Department of CSE, Indian Institute of Technology Kanpur, Kanpur, India**
amitks@cse.iitk.ac.in

Abstract

The motivation for this work comes from two problems– test algebraic independence of arithmetic circuits over a field of small characteristic, and generalize the structural property of algebraic dependence used by (Kumar, Saraf CCC'16) to arbitrary fields.

It is known that in the case of zero, or large characteristic, using a classical criterion based on the Jacobian, we get a randomized poly-time algorithm to test algebraic independence. Over small characteristic, the Jacobian criterion fails and there is no subexponential time algorithm known. This problem could well be conjectured to be in RP, but the current best algorithm puts it in $\mathrm{NP}^{\#\mathsf{P}}$ (Mittmann, Saxena, Scheiblechner Trans.AMS'14). Currently, even the case of two bivariate circuits over $\mathbb{F}_2$ is open. We come up with a natural generalization of Jacobian criterion, that works over all characteristic. The new criterion is efficient if the underlying *inseparable degree* is promised to be a constant. This is a modest step towards the open question of fast independence testing, over finite fields, posed in (Dvir, Gabizon, Wigderson FOCS'07).

In a set of linearly dependent polynomials, any polynomial can be written as a linear combination of the polynomials forming a basis. The analogous property for algebraic dependence is false, but a property approximately in that spirit is named as "functional dependence" in (Kumar, Saraf CCC'16) and proved for zero or large characteristic. We show that functional dependence holds for *arbitrary* fields, thereby answering the open questions in (Kumar, Saraf CCC'16). Following them we use the functional dependence lemma to prove the first exponential lower bound for *locally low algebraic rank* circuits for *arbitrary* fields (a model that strongly generalizes homogeneous depth-4 circuits). We also recover their quasipoly-time hitting-set for such models, for fields of characteristic smaller than the ones known before.

Our results show that approximate functional dependence is indeed a more fundamental concept than the Jacobian as it is field independent. We achieve the former by first picking a "good" transcendence basis, then translating the circuits by new variables, and finally approximating them by truncating higher degree monomials. We give a tight analysis of the "degree" of approximation needed in the criterion. To get the locally low algebraic rank circuit applications we follow the known shifted partial derivative based methods.

1998 ACM Subject Classification I.1 Symbolic and Algebraic Manipulation, F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases independence, transcendence, finite field, Hasse-Schmidt, Jacobian, differential, inseparable, circuit, identity testing, lower bound, depth-4, shifted partials

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.74

1 Introduction

Algebraic dependence is a fundamental concept in algebra that captures algebraic/polynomial relationship of objects like numbers, polynomials, rational functions or power series, over some field. Here we define algebraic dependence of polynomials, since in this work we deal only with polynomials. Polynomials $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$ are called *algebraically dependent* over field k if and only if there exists a nonzero polynomial $A(y_1, \ldots, y_m) \in \mathbb{F}[y_1, \ldots, y_m]$ such that $A(f_1, \ldots, f_m) = 0$ and such an A is called an *annihilating polynomial* of $f_1, \ldots, f_m$. If no such nonzero polynomial A exists, the given polynomials are called *algebraically independent* over k.

For example, $f_1 = (x + y)^2$ and $f_2 = (x + y)^3$ are algebraically dependent over any field, as $y_1^3 - y_2^2$ is an annihilating polynomial. Polynomials $x + y$ and $x^p + y^p$ are dependent over $\mathbb{F}_p$, but independent over $\mathbb{Q}$. Monomials $x_1, \ldots, x_n$ are examples of algebraically independent polynomials over any field.

Algebraic dependence can be viewed as a generalization of linear dependence as the former captures algebraic relationships of any degree, whereas the latter captures linear relationships. Algebraic dependence shares a few combinatorial properties (known as matroid properties [30]) with linear dependence. For example, if a set of polynomials are algebraically independent then any subset of them are algebraically independent. The *transcendence degree* (trdeg or algRank) of a set of polynomials is defined as the maximal number of algebraically independent polynomials and it is well defined thanks to the matroid properties. The concepts of rank and basis in linear algebra have analogs here as transcendence degree and *transcendence basis* respectively.

The concept of algebraic independence is useful in several areas of mathematics: field theory, commutative algebra, algebraic geometry, invariant theory, theory of algebraic matroids. It has found interesting applications in computer science as well. For example, [28] used algebraic dependence in analysis of program invariants of arithmetic straight line programs. To prove lower bounds on the formula size of determinant, [21] also used transcendence degree as a tool. [7, 9] constructed explicit deterministic randomness extractors for sources which are polynomial maps over finite fields. [8] gives a cryptography application, using algebraic characterization of entropy of low degree polynomials. [4, 1, 27] used it for designing faster deterministic hitting-sets for some interesting cases of the polynomial identity testing problem (PIT) and proving circuit lower bounds. [5] used algebraic independence of polynomials to show the hardness of a parameterized counting problem.

An example relevant to computer science is to compute the "entropy" of a given polynomial map $\phi : (x_1, \ldots, x_n) \mapsto (f_1, \ldots, f_n)$ in the space $\mathbb{F}_q^n$, where q is a power of $p = 2$ (more, generally, p grows as a polynomial in the input size). This turns out to be a question of computing the transcendence degree of the polynomials $f_1, \ldots, f_n$ [7]. For constant p, there are no good methods known. Our work improves the state of the art in this regime.

To discuss the complexity of algebraic independence testing, we have to specify the representation of input polynomials. An *arithmetic circuit* is a directed acyclic graph consisting of addition $(+)$ and multiplication $(\times)$ gates as nodes, takes variables $x_1, \ldots, x_n$ and field constants as input (leaves), and outputs a polynomial $f(x_1, \ldots, x_n)$. This is a succinct representation of multivariate polynomials, as polynomials of high degree (or having many monomials) can be represented by small circuits.

Perron [31, 32] gave a bound on the degree of the minimal annihilating polynomial, proving that it is bounded by the product of the degrees of the input polynomials. This bound was subsequently slightly improved in [22, 4]. Perron's bound gives us the brute-force approach.

It reduces the problem of computing annihilating polynomial to solving an exponential sized system of linear equations and this can be done in PSPACE. Thus, PSPACE is the "trivial" complexity upper bound for algebraic independence testing, over any field.

The degree bound on the minimal annihilating polynomial happens to be tight. We can give examples of n quadratic polynomials, such that the degree of their minimal annihilating polynomial is 2^n [22]. There is a hardness result known [22], that shows that computing even the constant term of the annihilating polynomial is NP-hard, and that annihilating polynomial is not of polynomial size in general, unless the polynomial hierarchy collapses.

It turns out that the decision version, i.e. checking if the polynomials are algebraically independent, is much more efficient over zero or large characteristic, even when the polynomials are succinctly represented as circuits. The key idea is a classical result, known as the Jacobian criterion [20, 4]. It says that if the characteristic of the field is zero, or large enough (compared to the product of degrees of the given polynomials), then the transcendence degree equals the linear rank of the Jacobian matrix of the polynomials. This leads to a simple randomized poly-time algorithm for checking algebraic independence, as we can get the circuits of the partial derivatives efficiently [3] and then use random evaluations to compute the rank of the Jacobian matrix. This final step of randomized evaluation is possible due to the Schwartz-Zippel-DeMillo-Lipton lemma [36, 6, 40].

One direction of the Jacobian criterion (if the polynomials are algebraically dependent, then their Jacobian matrix is not full rank) holds true for any characteristic. But the converse fails if the characteristic is small compared to the product of the degrees of input polynomials. For example, x^p is algebraically independent of $\mathbb{F}_p$, yet its derivative vanishes. We remark here that if two algebraically independent polynomials over characteristic p have zero Jacobian, then it does not mean that one of them is a p power. Consider, for example, $\{x^{p-1}y, xy^{p-1}\}$ over $\mathbb{F}_p$ for prime $p > 2$.

There are infinitely many input instances (set of polynomials), where the Jacobian criterion fails, i.e. Jacobian vanishes even though the given polynomials are independent. Those instances can be characterized by the notion of *inseparable extension*, that appears in Galois theory, and is formally defined in Sec.2.1. For example, the field extension $\mathbb{F}_p(x)/\mathbb{F}_p(x^p)$ has inseparable degree p as that many *conjugates* of $\sqrt[p]{x^p}$ in the splitting field are equal. This is a hard algebraic situation with no good geometric interpretation. Such behavior is absent over zero characteristic fields. So, positive characteristic requires inventing new concepts.

Naturally, we would like to come up with an efficient (randomized poly-time) algorithm over small characteristic. Though the failure of Jacobian criterion over small characteristic is known for long [12, 15], owing to the interest in algebraic independence from computer science perspective, several recent papers [7, 22, 4] posed the complexity status of this problem (whether it is in RP) as an open question. One curious aspect is that this problem is one of the rare ones in computer science where the gap between the known time complexity (EXP) and the conjectured one (RP) is that stark!

Talking about the two degrees. Let us consider a case where Jacobian criterion fails and certifying independence gets tricky. Let m_1m_2 be coprime to p, and $f_1 = x_1^{pm_1}, f_2 = x_2^{m_2}$. It is easy to deduce that the degree of the extension $\mathbb{F}_p(x_1, x_2)/\mathbb{F}_p(f_1, f_2)$ is pm_1m_2. In fact, the degree of the annihilating polynomial of $\{x_1, f_1, f_2\}$ (resp. $\{x_2, f_1, f_2\}$) is pm_1 (resp. m_2). However, the inseparable degree of the extension is only p, as the former annihilating polynomial (i.e. $y_1^{pm_1} - y_2$) is a polynomial in y_1^p but not in $y_1^{p^2}$. Thus, there are cases when the inseparable degree can be much smaller, even $O(1)$, compared to the extension degree. Notice that, in general, the inseparable degree is a p-power that divides the extension degree, which in turn is upper bounded by $\prod_i \deg(f_i)$ (by Perron's bound)– usually an exponentially large parameter. The methods developed in this work only depend on the underlying inseparable degree, thus, our algorithm is expected to be much better than brute-force (in many cases).

A criterion that works for all characteristic for a natural problem like testing algebraic independence would be mathematically interesting. Computational implications of an efficient Jacobian like criterion would include a possible generalization (to small characteristic) of PIT or lower bound results [1], and algebraic extractors or entropy concepts [7].

Work done in case of finite fields. [29] gave a criterion that works over all fields, which they named *Witt-Jacobian* criterion. One key idea of the Witt-Jacobian criterion is to lift the input polynomials from characteristic $p \geq 2$ to a field of p-adics, which is zero characteristic. Witt-Jacobian polynomial can be seen as a scaled up p-adic lift of Jacobian polynomial and the criterion involves checking certain monomials (degeneracy testing; which looks hard) rather than zero testing. The main object underlying the proof is the de Rham-Witt pro-complex; a tool from modern algebraic-geometry (an excellent survey is [18]).

Witt-Jacobian criterion improved the complexity of independence testing problem, over positive characteristic, from PSPACE to $\mathrm{NP}^{\#\mathsf{P}}$. In the hierarchy of complexity classes, $\mathrm{NP}^{\#\mathsf{P}}$ is far above RP; thus there is a huge gap between what we have and what we want.

Partial derivative (defined as formal operators on polynomials), that played a key role in Jacobian criterion, behaves strangely over positive characteristic. Though it satisfies the usual rules of derivatives like linearity, product rule and chain rule, one important difference here is the fact that a non-constant polynomial can have a zero derivative. Another difference is that the higher derivatives of order $k \geq p$ are zero for all polynomials over characteristic p. Hasse-Schmidt derivatives are variants of usual derivatives, that were originally defined by [17], and independently by [38], to tackle this problem. In computer science literature, Hasse derivatives were used recently in coding theory (see [10] and the references therein), and PIT or lower bounds via generalized versions of shifted partial derivatives [14, 13].

Background on PIT and circuit lower bounds. The problems of derandomization of PIT and proving lower bounds, for explicit family of polynomials, are two fundamental questions in complexity theory. The question of PIT asks to test whether the polynomial computed by an arithmetic circuit is identically zero. This question can be studied in two settings. In the whitebox setting we are allowed to see inside the circuit, whereas in the blackbox setting we can only evaluate the circuit at some field points. The problem of blackbox PIT is equivalent to the problem of designing *hitting-sets* efficiently. Hitting-set is defined as follows. Let $\mathcal{C}$ be a class of polynomials in N variables over a field $\mathbb{F}$. Then, a set $\mathcal{H} \subseteq \mathbb{F}^N$ is called a *hitting-set* for the class $\mathcal{C}$, if for every nonzero polynomial $C \in \mathcal{C}$, there exists an $x \in \mathcal{H}$ such that $C(x) \neq 0$. PIT has a randomized poly-time algorithm, thanks to Schwartz-Zippel-DeMillo-Lipton lemma [36, 40, 6]. Derandomization of PIT is an outstanding open question in complexity theory with several implications, including proving arithmetic circuit lower bounds (refer to [2] & the survey [37]).

In the world of arithmetic complexity, we have strong structural results like *depth reductions* [16, 2]. These results show that strong enough lower bound, or PIT, results for homogeneous depth-4 (or general depth-3) circuits would give us exponential lower bounds and quasipoly-time derandomized PIT for general circuits (up to VP). Recent years have seen a fast growth in papers giving lower bound and PIT results for several special cases of small depth arithmetic circuits [34, 35]. Although there are strong (almost exponential, [26, 23]) lower bounds for homogeneous depth-4 circuits, the best known lower bounds for non-homogeneous depth-4 circuits are only superlinear (see [33] & the references therein).

Circuits with locally low algebraic rank. Kumar & Saraf [27] defined a *locally low algebraic rank* circuit of degree n in N variables over $\mathbb{F}$, denoted $\Sigma\Gamma^{(k)}\Sigma\Pi^{d}$, as: $C = \sum_{i\in[T]} \Gamma_i(Q_{i1},\ldots,Q_{it})$, where Q_{ij} is a sparse polynomial (all monomials are given explicitly) of degree at most d, algRank of $\{Q_{ij} \mid j \in [t]\}$ is at most k, and Γ_i is an arbitrary t-variate polynomial, for $i \in [T]$.

The *size* of C comprises N, n, T and the maximum sparsity of Q_{ij}'s. Note that $k \le N$, and we will be interested in the cases when kd is somewhat restricted.

Interestingly, $\Sigma\Gamma^{(n)}\Sigma\Pi$ subsumes homogeneous depth-4 circuits computing a degree n polynomial, as for homogeneous circuits $k \le t \le n$ and Γ_i is merely the product gate. Since this class includes non-homogeneous circuits as well (where t can be arbitrarily larger than k, n), it can be seen as a significant generalization of homogeneous depth-4.

This model subsumes certain other interesting models that were studied by [14, 1, 4] in the context of lower bounds and PIT. Invariably, their methods need to assume that $\mathbb{F}$ has characteristic zero or exponentially large (since partial derivatives are involved). Our goal in this paper is to overcome such restrictions.

1.1 Our contribution and relation with previous works

Broadly, in this paper, we prove two main technical theorems, one about the algebraically dependent polynomials and the other about algebraically independent polynomials. We apply these two theorems to obtain an algebraic independence testing algorithm, an arithmetic circuit lower bound over arbitrary field and a PIT algorithm (over fields of characteristic larger than the individual-degree of the polynomial). We now describe each of the results.

Algebraic dependence to approximate functional dependence. We show that over arbitrary fields, algebraic dependence of polynomials $f_1,\ldots,f_m$ imply the existence of a transcendence basis such that all the polynomials $f_1,\ldots,f_m$ can be obtained (upto a random shift and a truncation) as a polynomial function of the basis elements (Thm.10). Essentially, to obtain the desired polynomial, say f_k, we truncate a polynomial function in the elements of the basis upto the degree of f_k. This generalizes the functional dependence result of [27, Lem.3.1] which asserted the same over fields of zero (or large) characteristic.

We use a proof approach which is different from [27] to achieve the more general results. In the case of fields of zero characteristic, the subtle strength that this functional dependence property possesses is that *any* transcendence basis serves the purpose, which in general is false over positive characteristic. Our result explains this subtlety using the concept of *separating transcendence basis* from Galois theory (Sec.2.1). With this, a simple algebraic manipulation on the annihilating polynomial, and subspace of polynomial products (Lem.12), yields a functional dependence up to *any* desired degree of approximation. (This is a bit simpler than the approach of [27, Lem.2.4] where they approximate the roots of any multivariate polynomial using [11, Lem.3.1]. Such methods also appear in classical analysis under *Implicit Function Theorems*, see [25].)

Eg. $\{x_1, x_2, x_1x_2^2\}$ are algebraically dependent over $\overline{\mathbb{F}}_2$. Pick random field elements a_1, a_2. The shifted polynomials are $\{x_1 + a_1, x_2 + a_2, (x_1 + a_1)(x_2^2 + a_2^2)\}$. Clearly, $(x_2 + a_2)$ is not a function of the other two modulo the ideal $\langle \mathbf{x}\rangle^2$. However, $(x_1 + a_1)$ is trivially a function of the other two, namely, $(x_1 + a_1) \equiv a_2^{-2} \cdot (x_1 + a_1)(x_2^2 + a_2^2) \mod \langle \mathbf{x}\rangle^2$.

Algebraically independent polynomials – Criterion. The above example shows that over fields of positive characteristic, an approximate functional dependence may exist even in the case of algebraically independent polynomials. We overcome this issue and show that the independence can be captured by truncating the polynomial function in the basis elements upto a precise parameter, i.e. if we choose the truncation point to be greater than that parameter, then algebraically independent polynomials *cannot* exhibit functional dependence (Thm.13). This parameter is actually the *inseparable degree* of an appropriate field extension, which is a well studied concept in Galois theory (Sec.2.1).

Continuing the above example– $\{x_1, x_1x_2^2\}$ are algebraically independent over $\overline{\mathbb{F}}_2$. Pick random field elements a_1, a_2. The shifted polynomials are $\{x_1 + a_1, (x_1 + a_1)(x_2^2 + a_2^2)\}$. It can be verified that neither is a polynomial function of the other modulo the ideal $\langle \mathbf{x} \rangle^3$. This becomes a certificate of algebraic independence. (Note that the inseparable degree of $\mathbb{F}_2(x_1, x_2)/\mathbb{F}_2(x_1, x_1x_2^2)$ is 2.)

When the inseparable degree is 1 (which means a *separable extension*), then looking at the truncation upto the linear term of shifted basis elements would suffice. So, our result implies that *separable extension* is precisely the case when the Jacobian works (an exposition can be found in the full version). For higher inseparable degree t, our result can be reinterpreted as giving a Jacobian like result: algebraically independent polynomials have $\mathbb{F}(\mathbf{z})$-linearly independent higher differentials (Sec.2.2), modulo a carefully chosen subspace $\mathcal{U}_t$ (Rmk.11). This follows by considering the Taylor series, around a "generic" point $\mathbf{z}$, whence, the functional independence of polynomials shifted by $\mathbf{z}$, implies the linear independence of shifted polynomials modulo $\mathcal{U}_t$. As shifted polynomials contain all the Hasse-Schmidt higher derivatives (wrt $\mathbf{x}$ and evaluated at the point $\mathbf{z}$), we deduce their $\mathbb{F}(\mathbf{z})$-linear independence modulo $\mathcal{U}_t$.

Again, a key technical lemma used in finishing the proof is Lem.12 (*subspace reduction*), which concerns the ideal theoretic properties of the subspace $\mathcal{U}_t$. Basically, it helps us prove that if $\{h_1, \ldots, h_n\}$ are polynomials with their degree($\leq t$)-part having algebraically independent leading monomials, and g_n functionally depends on $\{g_1, \ldots, g_{n-1}\}$ (with truncation beyond t), then some h_i is functionally independent of $\{g_1, \ldots, g_n\}$.

Application 1: Testing algebraic independence. An easy consequence of Thm.10 and Thm.13 is that we get a randomized poly-time algorithm for testing algebraic independence of polynomials over finite fields (say, $\mathbb{F}_q$ of characteristic p) in the cases when the inseparable degree is constant. Since the latter is a p-power (Sec.2.1), our algorithm is interesting when p is a constant. (Whenever required, we can assume wlog that the input is n circuits in n variables over an algebraically closed field; see full version for simple proofs.)

▶ **Theorem 1** (Independence testing). *For circuits $\mathbf{f} \in \mathbb{F}_q[\mathbf{x}]$, we have a randomized poly(s, $\binom{t+n}{n}$)-time algebraic independence testing algorithm, where the inseparable degree of the field extension $\mathbb{F}_q(\mathbf{x})/\mathbb{F}_q(\mathbf{f})$ is t (assuming $\mathbf{f}$ algebraically independent) & input size is s.*

This covers a lot of interesting cases as the inseparable degree can be quite small even in case of polynomials with exponential degree. As a simple example, take two bivariate circuits of exponential degree over $\mathbb{F}_2$. Suppose they are independent and their Jacobian is nonzero. Now if we square any one of these two, then Jacobian would fail as the inseparable degree becomes 2. Previously known algorithms cannot deal with even such a simple case, whereas we easily handle the case by trying our test with $t = 2$. In general, the inseparable degree is upper bounded by Perron's degree bound (product of degrees of given polynomials, [32]), so in the worst-case our algorithm is exponential-time. (Witt-Jacobian criterion [29] is exponential-time in all cases.) We illustrate the overall idea, and its comparison with Jacobian criterion, in the figure in the conclusion (Sec.4).

An interesting by-product of the algorithm is that it computes the inseparable degree, of the given independent polynomials, in the same time.

Application 2: Lower bound for locally low algebraic rank circuits. Using the functional dependence result, we give an explicit family of polynomials in VNP of degree n in N variables, where $N = n^{O(1)}$ such that any $\Sigma\Gamma^{(n)}\Sigma\Pi$ circuit computing it has size $N^{\Omega(\sqrt{n})}$. We obtain this lower bound over *arbitrary* fields. This generalizes the lower bound of [27, Thm.1.4] which itself was a strong generalization of the shifted partials based homogeneous depth-4 lower bounds [23] and Jacobian based lower bounds [1] (all over zero or large characteristic). Since our functional dependence generalizes the key technical ingredient of [27] to arbitrary fields, we are able to get the same lower bound (for the same model and hard polynomial family) over arbitrary fields. Formally,

▶ **Theorem 2.** *Let $\mathbb{F}$ be any field. There exists a family $\{P_n\}$ of polynomials in VNP, such that P_n is a polynomial of degree n in $N = n^{O(1)}$ variables with $0, 1$ coefficients, and for any $\Sigma\Gamma^{(k)}\Sigma\Pi$ circuit C, if $k \leq n$ and if C computes P_n over $\mathbb{F}$, then Size$(C) \geq N^{\Omega(\sqrt{n})}$.*

▶ Remark. As remarked by [27], the above model is challenging even for $k = 2$ (& was open before us for small characteristic fields). Also, the proof goes through for any $k = n^{O(1)}$, as long as one picks N as an appropriately large polynomial in n.

The proof of this theorem closely follows [27], and is sketched in the full version.

Application 3: Hitting-set for $\Sigma\Gamma^{(k)}\Sigma\Pi^d$ circuits. We show that for any size-s circuit $C \in \Sigma\Gamma^{(k)}\Sigma\Pi^d$, where $k, d = \text{polylog}(s)$, over fields of characteristic $p >$ individual-degree(C), there exists a quasipoly(s)-time hitting-set.

▶ **Theorem 3.** *Let $\mathbb{F}$ be any field of characteristic p. There exists an $\exp(\log^{O(1)} s)$-time constructible hitting-set $\mathcal{H} \subseteq \overline{\mathbb{F}}^N$ for size-s circuit $C \in \Sigma\Gamma^{(k)}\Sigma\Pi^d$ with $kd = \log^{O(1)} s$, assuming $p >$ individual-degree(C) or $p = 0$.*

Again, the proof follows [27]. For PIT, algebraic rank based models have already been considered by [4, 1, 27]. Our result generalizes some of these results to smaller positive characteristic (only requiring $p >$ individual-degree(C)). The previous results required $p > d^k$, which is super-polynomial in the above regime. Our inability to remove this restriction lies in the nature of shifted partials [14, Lem.4.13]. Eg. the dimension of shifted partials of a p-power monomial $x_1^{p^{e1}} \cdots x_n^{p^{en}}$ is not that large over $\mathbb{F}_p$.

2 Preliminaries: Jacobi, Galois and Hasse-Schmidt

We define the central object related to the testing of algebraic independence is the Jacobian.

▶ **Definition 4** (Jacobian). The *Jacobian* of polynomials $\mathbf{f} = \{f_1, \cdots, f_m\}$ in $\mathbb{F}[x_1, \cdots, x_n]$ is the matrix $\mathcal{J}_{\mathbf{x}}(\mathbf{f}) = (\partial_{x_j} f_i)_{m \times n}$, where $\partial_{x_j} f_i := \partial f_i / \partial x_j$.

We state the classical Jacobian criterion [20, 4].

▶ **Lemma 5** (Jacobian criterion). *Let $\mathbf{f} \subset \mathbb{F}[\mathbf{x}]$ be a finite set of polynomials of degree at most d, and $\text{trdeg}_{\mathbb{F}} \mathbf{f} \leq r$. If $\text{char}(\mathbb{F}) = 0$, or $\text{char}(\mathbb{F}) > d^r$, then $\text{trdeg}_{\mathbb{F}} \mathbf{f} = \text{rank}_{\mathbb{F}(\mathbf{x})} \mathcal{J}_{\mathbf{x}}(\mathbf{f})$.*

Previously, we saw some examples of polynomials over fields of smaller characteristic where the Jacobian *fails*. Here is another nontrivial example: $\mathbf{f} = \{x_1^2x_2 + x_1^3,\ x_1x_2^2 + x_1x_2^5\}$ in $\mathbb{F}_3[x_1, x_2]$ is a set of algebraically independent polynomials, but $\operatorname{rank}_{\mathbb{F}_3(\mathbf{x})}\mathcal{J}_{\mathbf{x}}(\mathbf{f}) = 1$, and hence the criterion fails.

2.1 Inseparability & separating transcendence basis

For this section, let $\mathbb{E} \supseteq \mathbb{F}$ be fields. Failure of the Jacobian criterion can be explained using the fundamental concept of inseparability from Galois theory [19].

▶ **Definition 6.** An $f \in \mathbb{F}[x]$ is *separable* if it has no multiple roots in its splitting field.

It is easy to prove that. For an irreducible f, separability is implied by the non-zeroness of $\partial_x f$. Thus, if $\operatorname{char}(\mathbb{F}) = 0$, then any irreducible polynomial is separable. It further implies that if $\operatorname{char}(\mathbb{F}) = p > 0$ then, an irreducible f is separable if and only if $f \notin \mathbb{F}[x^p]$. We have this notion of separability in case of field extensions as well. An algebraic extension $\mathbb{E}/\mathbb{F}$ is said to be *separable* if every element $\alpha \in \mathbb{E}$ has a minimal polynomial over $\mathbb{F}$ that is separable.

For polynomials $f_1, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$, we deal with the extension $\mathbb{F}(x_1, \ldots, x_n)/\mathbb{F}(f_1, \ldots, f_m)$. This extension is algebraic iff $\operatorname{trdeg}(\mathbf{f}) = n$ (as every x_j depends on $\mathbf{f}$). In which case, the extension $\mathbb{F}(\mathbf{x})/\mathbb{F}(\mathbf{f})$ is separable iff the minimal polynomial of x_j over $\mathbb{F}(\mathbf{f})$ is separable, for all $j \in [n]$. The latter, clearly, is the case when $\operatorname{char}(\mathbb{F}) = 0$. When $\operatorname{char}(\mathbb{F}) = p > 0$, the extension is inseparable if there exists $j \in [n]$, such that the minimal polynomial of x_j over $\mathbb{F}(\mathbf{f})$ lives in $\mathbb{F}(\mathbf{f})[y^p]$. Thus for every x_j, we have an m_j such that $x_j^{p^{m_j}}$ has a separable minimal polynomial over $\mathbb{F}(\mathbf{f})$.

The *inseparable degree of the extension* $\mathbb{F}(\mathbf{x})/\mathbb{F}(\mathbf{f})$ is defined as the minimum p^m such that the minimal polynomial of $x_j^{p^m}$ over $\mathbb{F}(\mathbf{f})$ is separable, for all $j \in [n]$. We also associate this inseparable degree with the set $\mathbf{f}$.

In the case when $\mathbf{f}$ are algebraically dependent, we would like to use a "good" transcendence basis. This is captured by:

▶ **Definition 7** (Separating transcendence basis)**.** A field extension $\mathbb{E}/\mathbb{F}$ is called *separably generated* if there exists an algebraically independent set (i.e. transcendence basis) $S = \{f_1, \ldots, f_r\} \subset \mathbb{E}$ such that $\mathbb{E}/\mathbb{F}(S)$ is algebraic and separable.

S is called a *separating transcendence basis* of $\mathbb{E}/\mathbb{F}$.

It is a classical result that such bases exist for fields that we are interested in.

▶ **Theorem 8.** *Consider a finite set of polynomials* $\mathbf{f} \subset \mathbb{F}[\mathbf{x}]$. *If* $\mathbb{F}$ *is a finite field (resp. an algebraically closed field) then there exists a separating transcendence basis, of* $\mathbb{F}(\mathbf{f})/\mathbb{F}$*, in* $\mathbf{f}$.

In case $\mathbb{F}$ *is a zero characteristic field then* every *transcendence basis of* $\mathbf{f}$ *is a separating one of the extension* $\mathbb{F}(\mathbf{f})/\mathbb{F}$.

Proof. It is clear that if $\mathbb{F}$ has characteristic zero then there is no possibility of inseparability.

Let $\mathbb{F}$ be a finite (resp. algebraically closed) field. [24, Thm.7.20] shows that the extension $\mathbb{F}(\mathbf{f})/\mathbb{F}$ is separably generated. Furthermore, [24, Thm.7.18] shows that $\mathbf{f}$ contains a subset that is a separating transcendence basis of the extension. ◀

Examples. Extension $\mathbb{F}_3(x^3)/\mathbb{F}_3$ has $\{x^3\}$ as a separating transcendence basis. Consider the two transcendence bases of the extension $\mathbb{F}_3(x^2, x^3)/\mathbb{F}_3$ – $\{x^3\}$ and $\{x^2\}$. The latter is a separating transcendence basis, but the former is not.

2.2 Taylor expansion at $\mathbf{z}$, higher derivatives & differentials

We consider the application of shift (or translation) to our polynomials. We view this as writing the Taylor expansion of a polynomial $f(\mathbf{x})$ at a "generic" point $\mathbf{z}$ [13, Sec.C.1]. A second view is that of computing the Hasse-Schmidt higher derivatives of f at the point $\mathbf{z}$ [14, 10]. A third view is seeing the shifted polynomial as a Hasse-Schmidt differential [39]. We collect these equivalent viewpoints in a single definition.

▶ **Definition 9** (Formal shift)**.** We see $f(\mathbf{x}+\mathbf{z})$ as a polynomial in $R := \mathbb{F}_p(\mathbf{z})[\mathbf{x}]$ where the variables $x_1, \dots, x_n$ are *shifted* respectively by the function field elements $z_1, \dots, z_n$.

Now the coefficient of $m := x_1^{\ell_1} \cdots x_n^{\ell_n}$ in the Taylor-series expansion of $f(\mathbf{x}+\mathbf{z})$ can be written as $\frac{1}{\ell_1! \cdots \ell_n!} \frac{\partial^{(\ell_1+\cdots+\ell_n)} f}{\partial x_1^{\ell_1} \cdots \partial x_n^{\ell_n}}(\mathbf{z})$.

This is called the *Hasse-Schmidt derivative of f wrt m evaluated at the point* $\mathbf{z}$. It can be denoted, by some abuse of notation, as $\partial_m f(\mathbf{x})|_{\mathbf{z}}$.

Finally, we can see the formal shift as a *Hasse-Schmidt differential*, namely, $f(\mathbf{x}+\mathbf{z}) = \sum_m m \cdot \partial_m f(\mathbf{x})|_{\mathbf{z}}$ (sum over all monomials m in $\mathbf{x}$).

Example. We have $\partial^2 x^2/\partial x^2 = 0$ over $\mathbb{F}_2$, but $\partial^2 x^2/2!\partial x^2 = 1$. Thus, Hasse-Schmidt derivatives offer a natural solution to this vanishing problem.

This connection between the shifts and Hasse-Schmidt higher derivatives/ differentials is what motivated us to search for the right framework to study algebraic independence.

Now the Jacobian criterion is given in terms of the first order derivatives of the polynomials and the failure of Jacobian essentially exposes the inability of first order derivative in capturing independence. Intuitively, it seems that going to higher derivatives may help. The above connection points out that perhaps we need to look at higher degree terms (wrt $\mathbf{x}$) of $f(\mathbf{x}+\mathbf{z})$ to get an algebraic independence criterion in cases where Jacobian fails. Eventually, we will see that the intuition is indeed true.

Operator $\mathcal{H}$. For notational convenience, we define the non-constant part of $f(\mathbf{x}+\mathbf{z})$ up to degree$\le t$ wrt $\mathbf{x}$, as $\mathcal{H}_t f := f^{\le t}(\mathbf{x}+\mathbf{z}) - f(\mathbf{z})$.

This is easier to work with when we do manipulations modulo the ideal $\langle \mathbf{x} \rangle_R^{t+1}$.

3 Main structure theorems

We use the following standard notation in the paper:

1. $\mathbb{F}$ is an arbitrary field. $\overline{\mathbb{F}}$ is its algebraic closure.
2. $\mathbb{F}_q$ is a finite field of size q and characteristic $p \ge 2$.
3. Let $R \supseteq S$ be a commutative ring extension over a field $\mathbb{F}$, let $v_1, \dots, v_m \in R$ and $r \ge 1$. Then $\langle v_1, \dots, v_m \rangle_S^r$ is simply the set of all S-linear combinations of products $v_{i_1} \cdots v_{i_r}$ (i_j's in $[m]$). It is both an S-module and an $\mathbb{F}$-vector space. (It is an ideal when $R = S$.)
4. For a polynomial $h \in \mathbb{F}[\mathbf{x}]$, $h^{\le d}$ extracts out the degree$\le d$ part of h and returns it as an element in $\mathbb{F}[\mathbf{x}]$ again.
5. For a polynomial $h \in \mathbb{F}[\mathbf{x}]$, $h^{[\le d]}$ extracts out the degree$\le d$ part of h and returns it as a $d+1$ tuple, where for $i \in [0 \dots d]$, i-th entry of the tuple contains $h^{=i}$ which is defined as the homogeneous component of h of degree i.

3.1 Functional dependence for algebraically dependent polynomials

A fact about linear independence is that if $f_1, \ldots, f_m \in \mathbb{F}[\mathbf{x}]$ are linearly dependent, it also implies that every polynomial can be written as a linear combination of the polynomials in the basis. The question is whether the same can be extended to algebraic dependence: Does algebraic dependence imply that all the polynomials can be written as a function of the polynomials in the transcendence basis? It was shown in [27, Lem.3.1] that it is indeed true (approximately) over fields of zero (and large) characteristic.

We generalize the property using a different proof approach and show that algebraic dependence implies functional dependence over arbitrary fields (to arbitrary degree of approximation t).

▶ **Theorem 10** (Functional dependence over arbitrary fields). *Let $\mathbf{f} = \{f_1, \ldots, f_m\} \subset \mathbb{F}[x_1, \ldots, x_n]$ be a set of polynomials, where $\mathbb{F}$ is any field, and $t \in \mathbb{N}$. If trdeg of $\{f_1, \ldots, f_m\}$ is k, then there exist algebraically independent $\{g_1, \ldots, g_k\} \subset \mathbf{f}$, such that for random $\mathbf{a} \in \overline{\mathbb{F}}^n$, there are polynomials $h_i \in \overline{\mathbb{F}}[Y_1, \ldots, Y_k]$ satisfying, $\forall i \in [m]$, $f_i^{\leq t}(\mathbf{x}+\mathbf{a}) = h_i^{\leq t}(g_1(\mathbf{x}+\mathbf{a}), \ldots, g_k(\mathbf{x}+\mathbf{a}))$.*

▶ Remark. Clearly, $\overline{\mathbb{F}}^n$ is an infinite space. What we mean here by a *random* $\mathbf{a}$ is "random point in any sufficiently large, but finite, subset of the space". It will be clear from the proof that it would suffice to sample from any set of size at most exponential in the input size. We skip the detailed estimate as in this paper merely existence of $\mathbf{a}$ is needed.

We will use $\mathbf{z}$ as a formal variable (n-tuple) and can fix it later to a suitable constant $\mathbf{a}$. To prove the theorem, we consider the ring $R := \overline{\mathbb{F}}(\mathbf{z})[\mathbf{x}]$ and its ideal $\mathcal{I}_0 := \langle \mathbf{x} \rangle_R$. The ideal collects the non-constant linear polynomials. Now, define the ideal $\mathcal{I}_t := \mathcal{I}_0^{t+1}$ and the quotient algebra $\mathcal{Q}_t := R/\mathcal{I}_t$, i.e. we are filtering out, or *truncating*, all the terms of degree $> t$. Now $\mathcal{Q}_t$ can also be seen as a finite $\binom{n+t}{n}$ dimensional vector space over $\overline{\mathbb{F}}(\mathbf{z})$ whose basis is monomials in $\mathbf{x}$ of degree at most t. In our theorems and proofs, most of the operations happen in this quotient ring $\mathcal{Q}_t$ for increasing t's.

In our analysis, we plan to use the shifting of the variables in the evaluated annihilating polynomial of $\{f_i, g_1, \ldots, g_k\}$, and it is clear that on applying the shifts, we will end up having terms of the form $(\mathcal{H}_t f_i)^{j_0} (\mathcal{H}_t g_1)^{j_1} \cdots (\mathcal{H}_t g_k)^{j_k}$ (recall that in Q_t, $f(\mathbf{x}+\mathbf{z}) = f(\mathbf{z}) + \mathcal{H}_t f(\mathbf{x})$). Now, note that due to the filtration in $\mathcal{Q}_t$, some of these terms will be equivalent to terms involving $\mathcal{H}_r$ with $r < t$. We consider an appropriate subspace $\mathcal{U}_t \subset \mathcal{Q}_t$ generated by such "higher" products, which we formally define as: $\mathcal{U}_1 := \{0\}$ and

$$\mathcal{U}_t := \langle \mathcal{H}_{t-1} f_i, \mathcal{H}_{t-1} g_1, \ldots, \mathcal{H}_{t-1} g_k \rangle^2_{\overline{\mathbb{F}}(\mathbf{z})} + \cdots + \langle \mathcal{H}_1 f_i, \mathcal{H}_1 g_1, \ldots, \mathcal{H}_1 g_k \rangle^t_{\overline{\mathbb{F}}(\mathbf{z})}, \quad t \geq 2.$$

▶ Remark 11. In $\mathcal{Q}_t$, observe that, this is the same subspace as $\langle \mathcal{H}_t f_i, \mathcal{H}_t g_1, \ldots, \mathcal{H}_t g_k \rangle^2_{\overline{\mathbb{F}}(\mathbf{z})} + \cdots + \langle \mathcal{H}_t f_i,\, \mathcal{H}_t g_1, \ldots, \mathcal{H}_t g_k \rangle^t_{\overline{\mathbb{F}}(\mathbf{z})}$

Proof of Theorem 10. Consider the set $\mathbf{f} := \{f_1, \ldots, f_m\} \subset \mathbb{F}[\mathbf{x}]$ with algebraic rank k. If we work over $\overline{\mathbb{F}}$, then Thm.8 guarantees the existence of a separating transcendence basis $\{g_1, \ldots, g_k\} \subseteq \mathbf{f}$. Let $g_0 := f_i$ for a fixed $i \in [m]$. Now we consider the separable annihilating polynomial $A(\mathbf{y}) = \sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell} \mathbf{y}^{\mathbf{e}_\ell}$ of the set $\mathbf{g} := \{g_0, g_1, \ldots, g_k\}$, and $a_{\mathbf{e}_\ell}$'s are in $\overline{\mathbb{F}}$ ($\mathbf{e}_\ell$ is a $(k+1)$-tuple $(e_{j\ell} \mid j \in [0 \ldots k])$). Thus, $A(\mathbf{g}) = \sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell} \prod_{j=0}^{k} g_j(\mathbf{x})^{e_{j\ell}} = 0$. We now apply the formal shift $\mathbf{x} \mapsto \mathbf{x} + \mathbf{z}$ to get $A(g_0(\mathbf{x}+\mathbf{z}), \ldots, g_k(\mathbf{x}+\mathbf{z})) = 0$, i.e. $\sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell} \prod_j g_j(\mathbf{x}+\mathbf{z})^{e_{j\ell}} = 0$.

We now study this relation in the algebra $\mathcal{Q}_t$. By Taylor series expansion, we know that $f(\mathbf{x}+\mathbf{z}) \equiv f(\mathbf{z}) + \mathcal{H}_t f(\mathbf{x})$ in $\mathcal{Q}_t$, so we get $\sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell} \prod_j (g_j(\mathbf{z}) + \mathcal{H}_t g_j)^{e_{j\ell}} \equiv 0$. The binomial expansion gives a compact expression:

$$\sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell} \sum_{\mathbf{0}\le\mathbf{s}\le\mathbf{e}_\ell} \binom{\mathbf{e}_\ell}{\mathbf{s}} \cdot (\mathcal{H}_t\mathbf{g})^{\mathbf{s}} \cdot \mathbf{g}^{\mathbf{e}_\ell-\mathbf{s}} \equiv 0\,.$$

Note that the contribution by $\mathbf{s}=\mathbf{0}$ terms sum up to $\sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell} \prod_{j=0}^{k} g_j(\mathbf{z})^{e_{j\ell}}$ which is zero. This implies that an $\overline{\mathbb{F}}(\mathbf{z})$-linear combination of the products of the form $(\mathcal{H}_t g_0)^{s_0}\cdots(\mathcal{H}_t g_k)^{s_k}$, $\sum_j s_j \ge 1$, vanishes in $\mathcal{Q}_t$. Now the key step is to separate out the terms *linear* in $\mathcal{H}_t g_j$ and switch the sums, to obtain

$$\mathcal{H}_t g_0 \cdot g_0(\mathbf{z})^{-1}\left(\sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell}\cdot e_{0\ell} g_0^{e_{0\ell}}\cdots g_k^{e_{k\ell}}\right) + \sum_{j\in[k]} \mathcal{H}_t g_j \cdot g_j(\mathbf{z})^{-1}\left(\sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell}\cdot e_{j\ell} g_0^{e_{0\ell}}\cdots g_k^{e_{k\ell}}\right)$$
$$+\,(\text{higher terms with } \sum_j s_j \ge 2) \equiv 0\,. \qquad (1)$$

Further, we argue using the minimality and separability of A (in terms of the first variable) that the "linear" term $\mathcal{H}_t g_0$ in the vanishing sum above has a non-zero coefficient: as it would either mean a lower degree annihilating polynomial $A := \sum_{\mathbf{e}_\ell} a_{\mathbf{e}_\ell} e_{0\ell} y_0^{e_{0\ell}-1}\cdot y_1^{e_{1\ell}}\cdots y_k^{e_{k\ell}}$ i.e. contradicting the minimality, or that all the $e_{0\ell}$'s are divisible by p (when $\mathbb{F}$ has characteristic p) which means that f_i does not depend separably on $\{g_1,\ldots,g_k\}$; which contradicts the fact that $\{g_1,\ldots,g_k\}$ is a separating transcendence basis.

Thus, we get that $\mathcal{H}_t g_0$ lives in the $\overline{\mathbb{F}}(\mathbf{z})$-linear span of $\mathcal{H}_t g_1,\ldots,\mathcal{H}_t g_k$ modulo the subspace generated by the higher terms of the summation in Eqn.1. So, $\mathcal{H}_t g_0$ lives in the $\overline{\mathbb{F}}(\mathbf{z})$-linear span of $\mathcal{H}_t g_1\ldots,\mathcal{H}_t g_k$ modulo the subspace $\mathcal{U}_t$ (Rmk.11) in $\mathcal{Q}_t$.

We got $\mathcal{H}_t f_i \in \langle \mathcal{H}_t g_1,\ldots,\mathcal{H}_t g_k\rangle_{\overline{\mathbb{F}}(\mathbf{z})} + \mathcal{U}_t$. Now, we are in a position to apply Lemma 12, which essentially says that if $\mathcal{H}_r f_n$ depends on higher order terms (in the sense of Equation 1) then it can be "dropped" from the ideal manipulations. Thus, we get that $\mathcal{H}_t f_i \in \langle \mathcal{H}_t g_1,\ldots,\mathcal{H}_t g_k\rangle_{\overline{\mathbb{F}}(\mathbf{z})} + \langle \mathcal{H}_{t-1} g_1,\ldots,\mathcal{H}_{t-1} g_k\rangle^2_{\overline{\mathbb{F}}(\mathbf{z})} + \cdots + \langle \mathcal{H}_1 g_1,\ldots,\mathcal{H}_1 g_k\rangle^t_{\overline{\mathbb{F}}(\mathbf{z})}$. The latter (by Rmk.11) is exactly $\langle \mathcal{H}_t g_1,\ldots,\mathcal{H}_t g_k\rangle_{\overline{\mathbb{F}}(\mathbf{z})} + \langle \mathcal{H}_t g_1,\ldots,\mathcal{H}_t g_k\rangle^2_{\overline{\mathbb{F}}(\mathbf{z})} + \cdots + \langle \mathcal{H}_t g_1,\ldots,\mathcal{H}_t g_k\rangle^t_{\overline{\mathbb{F}}(\mathbf{z})}$.

This implies $f_i(\mathbf{x}+\mathbf{z}) \in \langle 1, g_1(\mathbf{x}+\mathbf{z}),\ldots,g_k(\mathbf{x}+\mathbf{z})\rangle^t_{\overline{\mathbb{F}}(\mathbf{z})}$ in $\mathcal{Q}_t$, which yields the approximate functional dependence around a generic point $\mathbf{z}$.

Fixing $\mathbf{z}$ (avoiding some bad choices that make certain $\mathbf{z}$-polynomials in the above proof zero) to an element $\mathbf{a}\in\overline{\mathbb{F}}^n$ finishes the proof. ◀

We now formally state our subspace reduction lemma:

▶ **Lemma 12** (Subspace reduction). *Let $\mathbb{F}$ be any field, $R := \mathbb{F}(\mathbf{z})[\mathbf{x}]$, $\mathcal{Q}_r := R/\langle\mathbf{x}\rangle^{r+1}$ for $r\ge 1$, and $\mathbf{f}\subset\mathbb{F}[\mathbf{x}]$. Define $\mathcal{U}_1=\mathcal{V}_1=\{0\}$, and for $u\in\langle\mathbf{x}\rangle_R$, $r\ge 2$, define the subspaces (in the quotient algebra $\mathcal{Q}_r$),*

$$\mathcal{U}_r := \langle \mathcal{H}_{r-1}f_1,\ldots,\mathcal{H}_{r-1}f_n\rangle^2_{\mathbb{F}(\mathbf{z})} + \cdots + \langle \mathcal{H}_1 f_1,\ldots,\mathcal{H}_1 f_n\rangle^r_{\mathbb{F}(\mathbf{z})}\,,$$
$$\mathcal{V}_r := \langle \mathcal{H}_{r-1}f_1,\ldots,\mathcal{H}_{r-1}f_{n-1},u\rangle^2_{\mathbb{F}(\mathbf{z})} + \cdots + \langle \mathcal{H}_1 f_1,\ldots,\mathcal{H}_1 f_{n-1},u\rangle^r_{\mathbb{F}(\mathbf{z})}\,.$$

If $\mathcal{H}_t f_n \in \langle \mathcal{H}_t f_1,\ldots,\mathcal{H}_t f_{n-1},u\rangle_{\mathbb{F}(\mathbf{z})} + \mathcal{U}_t$, then $\mathcal{U}_t\subseteq\mathcal{V}_t$ (for any $t\in\mathbb{N}$).

▶ Remark. If $u=0$ then the lemma "reduces" the n polynomial generators, of the subspace $\mathcal{U}_t$, by one. Hence, the name "subspace reduction". A simple inductive proof of the lemma is given in the full version.

3.2 Algebraically independent polynomials: Criterion

Having proved the functional dependence for algebraically dependent polynomials, one naturally asks whether a converse exists (for arbitrary fields? to what degree?). We will characterize this completely.

It's all about the inseparable degree- We show that if f is algebraically independent of $\{g_1, \ldots, g_k\}$ then, under a random shift, f cannot be written as a function of $\{g_1, \ldots, g_k\}$ when chosen to truncate at (or beyond) the inseparable degree of the extension $\mathbb{F}_q(\mathbf{x})/\mathbb{F}_q(f, g_1, \ldots, g_k)$. Moreover, for each truncation at lower degrees we get functional dependence.

▶ **Theorem 13** (Algebraic to functional independence). *Let $\mathbf{f} \subset \mathbb{F}_q[\mathbf{x}]$ be algebraically independent polynomials (wlog n-variate n polynomials) with inseparable degree p^i. Then,*

1. *for all $t \geq p^i$, for random $\mathbf{a} \in \overline{\mathbb{F}}_q^n$, $f_n^{\leq t}(\mathbf{x}+\mathbf{a})$ cannot be written as $h^{\leq t}(f_1(\mathbf{x}+\mathbf{a}), \ldots, f_{n-1}(\mathbf{x}+\mathbf{a}))$, for any $h \in \overline{\mathbb{F}}_q[Y_1, \ldots, Y_{n-1}]$.*
2. *for all $1 \leq t < p^i$, $\exists j \in [n]$, for random $\mathbf{a} \in \overline{\mathbb{F}}_q^n$, $f_j^{\leq t}(\mathbf{x}+\mathbf{a})$ can be written as $h_{jt}^{\leq t}(f_1(\mathbf{x}+\mathbf{a}), \ldots, f_{j-1}(\mathbf{x}+\mathbf{a}), f_{j+1}(\mathbf{x}+\mathbf{a}), \ldots, f_n(\mathbf{x}+\mathbf{a}))$, for some $h_{jt} \in \overline{\mathbb{F}}_q[\mathbf{Y}]$.*

▶ Remark. Our proof works for any field $\mathbb{F}$ (manipulate in $\overline{\mathbb{F}}$). In case of characteristic $p \geq 2$ we get the above statement and in characteristic zero use inseparable degree $= 1$.

Proof idea: By the hypothesis we have that each monomial $x_j^{p^i}$, $j \in [n]$, algebraically depends on $\mathbf{f}$ with a *separable* annihilating polynomial over $\mathbf{F}_q$. Consider ring $R := \overline{\mathbb{F}}_q(\mathbf{z})[\mathbf{x}]$. The basic idea is to consider the minimal annihilating polynomial A_j of $\{x_j^{p^i}, \mathbf{f}\}$ and formally shift the relevant polynomials by $\mathbf{z}$. From the proof of Thm.10 we get a functional dependence of $x_j^{p^i}$ on $\mathbf{f}(\mathbf{x}+\mathbf{z})$ up to any degree t.

Interestingly, when we take $t < p^i$ the monomial $x_j^{p^i}$ vanishes mod $\langle \mathbf{x} \rangle^{t+1}$. This means that the above yields, in fact, a functional dependence among $\mathbf{f}(\mathbf{x}+\mathbf{z})$.

On the other hand, for $t \geq p^i$, we get a nontrivial functional dependence of $x_j^{p^i}$ on $\mathbf{f}(\mathbf{x}+\mathbf{z})$, for all $j \in [n]$. In this case, one can give an argument using monomial ordering that there exists no functional dependence among $\mathbf{f}(\mathbf{x}+\mathbf{z})$.

We can see the classical Jacobian criterion as a special case of Theorems 10 and 13. The detailed discussions and missing proofs are given in the full version.

4 Conclusion

We give a criterion for testing algebraic independence over positive characteristic, in the spirit of Jacobian criterion, that works for any field. Its complexity is parameterized by the inseparable degree bound. It is also strong enough to give the inseparable degree at the same time. We give applications to locally low algebraic rank circuits in the cases that were open before.

	Jacobian Criterion	**Our Criterion**
The approach:	reduces algebraic independence to linear independence testing	reduces algebraic independence to linear independence testing
Related "approximate" shift :	$\mathbf{f}(\mathbf{x}) \mapsto \mathbf{f}(\mathbf{x}+\mathbf{z}) \mod \langle \mathbf{x} \rangle^2_{\mathbb{F}(\mathbf{z})[\mathbf{x}]}$	$\mathbf{f}(\mathbf{x}) \mapsto \mathbf{f}(\mathbf{x}+\mathbf{z}) \mod \mathcal{U}_t$
Vectors for $\mathbb{F}(\mathbf{z})$-dependence:	$\mathcal{H}_1\mathbf{f} \mod \mathcal{U}_1$	$\mathcal{H}_t\mathbf{f} \mod \mathcal{U}_t$
Certifies alg.independence if:	$\mathbb{F}(\mathbf{x})/\mathbb{F}(\mathbf{f})$ is separable	separable or inseparable $\mathbb{F}(\mathbf{x})/\mathbb{F}(\mathbf{f})$
Efficiency in char$(\mathbb{F}) = 0$:	randomized poly-time algorithm	$t = 1$, (same as Jacobian criterion)
Efficiency in char$(\mathbb{F}) = p$, inseparable degree $\leq p^e$:	fails	randomized poly$\binom{n+p^e}{n}$-time algorithm

The main open problem is to investigate whether we can improve the criterion to get a randomized poly-time algorithm for circuits over a finite field. We mention a few special cases based on different restrictions on input. None of these cases are (efficiently) solved by presently known techniques.

- the polynomials are *supersparse*, i.e. sparse polynomials with possibly exponential degree.
- two bivariate circuits, with an exponentially large inseparable degree, over $\mathbb{F}_2$.
- n quadratic polynomials over $\mathbb{F}_2$.

Our hitting-set result, for locally low algebraic rank circuits, still has a mild assumption on the characteristic. Can this be eliminated?

Acknowledgements. We thank Manindra Agrawal, Rohit Gurjar & Arpita Korwar for the insightful discussions and encouragement. We thank Markus Bläser and the anonymous reviewers for the elaborate suggestions to improve the draft. NS acknowledges the support from DST/SJF/MSA-01/2013-14 and SB/FTP/ETA-177/2013.

References

1 M. Agrawal, C. Saha, R. Saptharishi, and N. Saxena. Jacobian hits circuits: Hitting-sets, lower bounds for depth-D occur-k formulas & depth-3 transcendence degree-k circuits. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 599–614, 2012. (In SICOMP special issue).

2 Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 67–75, 2008.

3 W. Bauer and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22(3):317–330, 1983.

4 M. Beecken, J. Mittmann, and N. Saxena. Algebraic Independence and Blackbox Identity Testing. *Inf. Comput.*, 222:2–19, 2013. (Conference version in ICALP 2011).

5 Radu Curticapean. Counting matchings of size k is #W[1]-hard. In *Automata, Languages, and Programming*, pages 352–363. Springer, 2013.

6 Richard A DeMillo and Richard J Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, 1978.

7 Z. Dvir, A. Gabizon, and A. Wigderson. Extractors and rank extractors for polynomial sources. *Comput. Complex.*, 18(1):1–58, 2009. (Conference version in FOCS 2007).

8 Z. Dvir, D. Gutfreund, G.N. Rothblum, and S.P. Vadhan. On approximating the entropy of polynomial mappings. In *Innovations in Computer Science (ICS)*, pages 460–475, 2011.

9 Zeev Dvir. Extractors for varieties. In *Proceedings of the 24th IEEE Conference on Computational Complexity (CCC)*, pages 102–113, 2009.

10 Zeev Dvir, Swastik Kopparty, Shubhangi Saraf, and Madhu Sudan. Extensions to the method of multiplicities, with applications to kakeya sets and mergers. *SIAM Journal on Computing*, 42(6):2305–2328, 2013. (Preliminary version in FOCS'09).

11 Zeev Dvir, Amir Shpilka, and Amir Yehudayoff. Hardness-randomness tradeoffs for bounded depth arithmetic circuits. *SIAM Journal on Computing*, 39(4):1279–1293, 2009.

12 Richard Ehrenborg and Gian-Carlo Rota. Apolarity and canonical forms for homogeneous polynomials. *European Journal of Combinatorics*, 14(3):157–181, 1993.

13 Michael A Forbes. *Polynomial identity testing of read-once oblivious algebraic branching programs.* PhD thesis, Massachusetts Institute of Technology, 2014.

14 Michael A Forbes. Deterministic divisibility testing via shifted partial derivatives. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 451–465. IEEE, 2015.

15 Krister Forsman. Two themes in commutative algebra: Algebraic dependence and kähler differentials, 1992.

16 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth three. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 578–587, 2013.

17 Helmut Hasse and Friedrich K. Schmidt. Noch eine begründung der theorie der höheren differentialquotienten in einem algebraischen funktionenkörper einer unbestimmten. (nach einer brieflichen mitteilung von f.k.schmidt in jena). *Journal für die reine und angewandte Mathematik*, 177:215–223, 1937.

18 L. Illusie. Crystalline cohomology. In *Proc. Sympos. Pure Math.*, volume 55, pages 43–70, 1994. Motives (Seattle, WA, 1991).

19 I Martin Isaacs. *Algebra: a graduate course*, volume 100. American Mathematical Soc., 1994.

20 C. G. J. Jacobi. De determinantibus functionalibus. *J. Reine Angew. Math.*, 22(4):319–359, 1841.

21 K. A. Kalorkoti. A Lower Bound for the Formula Size of Rational Functions. *SIAM J. Comp.*, 14(3):678–687, 1985. (Conference version in ICALP 1982).

22 N. Kayal. The Complexity of the Annihilating Polynomial. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 184–193, 2009.

23 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. In *Foundations of Computer Science (FOCS), IEEE 55th Annual Symposium on*, pages 61–70. IEEE, 2014.

24 Anthony W Knapp. *Advanced algebra.* Springer Science & Business Media, 2007.

25 Steven G Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications.* Springer Science & Business Media, 2012.

26 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *Foundations of Computer Science (FOCS), IEEE 55th Annual Symposium on*, pages 364–373. IEEE, 2014.

27 Mrinal Kumar and Shubhangi Saraf. Arithmetic circuits with locally low algebraic rank. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:194, 2015. (To appear in CCC 2016). URL: `http://eccc.hpi-web.de/report/2015/194`.

28 M.S. L'vov. Calculation of invariants of programs interpreted over an integrality domain. *Cybernetics and Systems Analysis*, 20:492–499, 1984.

29 Johannes Mittmann, Nitin Saxena, and Peter Scheiblechner. Algebraic independence in positive characteristic: A p-adic calculus. *Transactions of the American Mathematical Society*, 366(7):3425–3450, 2014.

30 James G Oxley. *Matroid theory*, volume 3. Oxford university press, 2006.

31 O. Perron. *Algebra I (Die Grundlagen).* W. de Gruyter, Berlin, 1927.

32 A. Płoski. Algebraic Dependence of Polynomials After O. Perron and Some Applications. In *Computational Commutative and Non-Commutative Algebraic Geometry*, pages 167–173. 2005.

33 Ran Raz. Elusive functions and lower bounds for arithmetic circuits. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 711–720. ACM, 2008.

34 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity, 2016. https://github.com/dasarpmar/lowerbounds-survey/.

35 Nitin Saxena. Progress on polynomial identity testing-ii. In *Perspectives in Computational Complexity*, pages 131–146. Springer, 2014.

36 J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

37 A. Shpilka and A. Yehudayoff. Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

38 Oswald Teichmüller. Differentialrechnung bei charakteristik p. *Journal für die reine und angewandte Mathematik*, 175:89–99, 1936.

39 William Nathaniel Traves. Differential operators and Nakai's conjecture, 1998.

40 Richard Zippel. *Probabilistic algorithms for sparse polynomials.* Springer, 1979.

Parameterized Algorithms on Perfect Graphs for Deletion to (r, ℓ)-Graphs[*]

Sudeshna Kolay[1], Fahad Panolan[2], Venkatesh Raman[3], and Saket Saurabh[4]

1 **Institute of Mathematical Sciences, Chennai, India**
2 **Department of Informatics, University of Bergen, Norway**
3 **Institute of Mathematical Sciences, Chennai, India**
4 **Institute of Mathematical Sciences, Chennai, India; and Department of Informatics, University of Bergen, Norway**

Abstract

For fixed integers $r, \ell \geq 0$, a graph G is called an (r, ℓ)*-graph* if the vertex set $V(G)$ can be partitioned into r independent sets and ℓ cliques. Such a graph is also said to have *cochromatic number* $r + \ell$. The class of (r, ℓ) graphs generalizes r-colourable graphs (when $\ell = 0$) and hence not surprisingly, determining whether a given graph is an (r, ℓ)-graph is NP-hard even when $r \geq 3$ or $\ell \geq 3$ in general graphs.

When r and ℓ are part of the input, then the recognition problem is NP-hard even if the input graph is a perfect graph (where the Chromatic Number problem is solvable in polynomial time). It is also known to be fixed-parameter tractable (FPT) on perfect graphs when parameterized by r and ℓ. I.e. there is an $f(r + \ell) \cdot n^{\mathcal{O}(1)}$ algorithm on perfect graphs on n vertices where f is a function of r and ℓ. Observe that such an algorithm is unlikely on general graphs as the problem is NP-hard even for constant r and ℓ.

In this paper, we consider the parameterized complexity of the following problem, which we call Vertex Partization. Given a perfect graph G and positive integers r, ℓ, k decide whether there exists a set $S \subseteq V(G)$ of size at most k such that the deletion of S from G results in an (r, ℓ)-graph. This problem generalizes well studied problems such as Vertex Cover (when $r = 1$ and $\ell = 0$), Odd Cycle Transversal (when $r = 2$, $\ell = 0$) and Split Vertex Deletion (when $r = 1 = \ell$).

1. Vertex Partization on perfect graphs is FPT when parameterized by $k + r + \ell$.
2. The problem, when parameterized by $k + r + \ell$, does not admit any polynomial sized kernel, under standard complexity theoretic assumptions. In other words, in polynomial time, the input graph cannot be compressed to an equivalent instance of size polynomial in $k + r + \ell$. In fact, our result holds even when $k = 0$.
3. When r, ℓ are universal constants, then Vertex Partization on perfect graphs, parameterized by k, has a polynomial sized kernel.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Graph deletion, FPT algorithms, Polynomial kernels

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.75

[*] The research leading to these results has received funding from the European Research Council (ERC) via grants Rigorous Theory of Preprocessing, reference 267959 and PARAPPROX, reference 306992.

1 Introduction

For fixed integers $r, \ell \geq 0$, a graph G is called an (r, ℓ)-*graph* if the vertex set $V(G)$ can be partitioned into r independent sets and ℓ cliques. Many special subclasses of this graph class, such as bipartite, chordal, interval, split and permutation, are well studied in various areas of algorithm design and intractability. For example, $(2, 0)$- and $(1, 1)$-graphs correspond to bipartite graphs and split graphs respectively. A $(3, 0)$-graph is a 3-colourable graph. Hence, there is a rich dichotomy even with respect to recognition algorithms for (r, ℓ)-graphs. It is well known that we can recognize $(2, 0)$- and $(1, 1)$-graphs, on n vertices and m edges, in $\mathcal{O}(m + n)$ time. In fact, one can show that recognizing whether a graph G is an (r, ℓ)-graph, when $r, \ell \leq 2$, can be done in polynomial time [4, 9]. On the other hand, when either $r \geq 3$ or $\ell \geq 3$, the recognition problem is as hard as the celebrated 3-Colouring problem, which is NP-complete [12]. Thus, the following problem is NP-hard when r or ℓ are at least 3:

Partization Recognition
Input: A graph G and positive integers r, ℓ
Question: Is G an (r, ℓ)-graph?

Partization Recognition has also been studied when the input is restricted to be a chordal graph. This restricted problem has a polynomial time algorithm [10]. On the other hand, when the input graphs are restricted to perfect graphs, Partization Recognition is NP-complete [23]. It was shown in [15], that the problem, when parameterized by $r + \ell$, has an FPT algorithm, i.e, an algorithm that runs in time $f(r + \ell) \cdot n^{\mathcal{O}(1)}$ for a computable function f. A natural extension to Partization Recognition is the Vertex Partization problem. The problem is formally stated below:

Vertex Partization **Parameter:** r, ℓ, k
Input: A graph G and positive integers r, ℓ, k
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is an (r, ℓ)-graph?

Because of the NP-hardness of the 3-Colouring problem, we do not expect to obtain an FPT algorithm on general graphs, parameterized by $k + r + \ell$, for Vertex Partization, when $r \geq 3$ or $\ell \geq 3$. It has been shown in [16, 2] that, for all other combinations of r and ℓ, namely when $0 \leq r, \ell \leq 2$, Vertex Partization has an FPT algorithm with running time $3.3146^k n^{\mathcal{O}(1)}$. Various special cases of this problem are very well studied. When $r = 1$ and $\ell = 0$, the problem is the same as the celebrated Vertex Cover problem, which has been extensively studied in parameterized complexity, and the current fastest algorithm runs in time $1.2738^k n^{\mathcal{O}(1)}$ and has a kernel with $2k$ vertices [5] (A brief overview of paramaterized complexity is given in the preliminaries).

For $r = 2$ and $\ell = 0$, the problem is the same as Odd Cycle Transversal (OCT) whose parameterized complexity was settled by Reed et al. [22] by using the iterative compression technique (which is also the technique we use to show that Vertex Partization on perfect graphs has an FPT algorithm in this paper) for the first time. The current best algorithm for the problem is by Lokshtanov et al. [20] with a running time of $2.3146^k n^{\mathcal{O}(1)}$ that uses a branching algorithm based on linear programming.

When $r = \ell = 1$, the Vertex Partization problem is the same as Split Vertex Deletion (SVD) for which Ghosh et al. [13] designed an algorithm with running time $2^k n^{\mathcal{O}(1)}$. They also gave the best known polynomial kernel for SVD. Later, Cygan and Pilipczuk [7] designed an algorithm for SVD running in time $1.2738^{k+o(k)} n^{\mathcal{O}(1)}$.

As in the case of Partization Recognition, the Vertex Partization problem has also been studied when the input graph is restricted to a non-trivial graph class. By a celebrated result of Lewis and Yannakakis [19], this problem is NP-complete even when restricted to the perfect graph class. In fact, Odd Cycle Transversal (OCT) restricted to perfect graphs is NP-hard, because of this result. Thus, we cannot expect an FPT algorithm for Vertex Partization parameterized by $r + \ell$, unless P = NP. Moreover, because of the NP-hardness of Partization Recognition on perfect graphs, we do not expect Vertex Partization on perfect graphs to be FPT when parameterized by k alone, again under the assumption that P $\neq$ NP. Krithika and Narayanaswamy [17] studied Vertex Partization problems on perfect graphs, and among several results, they obtain an $(r+1)^k n^{\mathcal{O}(1)}$ algorithm for Vertex $(r, 0)$-Partization on perfect graphs. In this paper, we generalize this for all values of r and ℓ. In other words, we show that Vertex Partization on perfect graphs, parameterized by $k + r + \ell$, is FPT.

Our Results and Methods. For Vertex Partization on perfect graphs, parameterized by $k + r + \ell$, we give an FPT algorithm using the method of iterative compression. This algorithm is inspired by the FPT algorithm for Cochromatic Number on Perfect Graphs, given in [15].

Then, we obtain a negative result for *kernelization* for Vertex Partization on perfect graphs. We show that Vertex Partization cannot have a polynomial kernel unless NP $\subseteq$ co-NP/poly. This is shown by exhibiting a polynomial parameter transformation from CNF-SAT. In fact, our result holds even when $k = 0$ and either r or ℓ is one. Thus, we show that Partization Recognition, parameterized by r, ℓ (also known as the Cochromatic Number problem [15]), does not admit a polynomial kernel on perfect graphs unless NP $\subseteq$ co-NP/poly. See Section 2 for the definition of polynomial parameter transformation and kernelization.

Finally, we consider the following parameterized problem:

Vertex (r, ℓ) Partization **Parameter:** k
Input: A graph G and a positive integer k
Question: Is there a vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is an (r, ℓ)-graph?

For each pair of constants r and ℓ, we give a polynomial kernelization algorithm for the above parameterized problem. To arrive at the kernelization algorithm, we consider a slightly larger class of graphs called (r, ℓ)-*split graphs*. A graph G is an (r, ℓ)-split graph if its vertex set can be partitioned into V_1 and V_2, such that the size of a largest clique in $G[V_1]$ is bounded by r and the size of a largest independent set in $G[V_2]$ is bounded by ℓ. Such a two-partition for the graph G is called as (r, ℓ)-split partition. The notion of (r, ℓ)-split graphs was introduced in [14]. For any fixed r and ℓ, there is a finite forbidden set $\mathbb{F}_{r,\ell}$ of graphs for (r, ℓ)-split graphs [14]. That is, a graph G is an (r, ℓ)-split graph if and only if G does not contain any graph $H \in \mathbb{F}_{r,\ell}$ as an induced subgraph. The size of the largest forbidden graph is bounded by $f(r, \ell)$, for some function f depending only on r and ℓ [14]. We use this to design the kernelization algorithm.

2 Preliminaries

We use $[n]$ to denote $\{1, \ldots, n\}$. We use standard notations from graph theory [8]. The vertex set and edge set of a graph are denoted as $V(G)$ and $E(G)$ respectively. The complement of the graph G, denoted by $\overline{G}$, has $V(G)$ as its vertex set and $\binom{V(G)}{2} \setminus E(G)$ as its edge set. Here, $\binom{V(G)}{2}$ denotes the family of two sized subsets of $V(G)$. The neighbourhood of a vertex v is represented as $N_G(v)$, or, when the context of the graph is clear, simply as $N(v)$. An induced subgraph of G on the vertex set $V' \subseteq V$ is written as $G[V']$. For a vertex subset $V' \subseteq V$, $G[V \setminus V']$ is also denoted as $G - V'$. We denote by $\omega(G)$ the size of a maximum clique in G. Similarly, $\alpha(G)$ denotes the size of a maximum independent set in G. The chromatic number of G is denoted by $\chi(G)$. In this paper, we consider the class of (r, ℓ)-graphs, The following is the formal definition of this graph class.

▶ **Definition 1** ((r, ℓ)-graph)**.** A graph G is an (r, ℓ)-graph if its vertex set can be partitioned into r independent sets and ℓ cliques. We call such a partition of $V(G)$ an (r, ℓ)-partition.

For a graph G, we say $S \subseteq V(G)$ is an (r, ℓ)-*vertex deletion set*, if $G - S$ is an (r, ℓ)-graph.

▶ **Definition 2** (IC-partition)**.** An IC-partition, of an (r, ℓ)-graph G, is a partition (V_1, V_2) of $V(G)$ such that $G[V_1]$ can be partitioned into r independent sets and $G[V_2]$ can be partitioned into ℓ cliques.

A graph G is a *perfect graph* if, for every induced subgraph H, $\chi(H) = \omega(H)$. We also need the following characterisation of perfect graphs – also known as strong perfect graph theorem.

▶ **Proposition 3** ([6])**.** *A graph G is perfect if and only if G does not have, as an induced subgraph, an odd cycle of length at least* 5 *or its complement.*

This tells us that perfect graphs are closed under complementation. However, this was proved earlier and was called weak perfect graph theorem.

▶ **Lemma 4** ([21])**.** *G is a perfect graph if and only if $\overline{G}$ is a perfect graph.*

Moreover, this class is well known for its tractability for several NP-hard problems.

▶ **Proposition 5** ([15, Lemma 3])**.** *Given a perfect graph G and an integer ℓ, there is a polynomial time algorithm to output*

(a) *either a partition of $V(G)$ into at most ℓ independent sets or a clique of size $\ell + 1$, and*
(b) *either a partition of $V(G)$ into at most ℓ cliques or an independent set of size $\ell + 1$.*

Parameterized Complexity. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x, k), where x is a classical problem instance, and k is called the parameter. A central notion in parameterized complexity is *Fixed Parameter Tractability (FPT)*. A parameterized problem Π is in FPT if there is an algorithm that takes an instance (x, k) and decides if $(x, k) \in \Pi$ in time $f(k) \cdot |x|^{\mathcal{O}(1)}$. Here, f is an arbitrary function of k. Such an algorithm is called a *Fixed Parameter Tractable* algorithm and, in short, an FPT algorithm.

Kernelization. A kernelization algorithm for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that, given $(x, k) \in \Gamma^* \times \mathbb{N}$, outputs, in time polynomial in $|x| + k$, a pair $(x', k') \in \Gamma^* \times \mathbb{N}$ such that (a) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$ and (b) $|x'|, k' \leq g(k)$, where g is some computable function. The output instance x' is called the kernel, and the function g is referred to as the size of the kernel. If $g(k) = k^{\mathcal{O}(1)}$ then we say that Π has a polynomial kernel.

Lower bounds in Kernelization. In recent years, several techniques have been developed to show that certain parameterized problems cannot have any polynomial sized kernel unless some classical complexity assumptions are violated. One such technique is the *polynomial parameter transformation.*

▶ **Definition 6.** Let Π, Γ be two parameterized problems. A polynomial time algorithm $\mathcal{A}$ is called a polynomial parameter transformation (or ppt) from Π to Γ if , given an instance (x, k) of Π, $\mathcal{A}$ outputs in polynomial time an instance (x', k') of Γ such that $(x, k) \in \Pi$ if and only if $(x', k') \in \Gamma$ and $k' \leq k^{\mathcal{O}(1)}$.

We use the following theorem together with ppt reductions to rule out polynomial kernels.

▶ **Proposition 7** ([3]). *Let Π, Γ be two parameterized problems such that Π is* NP*-hard, $\Gamma \in$ NP and there exists a polynomial parameter transformation from Π to Γ. Then, if Π does not admit a polynomial kernel neither does Γ.*

▶ **Proposition 8** ([11]). CNF-SAT *is* FPT *parameterized by the number of variables; however, it does not admit a polynomial kernel unless* NP $\subseteq$ co-NP*/poly.*

3 FPT algorithm for Vertex Partization

In this section, we show that Vertex Partization on perfect graphs is FPT, using the iterative compression technique. Let (G, r, ℓ, k) be an input instance of Vertex Partization on perfect graphs, and let $V(G) = \{v_1, \ldots, v_n\}$. We define, for every $1 \leq i \leq n$, the vertex set $V_i = \{v_1, \ldots, v_i\}$. Let G_i denote $G[V_i]$. Let $i_0 = r + \ell + k + 1$. We iterate through the instances (G_i, r, ℓ, k) starting from $i = i_0$. Given the i^{th} instance and a known (r, ℓ)-vertex deletion set S'_i of size at most $k+1$, our objective is to obtain an (r, ℓ)-vertex deletion set S_i of size at most k. The formal definition of this compression problem is as follows.

Vertex Partization Compression **Parameter:** r, ℓ, k
Input: A perfect graph G, non-negative integers r, ℓ, k and a $k+1$-sized vertex subset S' such that $G - S'$ is an (r, ℓ)-graph, along with an IC-partition (Q_1, Q_2) of $G - S'$.
Output: A vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is an (r, ℓ)-graph, and an IC-partition (P_1, P_2) of $G - S$.

Before we solve Vertex Partization Compression, we explain how to reduce the Vertex Partization problem to $n-(r+\ell+k+1)+1$ instances of the Vertex Partization Compression problem on G_i, from $i = i_0$ to $i = n$. For the graph G_{i_0}, the set V_{k+1} is a (r, ℓ)-vertex deletion set, S'_i, of size $k+1$. The graph $G_{i_0} - V_{k+1}$ has $r + \ell$ vertices. We set $Q_1^{i_0}$ to be a set of any r vertices of $V_{i_0} - V_{k+1}$ and $Q_2^{i_0}$ to be the remaining set of ℓ vertices; that is, $Q_2^{i_0} = V_{i_0} - V_{k+1} - Q_1^{i_0}$. Now, let $I_i = (G_i, r, \ell, k, S'_i, (Q_1^i, Q_2^i))$ be the i^{th} instance of Vertex Partization Compression. If S_{i-1} is a k-sized solution for I_{i-1}, then $S_{i-1} \cup \{v_i\}$ is a $(k+1)$-sized (r, ℓ)-vertex deletion set for G_i. So, the iteration begins with the instance $I_{i_0} = (G_{i_0}, r, \ell, k, V_{k+1}, (Q_1^{i_0}, Q_2^{i_0}))$ and we try to obtain an (r, ℓ)-vertex deletion set of size at most k. If such a solution S_{i_0} exists, we set $S'_{i_0+1} = S_{i_0} \cup \{v_{i_0+1}\}$ and ask for a k-sized solution for the instance I_{i_0+1}. We continue in this manner. If, during any iteration, the corresponding instance does not have an (r, ℓ)-vertex deletion set of size at most k, then this implies that the original instance (G, r, ℓ, k) is a NO instance for Vertex Partization. If S_n is a k-sized (r, ℓ)-vertex deletion set for G_n, where $G_n = G$, then clearly (G, r, ℓ, k) is YES instance of Vertex Partization. Since there are at most $n - (r + \ell + k + 1) + 1$ iterations, the total time taken by the algorithm to solve Vertex Partization is at most

$n - (r + \ell + k + 1) + 1$ times the time taken to solve Vertex Partization Compression. Thus, if Vertex Partization Compression is FPT, it follows that Vertex Partization is FPT.

We can also *view* the input graph G of an instance of Vertex Partization Compression as an $(r+k+1, \ell)$-graph with IC-partition $(Q_1 \cup S', Q_2)$. Equivalently, Vertex Partization Compression has an input positive integers r, ℓ, k and a graph G that is an $(r + k + 1, \ell)$-graph and the objective is to decide whether there is a k-sized set $S \subseteq V(G)$ such that $G - S$ is an (r, ℓ)-graph. This view point allows us to design an FPT algorithm for Vertex Partization Compression. Towards that we first define some notations. Let G be a graph and $V(G) = \{v_1, \ldots, v_n\}$. For partition $P = (A, B)$ of $V(G)$ we define an n-length bit vector B_P^G corresponding to P as follows. We set the i^{th} bit to 0 if $v_i \in A$ and to 1 otherwise. For two n-length bit vectors $a = a_1 \ldots a_n$ and $b = b_1 \ldots b_n$, the Hamming distance between a and b, denoted by $\mathcal{H}(a, b)$, is the number of indices on which a and b mismatches. That is, $\mathcal{H}(a, b) = |\{(a_i, b_i) \; | a_i \neq b_i, i \in [n] \; \}|$. The Hamming distance for the bitvectors corresponding to two IC-partitions, of a graph, is bounded as given by the following proposition.

▶ **Proposition 9** ([15]). *Let G be a graph. Let Q be an IC-partition of G that realizes G as an (r', ℓ')-graph and P is an IC-partition of G that realizes G as an (r, ℓ)-graph. Then $\mathcal{H}(B_Q^G, B_P^G) \leq r'\ell + r\ell'$.*

The following lemma follows from Proposition 9.

▶ **Lemma 10.** *Let G be a perfect graph and (Q_1, Q_2) be an IC-partition that realizes G as an (r', ℓ')-graph. Let S be an (r, ℓ)-vertex deletion set for G such that P is an IC-partition of G that realizes $G - S$ as an (r, ℓ)-graph and let $Q = (Q_1 \setminus S, Q_2 \setminus S)$. Then $\mathcal{H}(B_Q^{G-S}, B_P^{G-S}) \leq r'\ell + r\ell'$.*

Lemma 10 implies that to solve Vertex Partization Compression it is enough to solve the following problem.

Short Vertex Partization **Parameter:** r, ℓ, k, ρ
Input: A perfect graph G, positive integers r, ℓ, k, ρ, and a partition $Q = (Q_1, Q_2)$ of $V(G)$.
Output: A vertex subset $S \subseteq V(G)$ of size at most k such that $G - S$ is a (r, ℓ)-graph, and an IC-partition (P_1, P_2) of $G - S$ such that $\mathcal{H}(B_P^{G-S}, B_{Q'}^{G-S}) \leq \rho$ where $Q' = (Q_1 \setminus S, Q_2 \setminus S)$.

▶ **Lemma 11.** Short Vertex Partization *is* FPT.

Proof. We design a recursive algorithm $\mathcal{A}$ for Short Vertex Partization which takes a tuple $(G, r, \ell, k, \rho, Q = (Q_1, Q_2))$ as input, where G is a graph, Q is a partition of $V(G)$ and r, ℓ, k, ρ are integers. It outputs a k-sized (r, ℓ)-vertex deletion set S of G and an IC-partition P that realizes that $G - S$ is (r, ℓ)-graph such that $\mathcal{H}(B_P^{G-S}, B_{Q'}^{G-S}) \leq \rho$, where $Q' = (Q_1 \setminus S, Q_2 \setminus S)$, if such a tuple (S, P) exists, otherwise returns NO. The following are the steps of the recursive algorithm $\mathcal{A}$ on input $(G, r, \ell, k, Q = (Q_1, Q_2), \rho)$.

1. If $k < 0$ or $\rho < 0$ then output NO.
2. If $G[Q_1]$ is r-colorable and $\overline{G}[Q_2]$ is ℓ-colorable, then return $(\emptyset, Q)$.
3. If $G[Q_1]$ is not r-colorable, then there is an $r + 1$-sized clique in $G[Q_1]$. By Proposition 5, we can find such a $r + 1$-sized clique C in polynomial time. Make the following recursive calls to $\mathcal{A}$:

(a) For every vertex $v \in V(C)$, do a recursive call $\mathcal{A}(G-v, r, \ell, k-1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\}))$ and if the recursive call returns (S', P) then return $(S' \cup \{v\}, P)$ as output.
(b) For every vertex $v \in V(C)$, do a recursive call $\mathcal{A}(G, r, \ell, k, \rho - 1, (Q_1 \setminus \{v\}, Q_2 \cup \{v\}))$ and if the recursive call returns (S', P) then return (S', P) as output.

If all the recursive calls above (in Step 3) return NO, then return NO.

4 If $\overline{G}[Q_2]$ is not ℓ-colorable, then there is clique of size $\ell + 1$ in $\overline{G}[Q_2]$. Thus, using Proposition 5, we can find an $\ell + 1$-sized independent set I in $G[Q_2]$. Make the following recursive calls of the algorithm:
(a) For every vertex $v \in V(I)$, do a recursive call $\mathcal{A}(G-v, r, \ell, k-1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\}))$ and if the recursive call returns (S', P) then return $(S' \cup \{v\}, P)$ as output.
(b) For every vertex $v \in V(I)$, do a recursive call $\mathcal{A}(G, r, \ell, k, \rho - 1, (Q_1 \cup \{v\}, Q_2 \setminus \{v\}))$ and if the recursive call returns (S', P) then return (S', P) as output.

If all the recursive calls above (in Step 4) return NO, then return NO.

We now prove that the recursive algorithm $\mathcal{A}$ is correct. We show that if $(G, r, \ell, k, \rho, (Q_1, Q_2))$ is a YES instance of Short Vertex Partization, then the algorithm $\mathcal{A}$ will output correct solution. We prove this by induction on $k + \rho$.

1. Base case: $k = 0$ and $\rho = 0$. Since $(G, r, \ell, k, \rho, (Q_1, Q_2))$ is a YES instance, (Q_1, Q_2) is an IC-partition which realizes that G is an (r, ℓ)-graph. In Step 2 of the algorithm $\mathcal{A}$ we output $(\emptyset, (Q_1, Q_2))$ as the output.
2. By induction hypothesis we assume that $\mathcal{A}$ outputs the correct answer when $k + \rho < \gamma$, where $\gamma \geq 0$. Now we need to show that $\mathcal{A}$ outputs correct answer when $k + \rho = \gamma$. Let $(G, r, \ell, k, \rho, Q = (Q_1, Q_2))$ be the input of $\mathcal{A}$ such that $k + \rho = \gamma$. Let $(S, (P_1, P_2))$ be a solution of Short Vertex Partization. If $S = \emptyset$ and $(P_1, P_2) = (Q_1, Q_2)$ then in Step 2, algorithm $\mathcal{A}$ will output $(\emptyset, Q)$. Otherwise either $G[Q_1]$ is not r-colorable or $\overline{G}[Q_2]$ is not ℓ-colorable.
3. Case 1: Suppose $G[Q_1]$ is not r-colorable. Then there is a clique C of size $r + 1$ in $G[Q_1]$. In this case at least one vertex v from C either belongs to S or does not belong to P_1. If $v \in S$, then consider the recursive call $\mathcal{A}(G - v, r, \ell, k - 1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\})$. By induction hypothesis $\mathcal{A}(G-v, r, \ell, k-1, \rho, (Q_1 \setminus \{v\}, Q_2 \setminus \{v\})$ will return (S', P') such that S' is a $k - 1$ sized (r, ℓ)-vertex deletion set of $G - \{v\}$ such that $\mathcal{H}(B_{P'}^{G-S'}, B_{Q'}^{G-S'}) \leq \rho$, where $Q' = (Q_1 \setminus (S' \cup \{v\}), Q_2 \setminus (S' \cup \{v\}))$. Hence in Step $3(a)$, algorithm $\mathcal{A}$ will output $(S' \cup \{v\}, P)$ and this is a solution for Short Vertex Partization on input $(G, r, \ell, k, \rho, (Q_1, Q_2))$.
If $v \notin S$, then $v \notin P_1$ as well. Now consider the recursive call $\mathcal{A}(G, r, \ell, k, \rho - 1, (Q_1 \setminus \{v\}, Q_2 \cup \{v\})$. By induction hypothesis $\mathcal{A}(G, r, \ell, k, \rho - 1, (Q_1 \setminus \{v\}, Q_2 \cup \{v\})$ will return (S'', P'') such that S'' is a k sized (r, ℓ)-vertex deletion set of G such that $\mathcal{H}(B_{P''}^{G-S''}, B_{Q'}^{G-S''}) \leq \rho - 1$, where $Q' = ((Q_1 \setminus \{v\}) \setminus S'', (Q_2 \cup \{v\}) \setminus S'')$. Hence in Step $3(b)$, algorithm $\mathcal{A}$ will output (S'', P''). Clearly S'' is a k sized (r, ℓ)-vertex deletion set of G''. Now we show that $\mathcal{H}(B_{P''}^{G-S''}, B_{Q''}^{G-S''}) \leq \rho$, where $Q'' = (Q_1 \setminus S'', Q_2 \setminus S'')$. Note that $\mathcal{H}(B_{Q'}^{G-S''}, B_{Q''}^{G-S''}) \leq 1$. Since $\mathcal{H}(B_{P''}^{G-S''}, B_{Q'}^{G-S''}) \leq \rho - 1$ and $\mathcal{H}(B_{Q'}^{G-S''}, B_{Q''}^{G-S''}) \leq 1$, we have that
$\mathcal{H}(B_{P''}^{G-S''}, B_{Q''}^{G-S''}) \leq \rho$
4. Case 2: $\overline{G}[Q_2]$ is not ℓ-colorable. This case is symmetric to Case 1.

In the reverse direction we need to show that if the algorithm $\mathcal{A}$ returns YES then indeed the given instance is a YES instance. The proof of reverse direction is similar to the proof of forward direction. Again, we induct on $k + \rho$. The algorithm returns a bipartition as evidence of a YES instance. We show that this bipartition is the required IC-partition. Such a bipartition is returned in Steps 2, $3(a), (b)$ and $4(a), (b)$. By description of the algorithm, both $k, \rho \geq 0$ in order for the algorithm to return YES. In the base case, $k = \rho = 0$. Here, it must be the case that Step 2 is executed and the output bipartition is Q itself, while the output vertex deletion set is $\emptyset$. In this case, by definition Q is an IC-partition. Hence, Q is evidence that the input graph G is already an (r, ℓ)-graph and does not require any vertex to be deleted. Thus, in the base case, we correctly determine that the given input instance is a YES instance.

By induction hypothesis we assume that $\mathcal{A}$ outputs the correct answer when $k + \rho < \gamma$, where $\gamma \geq 0$. Now we need to show that if $\mathcal{A}$ outputs a bipartition P and a vertex set S when $k + \rho = \gamma$, then the vertex set S is an (r, ℓ)-vertex deletion set while the bipartition is an IC-partition of $G - S$. If the bipartition is output in Step 2, then by definition the input graph is already an (r, ℓ)-graph. Otherwise, a recursive call is made to an instance where $k + \rho$ is strictly smaller. By induction hypothesis, the recursive call returns an (r, ℓ)-vertex deletion set S' and a witnessing IC-partition of $G - S'$. It follows that the vertex set S and the bipartition output by the algorithm, on the current input instance, is an (r, ℓ)-vertex deletion set of size at most k, and an IC-partition respectively for the input instance. This completes the proof of correctness of the algorithm.

What remains is the running time analysis. Note that when $k < 0$ or $\rho < 0$, then the algorithm will stop in a single step. Each recursive call either decreases k by 1 or ρ by 1. Hence the depth of the recursion tree is bounded by $k + \rho + 1$. Note that in Step 3 we make at most $2(r + 1)$ recursive calls and in Step 4 we make at most $2(\ell + 1)$ recursive calls. Hence the total running time of the algorithm $\mathcal{A}$ is bounded by $\mathcal{O}(\max\{(2(r+1))^{k+\rho+1} n^{\mathcal{O}(1)}, (2(\ell + 1))^{k+\rho+1} n^{\mathcal{O}(1)}\})$. ◀

Vertex Partization Compression is a special case of Short Vertex Partization when $\rho = (r + k + 1)\ell + r\ell$. Therefore, we have the following theorem.

▶ **Theorem 12.** Vertex Partization *on perfect graphs has an* FPT *algorithm with running time* $2^{\mathcal{O}((k+r)\ell \log(r+\ell))} n^{\mathcal{O}(1)}$.

4 Kernel lower bound for Vertex Partization

In this section, we show that Vertex Partization on perfect graphs does not have polynomial kernels. Partization Recognition on perfect graphs, when parameterized by $r + \ell$, was shown to be FPT in [15]. Our proof implies that Partization Recognition on perfect graphs cannot have a polynomial kernel, when parameterized by $r + \ell$, unless NP $\subseteq$ co-NP/poly.

▶ **Theorem 13.** Partization Recognition *on perfect graphs, when parameterized by* $r + \ell$, *does not have a polynomial kernel unless* NP $\subseteq$ co-NP*/poly.*

Proof. We prove the theorem by giving a polynomial parameter transformation from CNF-SAT parameterized by the number of variables. From Proposition 8, we know that CNF-SAT, parameterized by the number of variables, does not have a polynomial kernel unless NP $\subseteq$ co-NP/poly [11]. Then the proof of the theorem follows from Proposition 7. We start with an instance (ϕ, n) of CNF-SAT, where ϕ is a CNF formula with m clauses and n variables. Without loss of generality, we assume that there is no clause where both literals of a variable appear together, since such a clause will be satisfied by any assignment and hence can be removed. The polynomial parameter transformation produces an instance $(G, n, 1)$ of

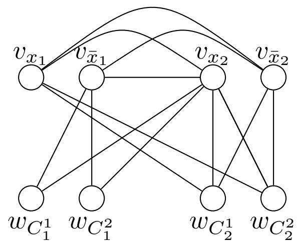


Figure 1 An illustration of the construction of the graph G in Theorem 13 for the formula $\phi = (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1)$. Here $C_1 = (x_1 \vee \overline{x}_2)$ and $C_2 = (\overline{x}_1)$.

Partization Recognition, where G is a perfect graph, such that (ϕ, n) is a YES instance of CNF-SAT if and only if $(G, n, 1)$ is a YES instance of Partization Recognition. Let $\mathcal{C} = \{C_1, \ldots, C_m\}$, $X = \{x_1, \ldots, x_n\}$ and $L = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ be the set of clauses, variables and literals of ϕ respectively. The construction of the graph G from the formula ϕ is as follows (illustrated in Figure 1):

1. For each variable x, we create two vertices $v_x, v_{\bar{x}}$ which represent the literals $x, \bar{x}$. We call them the literal vertices. More specifically, we call v_x the positive literal vertex and $v_{\bar{x}}$ the negative literal vertex.
2. For each clause C, we create two vertices w_C^1, w_C^2. We call these the clause vertices corresponding to the clause C.
3. For each pair of variables x, y, we add the edges $(v_x, v_y), (v_{\bar{x}}, v_y), (v_x, v_{\bar{y}})$, and $(v_{\bar{x}}, v_{\bar{y}})$. Notice that $(v_x, v_{\bar{x}})$ and $(v_y, v_{\bar{y}})$ are non-edges.
4. For each clause C and each literal $q \in L$, if $q \notin C$, we add edges (x_q, w_C^1) and (x_q, w_C^2). In other words if a literal q' belongs to a clause C, then $(q', w_C^1), (q', w_C^2) \notin E(G)$ So, there is a complete bipartite graph between $L \setminus C$ and $\{w_C^1, w_C^2\}$.

In short, the vertex set and edge set of G is defined as follows (note that for a literal x, if $x = \bar{y}$, then $y = \bar{x}$).

$$\begin{aligned} V(G) &= \{v_x, v_{\bar{x}} \mid x \in X\} \cup \{w_C^1, w_C^2 \mid C \in \mathcal{C}\} \\ E(G) &= \{(v_x, v_y) \mid x, y \in L, x \neq \bar{y}\} \cup \{(w_C^1, v_x), (w_C^2, v_x) \mid x \in L, x \notin C, C \in \mathcal{C}\} \end{aligned}$$

Let $V_X = \{v_x, v_{\bar{x}} \mid x \in X\}$ and $V_{\mathcal{C}} = \{w_C^1, w_C^2 \mid C \in \mathcal{C}\}$. Note that the set of vertices $V_{\mathcal{C}}$ corresponding to the clauses forms an independent set in G. First we show that G is a perfect graph.

▶ **Claim 14.** *The graph $\overline{G}$ does not contain an induced odd cycle of length ≥ 5.*

Proof. We first prove that there is no path of length 2 (path on 3 vertices) in $\overline{G}[V_X]$. Note that $E(\overline{G}[V_X]) = \{(v_x, v_{\bar{x}}) \mid x \in X\}$. This implies that the degree of each vertex in the graph $\overline{G}[V_X]$ is exactly equal to 1. Hence, there is no path of length 2 in $\overline{G}[V_X]$. Also, since $V_{\mathcal{C}}$ forms a clique in $\overline{G}$, any induced cycle of length at least 5 in $\overline{G}$ will either contain a vertex or an edge from $V_{\mathcal{C}}$.

Let C' be an induced odd cycle of length at least 5 in $\overline{G}$. There will be at most two vertices from $V_{\mathcal{C}}$ which are part of C' and these vertices will appear consecutively in C'. This implies that C' contains a path of length at least 2 using only vertices from V_X in $\overline{G}$, which is a contradiction. ◀

▶ **Claim 15.** *The graph G does not contain an induced odd cycle of length ≥ 5.*

Proof. We first show that any induced odd cycle of length at least 5 can contain at most 3 vertices from V_X. Suppose not. Let C' be an induced odd cycle of length at least 5 such that $|V(C') \cap V_X| \geq 4$. Let v_w, v_x, v_y, v_z be four distinct vertices from $V(C') \cap V_X$ appearing in that order, if we go around the cycle in a clockwise manner. That is, there are paths $v_w - v_x$, $v_x - v_y$, $v_y - v_z$ in C'. Since C' is an induced cycle, there is no edge (v_w, v_y) in $E(G)$. This implies that $y = \bar{w}$. By similar arguments, we can show that $x = \bar{z}$. This implies that $v_w v_x v_y v_z v_w$ form a cycle of length 4 in G, contradicting the fact that C' is induced odd cycle containing v_w, v_x, v_y and v_z. Hence any induced odd cycle of length at least 5 can contain at most 3 vertices from V_X.

Since $V_{\mathcal{C}}$ is an independent set in G, no two vertices from $V_{\mathcal{C}}$ can occur as consecutive vertices in any cycle. Let C' be an induced odd cycle of length at least 5 in G. Since $|V(C') \cap V_X| \leq 3$ and no two vertices from $V_{\mathcal{C}}$ appear as consecutive vertices in C', it must be the case that $|V(C') \cap V_{\mathcal{C}}| \leq 2$. This implies that the length of C' is exactly equal to 5 and C' is of the form $v_x w^i_{C_1} v_y w^j_{C_2} v_z v_x$, where $i, j \in \{1, 2\}$. Since C' is an induced cycle $(v_x, v_y), (v_y, v_z) \notin E(G)$. This implies that $y = \bar{x} = z$ and hence $v_y = v_z$. This contradicts the fact that C' is a cycle. This completes the proof of the claim. ◀

Proposition 3 and Claims 14 and 15 imply that G is a perfect graph. We now show that (ϕ, n) is a YES instance of CNF-SAT if and only if $(G, n, 1)$ is a YES instance of Partization Recognition.

First, suppose that (ϕ, n) is a YES instance of CNF-SAT. Then there is an assignment τ, such that every clause has at least one literal set to 1. Let $f : \mathcal{C} \to X$ be a map that arbitrarily maps one such satisfying literal to each clause. Note that for a clause C, $(w^1_C, v_{f(C)}), (w^2_C, v_{f(C)}) \notin E(G)$, because $f(C) \in C$. Now we construct n independent sets as follows. For each literal y, if $\tau(y) = 1$, let $I_y = \{w^1_C, w^2_C \mid f(C) = y\} \cup \{v_y\}$. Since $V_{\mathcal{C}}$ is an independent set $I_y \setminus \{v_y\}$ is an independent set. Note that for all $w^i_C \in I_y$, $i \in \{1, 2\}$ we have that $(w^i_C, v_y) \notin E(G)$, because $f(C) = y$ and $y \in C$. This implies that I_y is an independent set. Since τ is an assignment, exactly one of the literals of each variable is assigned 1 by τ. Thus, in this way we form n independent sets. Since τ is a satisfying assignment for ϕ, the function f maps each clause C to a literal which is assigned 1 by τ. This implies that all vertices in $V_{\mathcal{C}}$ are covered by the independent sets constructed above. The vertices in the graph G, which are not covered by the independent sets constructed, correspond to the literals that have been set to 0 by τ. By construction of G and by the definition of an assignment τ, these vertices form a clique. Hence, $(G, n, 1)$ is an YES instance of Partization Recognition.

Conversely, suppose $(G, n, 1)$ is a YES instance of Partization Recognition. Then there is an (r, ℓ)-partition $\mathcal{P}$ of G. Let $I_1, \ldots, I_n$ be n independent sets and K be a clique in the (r, ℓ)-partition $\mathcal{P}$. It is not possible, by construction of G, that there is a variable x such that both v_x and $v_{\bar{x}}$ belong to the clique K, because $(v_x, v_{\bar{x}}) \notin E(G)$. As there is only one clique in $\mathcal{P}$, at most one literal of each variable can be contained in the clique K of $\mathcal{P}$. Hence, for each variable x either v_x or $v_{\bar{x}}$ is part of an independent set in $\mathcal{P}$. Furthermore, since for two literals p and q such that $p \neq \bar{q}$, $(v_p, v_q) \in E(G)$, any independent set I in $\mathcal{P}$ cannot contain both v_p and v_q. This implies that each of the n independent sets can be identified by a variable $x \in X$. Since there are only n independent sets in $\mathcal{P}$, there cannot be a variable x such that both v_x and $v_{\bar{x}}$ are part of distinct independent sets in $\mathcal{P}$. Thus the construction of G forces only two possibilities for each variable:

(a) there is exactly one literal vertex that is part of an independent set while the other one belong to the clique K, or

(b) both literals together form an independent set.

Now we construct an assignment τ and show that τ is a satisfying assignment for ϕ. For a literal z, if $v_z \in K$, then we set $\tau(z) = 0$. If for a variable x, both vertices v_x and $v_{\bar{x}}$ do not belong to K then we set $\tau(x) = 1$. Now we show that τ is a satisfying assignment for ϕ. Let C be a clause in the formula ϕ. Since $(w_C^1, w_C^2) \notin E(G)$ at least one of w_C^1 or w_C^2 belongs to an independent set I in $\mathcal{P}$. Let $w_C^i \in I$, where $i \in \{1, 2\}$. We have shown that each independent set contains at least one vertex corresponding to a literal y. Since v_y and w_C^i belong to I, we have that $(v_z, w_C^i) \notin E(G)$. This implies that $y \in C$. Furthermore, this implies that $v_{\bar{y}} \notin I$ as no clause contains both y and $\bar{y}$. Hence, $v_{\bar{y}}$ is in K and $\tau(\bar{y}) = 0$. This implies that y is set to 1 and hence the clause C is satisfied. This proves that τ is a satisfying assignment for ϕ. ◀

Note that Partization Recognition is same as Vertex Partization, when $k = 0$. Hence we get the following corollary.

▶ **Corollary 16.** Vertex Partization *parameterized by* $k + r + \ell$ *on perfect graphs does not have a polynomial kernel unless* NP $\subseteq$ co-NP*/poly.*

5 Polynomial kernel when r and ℓ are constants

We saw that there is no polynomial kernel for Vertex Partization, unless NP $\subseteq$ co-NP/poly. The parameter for this problem is $k+r+\ell$, where the size of the deletion set is at most k and the final graph is an (r, ℓ)-graph. In this section, we consider the Vertex (r, ℓ) Partization problem on perfect graphs, which is a special case of Vertex Partization on perfect graphs. Here, for a given pair of fixed positive constants (r, ℓ), we take a perfect graph G and a positive integer k as input and decide whether there is a vertex set S of size at most k the deletion of which results in an (r, ℓ)-graph. This simplified problem has a polynomial kernel, as shown below.

We first observe that when perfect graphs are (r, ℓ)-graphs, this class coincides with another graph class, namely the class of perfect graphs that are (r, ℓ)-split graphs. The notion of (r, ℓ)-split graphs was introduced in [14].

▶ **Definition 17** ((r, ℓ)-split graph). A graph G is an (r, ℓ)-split graph if its vertex set can be partitioned into V_1 and V_2 such that the size of a largest clique in $G[V_1]$ is bounded by r and the size of a largest independent set in $G[V_2]$ is bounded by ℓ. We call such a partition, (V_1, V_2), an (r, ℓ)-split partition.

From the definition of (r, ℓ)-split graph, it follows that any (r, ℓ)-graph is also an (r, ℓ)-split graph.

▶ **Lemma 18.** *Let G be a perfect graph. If G is an (r, ℓ)-split graph, then G is also an (r, ℓ)-graph.*

Proof. Since G is a perfect graph, for any induced subgraph G' of G, the chromatic number of G' ($\chi(G')$) is equal to the cardinality of a maximum sized clique of G' ($\omega(G')$). We know that G is an (r, ℓ)-split graph. Let (P_1, P_2) be an (r, ℓ)-split partition with $\omega(G[P_1]) \leq r$ and $\alpha(G[P_2]) \leq \ell$ Now we show that (P_1, P_2) is indeed an (r, ℓ)-partition of G. Since G is a perfect graph, the graphs $G[P_1]$ and $\overline{G}[P_2]$ are perfect graphs. Since $G[P_1]$ is a perfect graph and $w(G[P_1]) \leq r$, $\chi(G[P_1]) \leq r$. This implies that P_1 can be partitioned into r independent sets. Since $\overline{G}[P_2]$ is a perfect graph and $\alpha(G[P_2]) \leq \ell$, $w(\overline{G}[P_2]) \leq \ell$ and hence $\chi(\overline{G}[P_2]) \leq \ell$. This implies that P_2 can be partitioned into ℓ sets such that each set is independent in $\overline{G}[P_2]$. Hence P_2 can be partitioned into ℓ cliques in $G[P_2]$. So (P_1, P_2) is an (r, ℓ)-partition of G. This completes the proof of the lemma. ◀

For any fixed r and ℓ, there is a finite forbidden set $\mathbb{F}_{r,\ell}$ for (r, ℓ)-split graphs [14]. That is, a graph G is an (r, ℓ)-split graph if and only if G does not contain any graph $H \in \mathbb{F}_{r,\ell}$ as an induced subgraph. The size of the largest forbidden graph is bounded by $f(r, \ell)$, f being a function given in [14]. In [18], a bound was given for the number of minimal forbidden perfect graphs. This function, say $g(r, \ell)$, has a bound of $2(\ell+1)(R(r(\ell+1), (r(l+1))^2+r(l+1)+3)+1)$. Since $g(r, \ell)$ is a constant, it is possible to compute the forbidden set $\mathbb{F}'_{r,\ell}$ of perfect forbidden graphs in polynomial time. Thus, the class of perfect (r, ℓ)-graphs has a refined finite forbidden characterization. This implies that Vertex (r, ℓ) Partization on perfect graphs reduces to the d-Hitting Set problem, where d is the constant $g(r, \ell)$. In an equivalent d-Hitting Set instance, the universe will be the set of vertices of the input graph G, while the family of sets will be the vertex sets of induced subgraphs of G that are isomorphic to a forbidden graph. The set sizes are bounded by $g(r, \ell)$. Hence, by [1], this problem has a polynomial kernel. This gives us the following theorem.

▶ **Theorem 19.** Vertex (r, ℓ) Partization *on perfect graphs admits a kernel of size* $k^{\mathcal{O}(d)}$ *and has an algorithm with running time* $d^k n^{\mathcal{O}(d)}$. *Here,* $d = g(r, \ell)$.

6 Conclusion

In this paper we studied the Vertex Partization problem on perfect graphs, and showed that it is FPT and does not admit a polynomial kernel. Furthermore, we observed that Vertex (r, ℓ) Partization has an induced finite forbidden characterization and utilized that to give a faster FPT algorithm and a polynomial kernel for the problem. However, the algorithms for Vertex (r, ℓ) Partization has a factor of $n^{\mathcal{O}(d)}$, where d depends on the size of a largest graph in the finite forbidden set. It would be interesting to replace the factor $n^{\mathcal{O}(d)}$ by $\tau(d) \cdot n^{\mathcal{O}(1)}$.

References

1 Faisal N. Abu-Khzam. A kernelization algorithm for d-hitting set. *J. Comput. Syst. Sci.*, 76(7):524–531, 2010.

2 Julien Baste, Luerbio Faria, Sulamita Klein, and Ignasi Sau. Parameterized complexity dichotomy for (r, ℓ)-vertex deletion. *CoRR*, abs/1504.05515, 2015.

3 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.

4 Andreas Brandstädt, Van Bang Le, and Thomas Szymczak. The complexity of some problems related to graph 3-colorability. *Discrete Applied Mathematics*, 89(1–3):59–73, 1998.

5 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

6 Maria Chudnovsky, Neil Robertson, Paul D. Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006.

7 Marek Cygan and Marcin Pilipczuk. Split vertex deletion meets vertex cover: New fixed-parameter and exact exponential-time algorithms. *Inf. Process. Lett.*, 113(5-6):179–182, 2013.

8 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

9 Tomas Feder, Pavol Hell, Sulamita Klein, and Rajeev Motwani. List partitions. *STOC*, 16:464–472, 2003.

10 Tomás Feder, Pavol Hell, and Shekoofeh Nekooei Rizi. Partitioning chordal graphs. *Electronic Notes in Discrete Mathematics*, 38:325–330, 2011.

11 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct pcps for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

13 Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015.

14 András Gyárfás. Generalized split graphs and ramsey numbers. *J. Comb. Theory, Ser. A*, 81(2):255–261, 1998.

15 Pinar Heggernes, Dieter Kratsch, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fixed-parameter algorithms for cochromatic number and disjoint rectangle stabbing via iterative localization. *Inf. Comput.*, 231:109–116, 2013.

16 Sudeshna Kolay and Fahad Panolan. Parameterized algorithms for deletion to (r, ell)-graphs. In *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, pages 420–433, 2015.

17 R. Krithika and N. S. Narayanaswamy. Parameterized algorithms for (r, l)-partization. *J. Graph Algorithms Appl.*, 17(2):129–146, 2013.

18 André E. Kézdy, Hunter S. Snevily, and Chi Wang. Partitioning permutations into increasing and decreasing subsequences. *Journal of Combinatorial Theory, Series A*, 73(2):353–359, 1996.

19 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is np-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

20 Daniel Lokshtanov, N. S. Narayanaswamy, Venkatesh Raman, M. S. Ramanujan, and Saket Saurabh. Faster parameterized algorithms using linear programming. *ACM Transactions on Algorithms*, 11(2):15, 2014.

21 László Lovász. A characterization of perfect graphs. *J. Comb. Theory, Ser. B*, 13(2):95–98, 1972.

22 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. `doi:10.1016/j.orl.2003.10.009`.

23 Klaus W. Wagner. Monotonic coverings of finite sets. *Elektronische Informationsverarbeitung und Kybernetik*, 20(12):633–639, 1984.

Supplementarity is Necessary for Quantum Diagram Reasoning*

Simon Perdrix[1] and Quanlong Wang[2]

1 CNRS, France; and
Inria project team Carte, LORIA, Université de Lorraine, France
simon.perdrix@loria.fr

2 Inria project team Carte, LORIA, Université de Lorraine, France; and
Department of Computer Science, University of Oxford, UK
quanlong.wang@wolfson.ox.ac.uk

Abstract

The ZX-calculus is a powerful diagrammatic language for quantum mechanics and quantum information processing. We prove that its $\frac{\pi}{4}$-fragment is not complete, in other words the ZX-calculus is not complete for the so called "Clifford+T quantum mechanics". The completeness of this fragment was one of the main open problems in *categorical quantum mechanics*, a programme initiated by Abramsky and Coecke. The ZX-calculus was known to be incomplete for quantum mechanics. On the other hand, its $\frac{\pi}{2}$-fragment is known to be complete, i.e. the ZX-calculus is complete for the so called "stabilizer quantum mechanics". Deciding whether its $\frac{\pi}{4}$-fragment is complete is a crucial step in the development of the ZX-calculus since this fragment is approximately universal for quantum mechanics, contrary to the $\frac{\pi}{2}$-fragment.

To establish our incompleteness result, we consider a fairly simple property of quantum states called supplementarity. We show that supplementarity can be derived in the ZX-calculus if and only if the angles involved in this equation are multiples of $\pi/2$. In particular, the impossibility to derive supplementarity for $\pi/4$ implies the incompleteness of the ZX-calculus for Clifford+T quantum mechanics. As a consequence, we propose to add the supplementarity to the set of rules of the ZX-calculus.

We also show that if a ZX-diagram involves antiphase twins, they can be merged when the ZX-calculus is augmented with the supplementarity rule. Merging antiphase twins makes diagrammatic reasoning much easier and provides a purely graphical meaning to the supplementarity rule.

1998 ACM Subject Classification F.1.1 Models of Computation, F.3.2. Semantics of Programming Languages

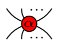
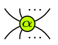
Keywords and phrases Quantum Diagram Reasoning, Completeness, ZX-calculus, Quantum Computing, Categorical Quantum Mechanics

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.76

1 Introduction

The ZX-calculus has been introduced by Coecke and Duncan [7] as a graphical language for pure state qubit quantum mechanics where each diagram can be interpreted as a linear map or a matrix in a typical way (so-called *standard interpretation*). Intuitively, a ZX-diagram

* This work was partially supported by Région Lorraine.

is made of three kinds of vertices: , , and , where each green or red vertex is parameterised by an angle.

Unlike the quantum circuit notation which has no transformation rules, the ZX-calculus combines the advantages of being intuitive with a built-in system of rewrite rules. These rewrite rules make the ZX-calculus into a formal system with nontrivial equalities between diagrams. As shown in [7], the ZX-calculus can be used to express any operation in pure state qubit quantum mechanics, i.e. it is *universal*. Furthermore, any equality derived in the ZX-calculus can also be derived in the standard matrix mechanics, i.e. it is *sound*.

The converse of soundness is *completeness*. Informally put, the ZX-calculus would be complete if any equality that can be derived using matrices can also be derived graphically. It has been shown in [15] that the ZX-calculus is incomplete for the overall pure state qubit quantum mechanics, and there is no way on how to complete it by now. However, some fragments of the ZX-calculus are known to be complete. The $\frac{\pi}{2}$-fragment, which corresponds to diagrams involving angles multiple of $\pi/2$, is complete [1]. This fragment corresponds to the so called stabilizer quantum mechanics [14]. The π-fragment is also complete [12] and corresponds to real stabilizer quantum mechanics. Meanwhile, the stabilizer completeness proof in [1] carries over to a ZX-like graphical calculus for Spekkens′ toy theory [4].

While it is an important and active area of research, stabilizer quantum mechanics is only a small part of all quantum mechanics. In particular stabilizer quantum mechanics is not universal, even approximately. This fragment is even efficiently simulatable on a classical computer. On the contrary, the $\frac{\pi}{4}$-fragment, which corresponds to the so-called "Clifford+T quantum mechanics" is approximately universal [6]: any unitary transformation can be approximated with an arbitrary precision by a diagram involving angles multiple of $\pi/4$ only. The $\frac{\pi}{4}$-fragment corresponds actually to the post selected Clifford+T quantum mechanics: any diagram of the $\frac{\pi}{4}$-fragment can be interpreted as a composition of: (*i*) preparations of qubits in the computational basis; (*ii*) "Clifford+T" unitary transformations; (*iii*) post selected measurements in the computational basis. Post selected measurements, which are noting but projections, are useful to compute the probability that a given computation produces a given output. An actual quantum measurement which, roughly speaking, consists in applying with some probability a projector among a complete set of projectors, can also be represented as a ZX-diagram using formal variables as described in [11].

The completeness of the $\frac{\pi}{4}$-fragment is a crucial property and has even been stated as one of the major open question in the categorical approach to quantum mechanics [1, 2, 17]. A partial result has been proved in [2]: the fragment composed of path diagrams involving angles multiple of $\pi/4$ is complete.

Our main contribution is to prove that the $\frac{\pi}{4}$-fragment of the ZX-calculus is incomplete. In other words, we prove that the ZX-calculus is not complete for the "Clifford+T quantum mechanics". To this end, we consider a simple equation called supplementarity. This equation is inspired by a work by Coecke and Edwards [8] on the structures of quantum entanglement. We show that supplementarity can be derived in the ZX-calculus if and only if the angles involved in this equation are multiples of $\pi/2$. In particular, the impossibility to derive this equation for $\pi/4$ implies the incompleteness of the ZX-calculus for Clifford+T quantum mechanics.

We also show that in the ZX-calculus augmented with the supplementarity rule, antiphase twins can be merged. A pair of antiphase twins is a pair of vertices which have: the same colour; the same neighbourhood; and antiphase angles (the difference between their angles is π). Merging antiphase twins makes diagrammatic reasoning much easier and provides a purely graphical meaning to the supplementarity rule.

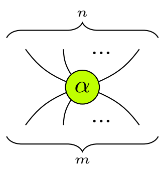
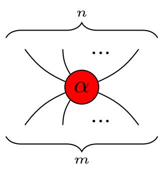
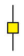
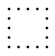
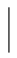
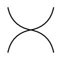
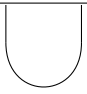
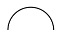
Notice that various slightly different notions of soundness/completeness have been used so far in the context of the ZX-calculus, depending on whether the rules of the language should strictly preserve the standard interpretation (as used in this paper), or up to a global phase, or even up to a (non-zero) scalar. Our result of incompleteness applies to any of these variants. However, we believe that the recent attempts to treat carefully the scalars and in particular the zero scalar are valuable, that is why we consider in this paper the strict notion of soundness and completeness. It should also be noticed that the notion of completeness used in the context of the ZX-calculus is different from a related one used in [16] to prove that finite dimensional Hilbert spaces are complete for dagger compact closed categories. The difference lies in that the concept of completeness used in the present paper is only concerned with the standard interpretation in finite dimensional Hilbert spaces, whereas, roughly speaking, in [16] it is considered for every possible interpretation (of object variables as spaces and morphism variables as linear maps).

This paper is structured as follows: the ZX-calculus (diagrams, standard interpretation, and rules) is presented in Section 2. Section 3 is dedicated to the supplementarity equation. In Section 4 we show that supplementarity involving angles which are not multiples of $\pi/2$ cannot be derived in the ZX-calculus which implies the incompleteness of the $\frac{\pi}{4}$-fragment. In Section 5, we identify an infinite family of equations which derivations require the supplementarity rule, and given a graphical interpretation of supplementarity by means of antiphase twins. Finally, in Section 6, we discuss the simplification of the ZX-calculus augmented with the supplementarity rule and its completeness.

2 ZX-calculus

2.1 Diagrams and standard interpretation

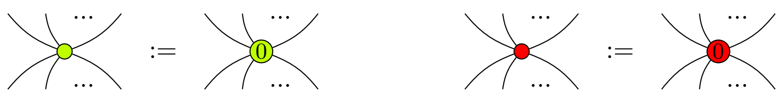
A ZX-diagram $D : k \to l$ with k inputs and l outputs is generated by:

$R_Z^{(n,m)}(\alpha) : n \to m$		$R_X^{(n,m)}(\alpha) : n \to m$	
$H : 1 \to 1$		$e : 0 \to 0$	
$\mathbb{I} : 1 \to 1$		$\sigma : 2 \to 2$	
$\epsilon : 2 \to 0$		$\eta : 0 \to 2$	

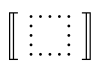
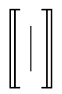
where $m, n \in \mathbb{N}$ and $\alpha \in [0, 2\pi)$.

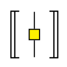
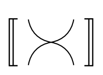
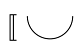
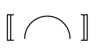
- Spacial composition: for any $D_1 : a \to b$ and $D_2 : c \to d$, $D_1 \otimes D_2 : a + c \to b + d$ consists in placing D_1 and D_2 side-by-side, D_2 on the right of D_1.
- Sequential composition: for any $D_1 : a \to b$ and $D_2 : b \to c$, $D_2 \circ D_1 : a \to c$ consists in placing D_1 on the top of D_2, connecting the outputs of D_1 to the inputs of D_2.

When equal to 0 modulo 2π the angles of the green and red dots are omitted:

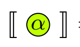
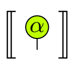
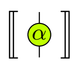
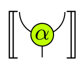


The standard interpretation of the ZX-diagrams associates with any diagram $D : n \to m$ a linear map $[\![D]\!] : \mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$ inductively defined as follows:

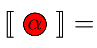
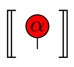
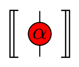
$[\![D_1 \otimes D_2]\!] := [\![D_1]\!] \otimes [\![D_2]\!]$ $\quad [\![D_2 \circ D_1]\!] := [\![D_2]\!] \circ [\![D_1]\!]$  $:= 1$  $:= \begin{pmatrix}10\\01\end{pmatrix}$

 $:= \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & \text{-}1\end{pmatrix}$  $:= \begin{pmatrix}1000\\0010\\0100\\0001\end{pmatrix}$  $:= (1001)$  $:= \begin{pmatrix}1\\0\\0\\1\end{pmatrix}$

$[\![R_Z^{(0,0)}(\alpha)]\!] := 1{+}e^{i\alpha}$, and when $a{+}b > 0$, $[\![R_Z^{(a,b)}(\alpha)]\!]$ is a matrix with 2^a columns and 2^b rows such that all entries are 0 except the top left one which is 1 and the bottom right one which is $e^{i\alpha}$, e.g.:

 $= 1 + e^{i\alpha}$  $= \begin{pmatrix}1\\e^{i\alpha}\end{pmatrix}$  $= \begin{pmatrix}1 & 0\\0 & e^{i\alpha}\end{pmatrix}$  $= \begin{pmatrix}1 & 0 & 0 & 0\\0 & 0 & 0 & e^{i\alpha}\end{pmatrix}$

For any $a, b \geq 0$, $[\![R_X^{a,b}(\alpha)]\!] := [\![H]\!]^{\otimes b} \times [\![R_Z^{a,b}(\alpha)]\!] \times [\![H]\!]^{\otimes a}$, where $M^{\otimes 0} = 1$ and for any $k > 0$, $M^{\otimes k} = M \otimes M^{\otimes k-1}$. E.g.,

 $= 1 + e^{i\alpha}$  $= \sqrt{2}e^{i\frac{\alpha}{2}}\begin{pmatrix}\cos(\alpha/2)\\ \text{-}i\sin(\alpha/2)\end{pmatrix}$  $= e^{i\frac{\alpha}{2}}\begin{pmatrix}\cos(\alpha/2) & \text{-}i\sin(\alpha/2)\\ \text{-}i\sin(\alpha/2) & \cos(\alpha/2)\end{pmatrix}$

ZX-diagrams are universal in the sense that for any $m, n \geq 0$ and any linear map $U : \mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$, there exists a diagram $D : n \to m$ such that $[\![D]\!] = U$ [7]. In particular, any unitary quantum evolution on a finite number of qubits can be represented by a ZX-diagram. Notice that universality implies working with a uncountable set of angles. As a consequence, the approximate version of universality, i.e. the ability to approximate with arbitrary accuracy any linear map, is generally preferred in quantum information processing. The $\frac{\pi}{4}$-fragment of language, which consists of all diagrams which angles are multiples of $\pi/4$, is approximately universal, whereas the $\frac{\pi}{2}$-fragment is not.

2.2 Calculus

The representation of a matrix in this graphical language is not unique. We present in this section the rules of the ZX calculus. These rules are sound in the sense that if two diagrams D_1 and D_2 are equal according to the rules of the ZX calculus, denoted $ZX \vdash D_1 = D_2$, then $[\![D_1]\!] = [\![D_2]\!]$. The rules of the language are given in Figure 1, and detailed bellow. The colour-swapped version and upside-down version of each rule given in Figure 1 also apply.

Spider. According to the (S1) rule any two directly connected green dots can be merged. Moreover, a dot with a single input, single output and angle 0 can be removed according to the (S2) rule. These rules have their origins in the axiomatisation of orthonormal bases by means of dagger special Frobenius algebras (see [9] for details). According to the standard interpretation $[\![.]\!]$, the green dots are associated with the so-called standard basis $\{\binom{1}{0}, \binom{0}{1}\}$, whereas the red dots (which also satisfies the spider property since colour-swapped rules also apply) are associated with the so-called diagonal basis $\{\frac{1}{\sqrt{2}}\binom{1}{1}, \frac{1}{\sqrt{2}}\binom{1}{\text{-}1}\}$.

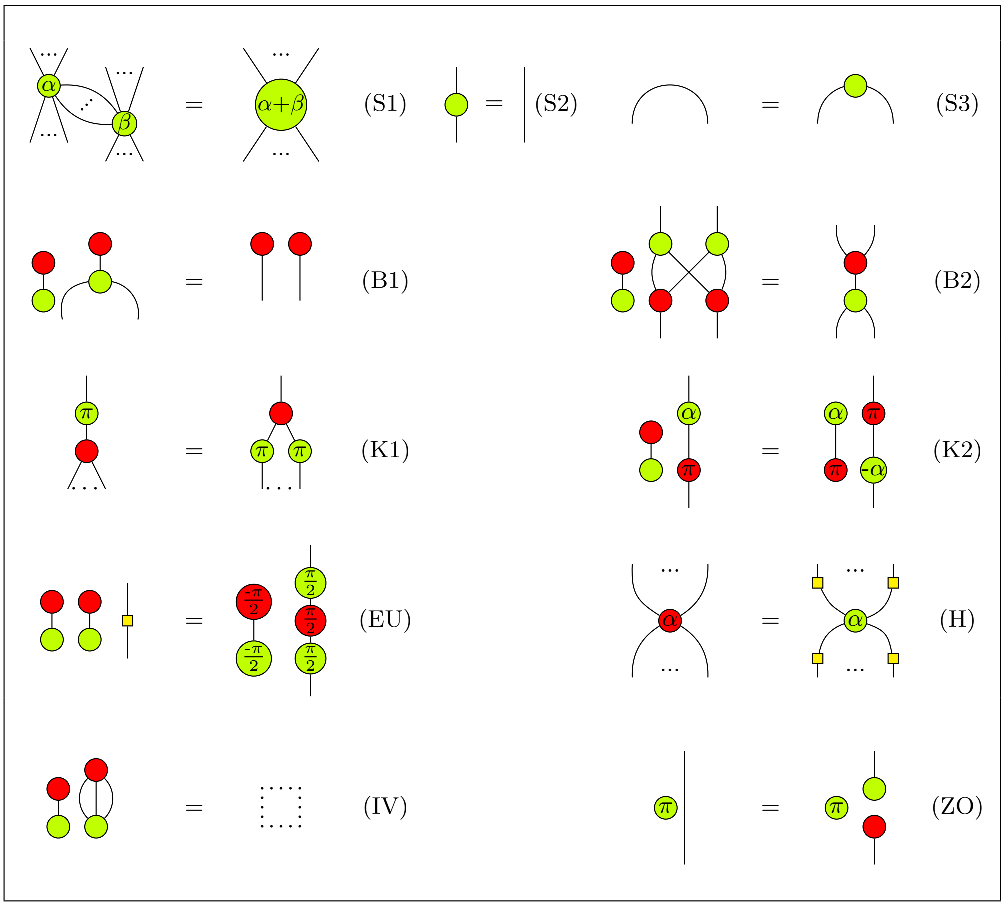
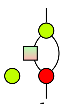
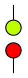
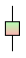
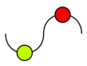
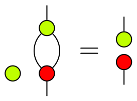


Figure 1 Rules of ZX-calculus. The colour-swapped and/or upside-down versions of each rule also applies. Horizontal dots (. . .) mean ‘arbitrary number’, whereas diagonal dots (⋰) mean ‘at least one’.

Green-Red Interactions. Monochromatic diagrams are lax: according to the (S1) rule any (green- or red-) monochromatic connected diagram is equivalent to a single dot with the appropriate number of legs and which angle is the sum of the angles. Thus the interesting structures arises when the two colours interact. The bialgebra rule (B1) and the copy rule (B2), imply that the red and the green bases are complementary, which roughly speaking capture the notion of uncertainty principle and of unbiasedness a fundamental property in quantum information (see [7] for details).

Parallel wires and Hopf law. (B1) and (B2) rules imply the following Hopf law [7, 10]:  =  where  :=  is the called the antipode. The (S3) rule trivialises the antipode and simplifies the Hopf law:

 **(Hopf Law)**

Hopf law has then a simple graphical meaning: two parallel wires between dots of distinct colours can be removed (up to the scalar). Notice that any pair of complementary basis in arbitrary finite dimension satisfies the rules (S1), (S2), (B1) and (B2). However the (S3) rule implies that the dimension of the corresponding Hilbert space is a power of two. As a consequence the ZX-calculus is a language dedicated to *qubit quantum mechanics*.

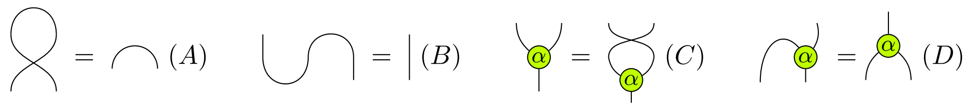
Classical point. In the context of complementary basis, the rules (K1) and (K2) imply that is a classical point. Intuitively, it means that together with are two elements of the red basis, so in dimension 2 they form an orthogonal basis.

Colour change. According to the (H) rule, can be used to change the colour of a dot. The (EU) rule corresponds to the Euler decomposition of the Hadamard matrix into three elementary rotations.

Scalar and zero. A scalar is a diagram with no input and no output. The standard interpretation of such a diagram is a complex number. While for simplicity, scalars have been ignored in several versions of the ZX calculus [7, 1], recently several rules have been introduced for scalars [3] and then simplified in [5], leading to the two rules (IV) and (ZO) presented in Figure 1. As the interpretation of the empty diagram is 1, the (IV) rule implies that is the inverse of . The interpretation of is 0, as a consequence for any diagrams D_1 and D_2, $[\![$ $\otimes D_1]\!] = [\![$ $\otimes D_2]\!]$. This absorbing property is captured by the (ZO) rule.

Context. The axioms of the language presented in Figure 1 can be applied to any subdiagram. In other word, if $ZX \vdash D_1 = D_2$ then, for any D (with the appropriate number of inputs/ouputs), $ZX \vdash D \otimes D_1 = D \otimes D_2$; $ZX \vdash D_1 \otimes D = D_2 \otimes D$; $ZX \vdash D \circ D_1 = D \circ D_2$; and $ZX \vdash D_1 \circ D = D_2 \circ D$.

Only topology matters. A ZX-diagram can be deformed without changing its interpretation. This property is known as "only topology matters" in [7]. E.g.



"Only topology matters" is a consequence of the underlying dagger compact closed structure (e.g. Eq. A and B), together with the ability to interchange any two legs (Eq. C) and to turn inputs into outputs (Eq D) and vice-versa. Equations C and D are non standard in dagger compact closed categories, and are consequences of the other rules of the ZX-calculus [5].

2.3 Soundness and Completeness

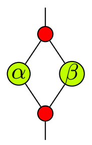
(In-)Completeness. All the rules of the ZX calculus are sound with respect to the standard interpretation, i.e. if $ZX \vdash D_1 = D_2$ then $[\![D_1]\!] = [\![D_2]\!]$. The converse of soundness is completeness: the language would be complete if $[\![D_1]\!] = [\![D_2]\!]$ implies $ZX \vdash D_1 = D_2$. The completeness would imply that one can forget matrices and do graphical reasoning only. Completeness would also imply that all the fundamental properties of qubit quantum mechanics are graphically captured by the rules of the ZX-calculus. This desirable property is one of the main open questions in categorical quantum mechanics. In the following , we review the known results about the completeness of the ZX-calculus, which are essentially depending on the considered fragment (restriction on the angles) of the language.

The very first result of incompleteness was about the original ZX-calculus in which the Euler decomposition[1] of H, the (EU) rule in Figure 1 was not derivable. This equation is now part of the language. Backens [1] proved that the $\frac{\pi}{2}$ fragment is complete. Schröder and Zamdzhiev proved that the language is not complete in general. Their argument is also based on some Euler decomposition, but contrary to the previous case this decomposition involves non rational multiples of π. The most natural – and actually the only known way – to bypass this incompleteness result is to consider a fragment of the language. Indeed, irrational multiple of π are not necessary for approximate universality. As the $\frac{\pi}{2}$-fragment is not approximately universal, the most interesting candidate for completeness is the $\frac{\pi}{4}$-fragment which is approximately universal. The completeness for the $\frac{\pi}{4}$-fragment has been conjectured in [2] and actually proved in the single qubit case, i.e. for path diagrams. The use of path diagrams (diagrams with all dots of degree two) is rather restrictive, but the completeness for this class of diagrams is not trivial and is sufficient to show that any argument based on some Euler decomposition cannot be applied in the $\pi/4$ case. However, we disprove the conjecture: the $\frac{\pi}{4}$-fragment of the ZX-calculus is not complete (Corollary 3), using a novel approach not based on Euler decompositions.

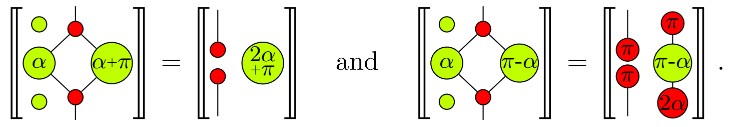
Scalars and completeness. In several versions of the ZX-calculus scalars are ignored, leading to a slightly different notion of soundness and completeness involving proportionality. Roughly speaking, ignoring the scalars consists in an additional rule which allows one to freely add or remove diagram with no input/output. A particular attention has to be paid to 'zero' diagrams, i.e. diagrams whose interpretations are zero, like π. When scalars are ignored, the notion of soundness is modified as follows: if $D_1 = D_2$ then $[\![D_1]\!]$ and $[\![D_2]\!]$ are proportional. The definition of completeness is modified likewise. Notice that in [15] yet another notion of soundness is considered where scalars are not ignored in general but global phases are, i.e. if $D_1 = D_2$ then $\exists\theta, [\![D_1]\!] = e^{i\theta}[\![D_2]\!]$. Our main result of incompleteness (Theorem 2) applies for any of these variants of soundness/completeness. However, we believe that the recent attempts to treat carefully the scalars and in particular the zero scalar are valuable that is why we consider in this paper the strict notion of soundness and completeness.

3 Supplementarity

In [8], Coecke and Edwards introduced the notion of *supplementarity* by pointing out that when $\alpha \neq 0 \bmod \pi$ the standard interpretation of the following diagram is proportional to the projector $\left(\begin{smallmatrix}10\\00\end{smallmatrix}\right)$ if $\alpha - \beta = \pi$ and to the projector $\left(\begin{smallmatrix}00\\01\end{smallmatrix}\right)$ if $\alpha + \beta = \pi$.



Putting back the scalars, one gets the following equations, which are true for any angle α, even when $\alpha = 0$:

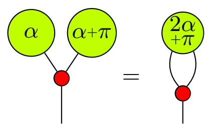


[1] By Euler decomposition we mean the existence, for any 1-qubit unitary U, of 4 angles $\alpha, \beta, \gamma, \delta$ s.t. $U = e^{i\alpha} R_x(\beta) R_z(\gamma) R_x(\delta)$ where $R_x(.)$ and $R_z(.)$ are elementary rotations about orthogonal axis.

Coecke and Edwards showed that the concept of supplementarity is related to the entanglement of quantum states. Up to stochastic local operations and classical communications (SLOCC), there are only two kinds of three-qubit states with genuine tripartite entanglement: those which are SLOCC-equivalent to a GHZ state, and those which are SLOCC-equivalent to a W state. A GHZ-state is a particular instance of a graph state which can be easily represented with a ZX-diagram [13]. On the other hand this is more involved to represent a W-type entangled states. The concept of supplementarity allowed Coecke and Edwards to characterise inhabitants of the W-class.

Albeit Coecke and Edwards did not address explicitly the question of proving whether the above equations can be derived in the ZX-calculus or not, these equations were known to be candidates for proving the incompleteness of the language[2]. We prove in Section 4 that these equations can be derived in the ZX-calculus only when $\alpha = 0 \bmod \frac{\pi}{2}$.

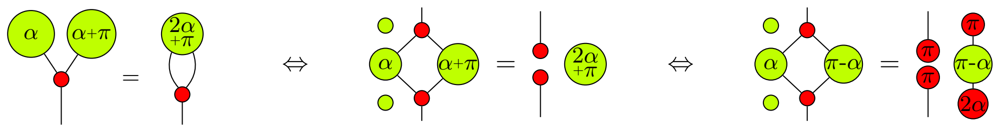
Inspired by the property pointed out by Coecke and Edwards we introduce the following equation that we call *supplementarity*:



(1)

Supplementarity is sound in the sense that both diagrams of (Eq. 1) have the same standard interpretation $\frac{1}{\sqrt{2}}\binom{1-e^{2i\alpha}}{0}$. It is provable in the ZX-calculus that supplementarity (Eq. 1) is equivalent to the equations pointed out by Coecke and Edwards:

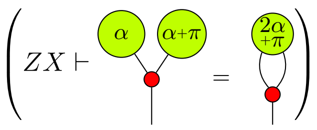
▶ **Lemma 1.** *In the ZX calculus, for any $\alpha \in [0, 2\pi)$:*



4 Supplementarity is necessary

In this section, we prove the main result of the paper: supplementarity involving angles which are not multiples of $\frac{\pi}{2}$ cannot be derived using the rules of the ZX-calculus, and as a corollary the $\frac{\pi}{4}$-fragment of ZX-calculus is incomplete.

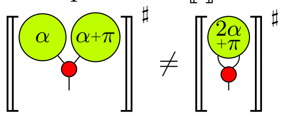
▶ **Theorem 2.** *Supplementarity can be derived in the ZX-calculus only for multiples of $\pi/2$:*



$$\Leftrightarrow \qquad \alpha = 0 \bmod \frac{\pi}{2}$$

▶ **Corollary 3.** *The $\frac{\pi}{4}$-fragment of ZX-calculus is not complete. In other words, ZX-calculus is not complete for the so-called "Clifford+T quantum mechanics".*

The rest of the section is dedicated to the proof of Theorem 2. To do so, we introduce an alternative interpretation $[\![.]\!]^\sharp$ for the diagrams, that we prove to be sound (Lemma 5) but for which

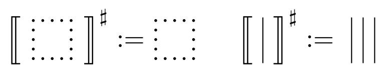


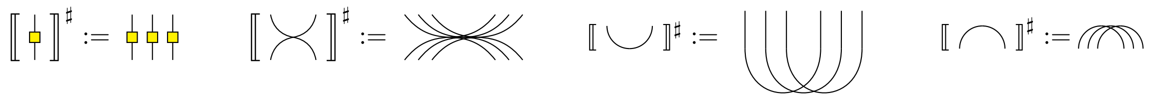
when $\alpha \neq 0 \bmod \frac{\pi}{2}$.

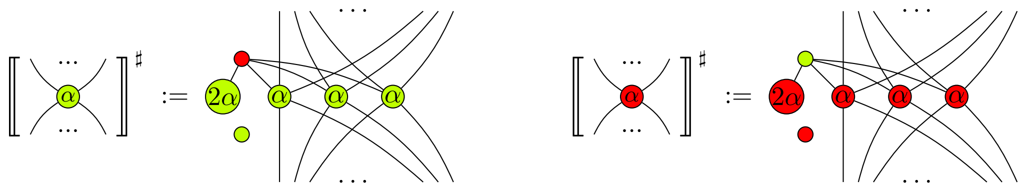
[2] Personal communications with Miriam Backens and Aleks Kissinger.

▶ **Definition 4.** For any diagram $D : n \to m$, let $[\![D]\!]^\sharp : 3n \to 3m$ be a diagram defined as follows:

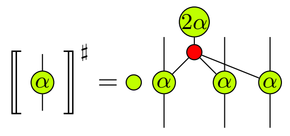
$[\![D_1 \otimes D_2]\!]^\sharp := [\![D_1]\!]^\sharp \otimes [\![D_2]\!]^\sharp \qquad [\![D_2 \circ D_1]\!]^\sharp := [\![D_2]\!]^\sharp \circ [\![D_1]\!]^\sharp$



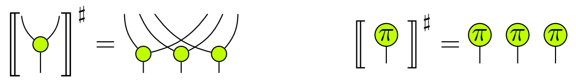




Roughly speaking, $[\![D]\!]^\sharp$ consists of three copies of D together with, for each dot of angle α, a *gadget* parameterized by the angle 2α connecting the three copies of the dot. E.g.

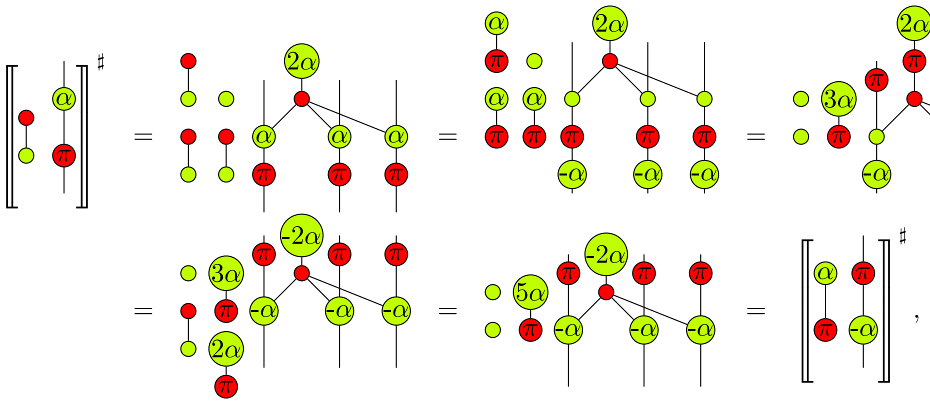


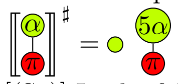
Simple calculations show that the gadget disappears when $\alpha = 0 \bmod \pi$, e.g.:



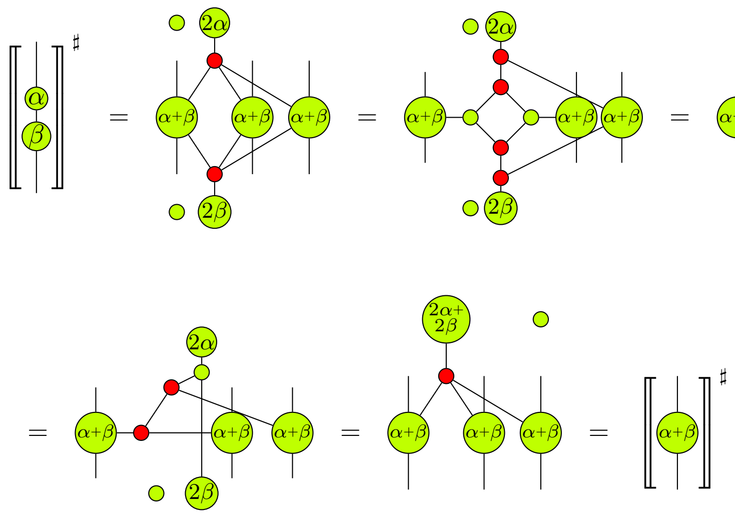
▶ **Lemma 5** (Soundness). *$[\![.]\!]^\sharp$ is a sound interpretation: if $ZX \vdash D_1 = D_2$ then $ZX \vdash [\![D_1]\!]^\sharp = [\![D_2]\!]^\sharp$.*

Proof. Soundness is trivial for the π-fragment of the language (i.e. when angles are multiples of π). Thus, it remains the four rules $(S1)$, $(K2)$, (EU), and (H) to complete the proof. We give the proof of $(K2)$ and a particular case of $(S1)$ to illustrate the proof, the other cases are omitted.
[(K2)]

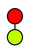


The first equality is nothing but the definition of $[\![.]\!]^\sharp$. The second step is based on the (K2) rule. The third step consists in (i) grouping the 3 scalars depending on α into a single one, to do so rules (B1), (K1) and finally (S1) are combined; (ii) applying the (K1) rule on the non scalar part of the diagram. Fourth step consists in applying the (K2) rule on the gadget. The fifth step is combining the scalars depending on α. Finally for the last step we use .

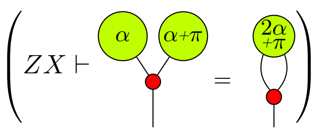
[(S1)] In the following we consider a particular case of the (S1) rule where the two dots are of degree 2. The following derivation essentially consists in applying the bialgebra rule (B2) twice:



◀

▶ Remark (1). The interpretation $[\![.]\!]^\sharp$ can be naturally extended to an interpretation $[\![.]\!]^\sharp_{k,\ell}$ which associates with every diagram $D : n \to m$ a diagram $[\![D]\!]^\sharp_{k,\ell} : k \times n \to k \times m$ which consists in k copies of D where the k copies of each dot are connected by a "gadget" parameterized by an angle ℓ times larger than the angle of the original dot. Moreover $[\![D]\!]^\sharp_{k,\ell}$ has additional scalars, namely $k-1$ times  per dot in D. Notice that the interpretation $[\![.]\!]^\sharp$ used in this section is nothing but $[\![.]\!]^\sharp_{3,2}$. The interpretation $[\![.]\!]^\sharp_{k,\ell}$ is sound if and only if $k = 1 \bmod 2$ and $\ell = 0 \bmod 2$, indeed $(K1)$ forces k to be odd while (EU) and (ZO) force $\ell = 0 \bmod 2$. All the other rules are sound for any k, ℓ. When $k = 1$, $[\![.]\!]^\sharp_{1,\ell}$ is nothing but an interpretation which multiplies the angles by $\ell + 1$, without changing the structure of the diagrams: $[\![.]\!]^\sharp_{1,0}$ is the identity, while $[\![.]\!]^\sharp_{1,-1}$ has been used to prove that the (EU) rule is necessary [13] and $[\![.]\!]^\sharp_{1,-2}$ has been used to prove that the ZX-calculus is incomplete [15].

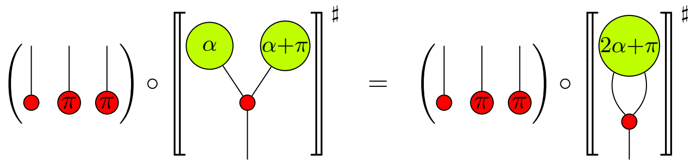
Proof of Theorem 2. In the following we prove that supplementarity can be derived in the ZX-calculus if and only if the involved angles are multiples of $\pi/2$:



$$\Leftrightarrow \qquad \alpha = 0 \bmod \frac{\pi}{2}$$

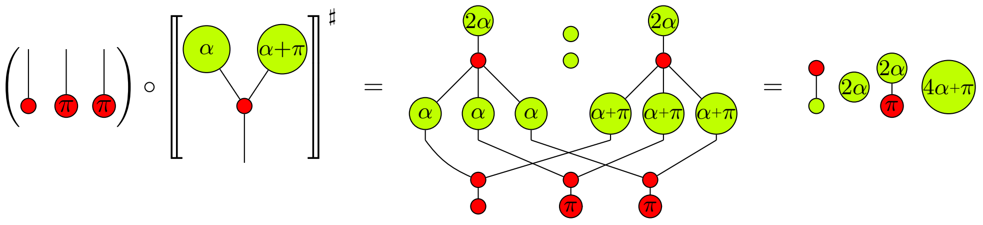
[⇐] Since both diagrams of the supplementarity equation have the same standard interpretation $\frac{1}{\sqrt{2}}\binom{1-e^{2i\alpha}}{0}$, by completeness of the $\frac{\pi}{2}$-fragment of the ZX-calculus, supplementarity can be derived when α is a multiple of $\frac{\pi}{2}$.

[$\Rightarrow$] Let $\alpha \in [0, 2\pi)$, and assume that supplementarity (1) can be derived in the ZX-calculus. Since $[\![.]\!]^\sharp$ is sound, the following equation must be derivable in the ZX-calculus:

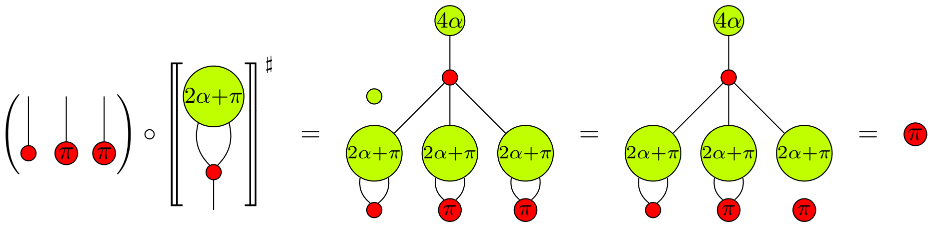


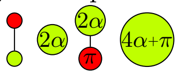
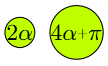
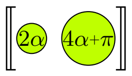
(2)

The LHS diagram is as follows.



The RHS diagram of Eq. 2 is:

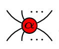
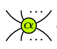


which is obtained first by applying the Hopf law and then thanks to the absorbing property of . Thus, Eq. 2 is equivalent to  $=$ which can be simplified, leading to  $=$. Finally, since $[\![.]\!]$ is sound, it implies  $= [\![$ $]\!]$, thus $(1 + e^{2i\alpha})(1 - e^{4i\alpha}) = 0$ which is equivalent to $\alpha = 0 \bmod \frac{\pi}{2}$. ◀

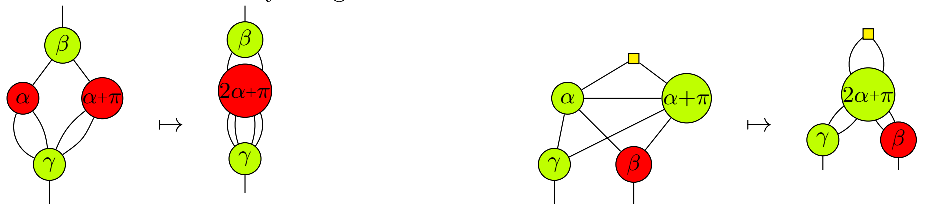
5 Supplementarity as an axiom

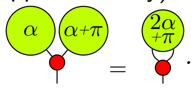
As supplementarity cannot be derive from the other rules of the language, we propose to add this equation as an axiom, a rule of the ZX-calculus. We identify an infinite class of equations that cannot be derived without the supplementarity rule. This class of equations also provides a graphical meaning to the supplementarity equation. Graphically, the supplementarity equation (Eq. 1) can be interpreted as merging two dots in a particular configuration: they are antiphase (i.e. same colour and the difference between the two angles is π); of degree one; and they have the same neighbour. While antiphase is a necessary condition, the other conditions can be relaxed to any "twins" as follows:

▶ **Definition 6** (Antiphase Twins). Two dots u and v in a ZX-diagram are antiphase twins if:

- they have the same colour;
- the difference between their angles is π;
- they have the same neighbourhood: for any other vertex (,  or) w, the number of wires connecting u to w, and v to w are the same.

Notice that antiphase twins might be directly connected or not. Here two examples of antiphase twins and how they merge:

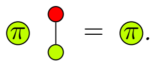


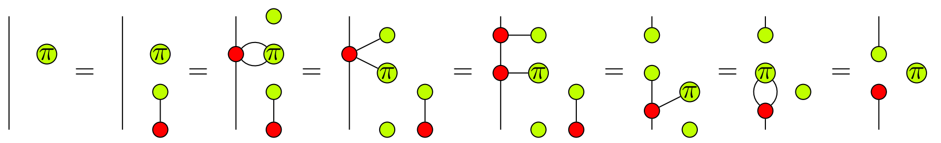
▶ **Theorem 7** (Antiphase Twins and Supplementarity). *In ZX-calculus, any pair of antiphase twins can be merged if and only if* $\forall\alpha$, .

▶ **Corollary 8.** *In the ZX-calculus augmented with the supplementarity rule, any pair of antiphase twins can be merged.*

6 Discussions

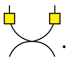
Simplified ZX-calculus. Adding a new rule to the language may lead to a simplification of the other rules of the language. Indeed the (ZO) rule can be replaced by a simpler rule:

▶ **Lemma 9.** *In the ZX-calculus augmented with the supplementarity rule, the (ZO) rule can be replaced by the following (ZO') rule:* .

Proof.  ◀

However, it seems that the language cannot be simplified much. Actually, the two most interesting candidates for simplification are the (EU) rule – which is the only one which is specific to the $\pi/2$ angle and thus may lack generality – and the (K2) rule. Even in the presence of the supplementarity rule, the (EU) rule cannot be derived from the other rules:

▶ **Lemma 10.** *In the ZX-calculus augmented with the supplementary rule, the (EU) rule cannot be derived from the other rules.*

Proof. Let $[\![.]\!]^\flat$ be defined as $[\![.]\!]^\sharp_{2,0}$ (see Remark 1) for all generators but ; and $[\![$ $]\!]^\flat$ = . So intuitively, $[\![.]\!]^\flat$ 'doubles' the diagram, and each 'swaps' the two copies. This interpretation is sound for all rules, including the supplementarity rule, but is not sound for the (EU) rule. ◀

Regarding the (K2) rule, it has been shown recently that (K2) instantiated with an angle multiple of $\frac{\pi}{2}$ can be derived from the other rules [5], without using the supplementarity rule. We leave its necessity for arbitrary angles and in the presence of the supplementarity rule as an open question.

Completeness. Even augmented with the supplementarity rule the ZX-calculus is incomplete in general since the argument of [15] still applies. Indeed, the cornerstone of the incompleteness argument is the soundness of the interpretation which consists in multiplying the angles by -3 ($[\![.]\!]^\sharp_{1,-4}$ according to the notation of Remark 1). This interpretation is also sound with respect to the supplementarity rule, and thus the ZX-calculus is still incomplete, even augmented with the supplementarity rule. However, the second ingredient of the incompleteness result of [15] is based on the Euler decomposition of some unitary transformation which diagrammatic representation involves irrational multiples of π. As a consequence, the completeness of the ZX-calculus augmented with the supplementarity is an open question for any fragment which does not contain rational multiples of π. In particular, the completeness of the $\frac{\pi}{4}$-fragment – i.e. for Clifford+T quantum mechanics – is open.

7 Conclusion

We have proved that the ZX-calculus is not complete, even for Clifford+T quantum mechanics, which corresponds to the $\frac{\pi}{4}$-fragment of the language. We have identified an infinite set of equations that cannot be derived in the language. Moreover, we have shown that a single simple rule, called supplementarity, is sufficient to derive these equations. Supplementarity has been introduced as fundamental structure of multipartite entanglement by Coecke and Edwards. In addition to this physical interpretation, we provide a graphical meaning to the supplementarity rule by means of antiphase twins.

References

1 M. Backens. The ZX-calculus is complete for stabilizer quantum mechanics. New Journal of Physics, 16(9):093021, 2014.

2 M. Backens. The ZX-calculus is complete for the single-qubit Clifford+T group. Electronic Proceedings in Theoretical Computer Science 172, pp. 293–303, 2014.

3 M. Backens. Making the stabilizer ZX-calculus complete for scalars: Electronic Proceedings in Theoretical Computer Science 195, pp. 17–32, 2015.

4 M. Backens, A. N. Duman. A complete graphical calculus for Spekkens' toy bit theory. Foundations of Physics, pp. 1–34, 2015.

5 M. Backens, S. Perdrix, Q. Wang. A Simplified Stabilizer ZX-calculus. arXiv:1602.04744, 2016.

6 P. O. Boykin, T. Mor, M. Pulver, V. P. Roychowdhury, F. Vatan. On Universal and Fault-Tolerant Quantum Computing: A Novel Basis and a New Constructive Proof of Universality for Shor's Basis. Proc. 40th FOCS, pp. 486–494, 1999.

7 B. Coecke, R. Duncan. Interacting quantum observables: Categorical algebra and diagrammatics. New Journal of Physics 13, p. 043016, 2011.

8 B. Coecke, B. Edwards. Three qubit entanglement within graphical Z/X-calculus. Electronic Proceedings in Theoretical Computer Science 52, pp. 22–33, 2010.

9 B. Coecke, D. Pavlovic, J. Vicary. A new description of orthogonal bases. Math. Structures in Comp. Sci., 2011.

10 R. Duncan, K. Dunne. Interacting Frobenius Algebras are Hopf. LICS'16. arXiv:1601.04964, 2016.

11 R. Duncan, S. Perdrix. Rewriting measurement-based quantum computations with generalised flow ICALP 2010, Lecture Notes in Computer Science, vol 6199, pp. 285–296, Springer, 2010.

12 R. Duncan, S. Perdrix. Pivoting makes the ZX-calculus complete for real stabilizers. Electronic Proceedings in Theoretical Computer Science 171, pp. 50–62, 2014.

13 R. Duncan, S. Perdrix. Graph States and the Necessity of Euler Decomposition. Mathematical Theory and Computational Practice, Volume 5635, pp. 167-177, 2009.

14 D. Gottesman. Stabilizer Codes and Quantum Error Correction, Ph.D. Thesis, 1997.

15 C. Schröder de Witt, V. Zamdzhiev. The ZX-calculus is incomplete for quantum mechanics. Electronic Proceedings in Theoretical Computer Science 172, pp. 285–292, 2014.

16 P. Selinger. Finite dimensional Hilbert spaces are complete for dagger compact closed categories. Electronic Notes in Theoretical Computer Science Volume 270, Issue 1, pp. 113–119, 2011.

17 `http://cqm.wikidot.com/zx-completeness`

The Covering Problem: a Unified Approach for Investigating the Expressive Power of Logics*

Thomas Place[1] and Marc Zeitoun[2]

1 **LaBRI, Bordeaux University, France**
thomas.place@labri.fr

2 **LaBRI, Bordeaux University, France**
marc.zeitoun@labri.fr

Abstract

An important endeavor in computer science is to precisely understand the expressive power of logical formalisms over discrete structures, such as words. Naturally, "understanding" is not a mathematical notion. Therefore, this investigation requires a concrete objective to capture such a notion. In the literature, the standard choice for this objective is the *membership problem*, whose aim is to find a procedure deciding whether an input regular language can be defined in the logic under study. This approach was cemented as the "right" one by the seminal work of Schützenberger, McNaughton and Papert on first-order logic and has been in use since then.

However, membership questions are hard: for several important fragments, researchers have failed in this endeavor despite decades of investigation. In view of recent results on one of the most famous open questions, namely the quantifier alternation hierarchy of first-order logic, an explanation may be that membership is too restrictive as a setting. These new results were indeed obtained by considering more general problems than membership, taking advantage of the increased flexibility of the enriched mathematical setting. This opens a promising avenue of research and efforts have been devoted at identifying and solving such problems for natural fragments. However, until now, these problems have been *ad hoc*, most fragments relying on a specific one. A unique new problem replacing membership as the right one is still missing.

The main contribution of this paper is a suitable candidate to play this role: the Covering Problem. We motivate this problem with three arguments. First, it admits an elementary set theoretic formulation, similar to membership. Second, we are able to reexplain or generalize all known results with this problem. Third, we develop a mathematical framework as well as a methodology tailored to the investigation of this problem.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Membership Problem, Separation Problem, Covering Problem, Regular Languages, Logics, Decidable Characterizations

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.77

1 Introduction

One of the most successful applications of the notion of regularity in computer science is the investigation of logics on discrete structures such as words or trees. The story began in the 60s when Büchi [5], Elgot [10] and Trakhtenbrot [36] proved that the *regular languages* of

* This study has been carried out with financial support from the French State, managed by the French National Research Agency (ANR) in the frame of the "Investments for the future" Programme IdEx Bordeaux – CPU (ANR-10-IDEX-03-02).

finite words are those that can be defined in monadic second order logic (MSO). This result has since been exploited to study the expressive power of important fragments of MSO by relying on a decision problem: the *membership problem.* Given a regular language as input, this problem asks if it can be defined by a sentence of the fragment under investigation.

Getting membership algorithms is difficult. In fact, this is still open on finite trees for the most natural fragment of MSO, namely first-order logic (FO). On words however, this question was solved in the 70s by Schützenberger, McNaughton and Papert [30, 14]. This theorem was very influential and has often been revisited [38, 9, 7, 20]. It paved the way to a series of results of the same nature. A famous example is Simon's Theorem [31], which yields an algorithm for the first level of the quantifier alternation hierarchy of FO. Other examples include [4, 13, 39, 33] which consider fragments of FO where the linear order on positions is replaced by the successor relation or [34] which considers the 2-variable fragment of FO. The relevance of this approach is nowadays validated by a wealth of results.

The reason for this success is twofold. First, these results cemented membership as the "right" question: a solution conveys a deep intuition on the investigated logic. In particular, most results include a *generic method for building a canonical sentence* witnessing membership of an input language in the logic. Second, Schützenberger's solution established a suitable framework and a methodology to solve membership problems. This methodology is based on a canonical algebraic abstraction of a regular language which is finite and computable, the *syntactic monoid.* The core of the approach is to translate the semantic question (*is the language definable in the fragment?*) into a purely syntactical, easy question to be tested on the syntactic monoid (*does the syntactic monoid satisfy some equation?*).

Unfortunately, this methodology seems to have reached its limits for the hardest questions. An emblematic example is the *quantifier alternation hierarchy of first-order logic* which classifies sentences according to the number of alternations between $\exists$ and $\forall$ quantifiers in their prenex normal form. A sentence is Σ_i if its prenex normal form has $(i-1)$ alternations and starts with a block of existential quantifiers. A sentence is $\mathcal{B}\Sigma_i$ if it is a boolean combination of Σ_i sentences. Obtaining membership algorithms for all levels in this hierarchy is a major open question and has been given a lot of attention (see [37, 35, 15, 16, 17, 18, 27, 19] for details and a complete bibliography). However, progress on this question has been slow: until recently, only the lowest levels were solved: Σ_1 [3, 21], $\mathcal{B}\Sigma_1$ [31] and Σ_2 [3, 21].

It took years to solve higher levels. Recently, membership algorithms were obtained for Σ_3 [25], $\mathcal{B}\Sigma_2$ [25] and Σ_4 [22]. This was achieved by introducing new ingredients into Schützenberger's methodology: problems that are *more general than membership.* For each result, the strategy is the same: first, a well-chosen more general problem is solved for a lower level in the hierarchy, then, this result is transferred into a membership algorithm for the level under investigation. Let us illustrate what we mean by "more general problem" and present the simplest of them: the *separation problem.* It takes *two* regular languages as input and asks whether there exists a third one which is definable in the logic, contains the first, and is disjoint from the second. Being more general, such problems are also more difficult than membership. However, this generality also makes them more rewarding in the insight they give on the investigated logic. This motivated a series of papers on the separation problem [28, 8, 23, 24, 26] which culminated in the three results above [25, 22]. However, while this avenue of research is very promising, it presently suffers three important flaws:

1. The family of problems that have been considered up until now is a jungle: each particular result relies on a specific ad-hoc problem. For example, the results of [25, 22] rely on three different problems. In fact, even if one is only interested in separation, the actual solution often considers an even more general problem (see [28, 25, 22] for example).

2. Among the problems that have been investigated, separation is the only one that admits a simple and generic set-theoretic definition (which is why it is favored as an example). On the other hand, for all other problems, the definition requires to introduce additional concepts such as semigroups and Ehrenfeucht-Fraïssé games.
3. In contrast to membership solutions, the solutions that have been obtained for these more general problems are non-constructive. For example, most of the separation solutions do not include a generic method for building a separator language when it exists (the algorithms are built around the idea of proving that the two inputs are *not* separable).

Contributions. Our objective in this paper is to address these three issues. Our first contribution is the presentation of a single general problem, the "*covering problem*", which admits a purely set-theoretic definition and generalizes all problems that have already been considered. Furthermore, its definition is modular: the covering problem is designed so that it can easily be generalized to accommodate future needs. Its design is based on an analysis of the methods used to solve membership and separation. In both cases, the algorithms almost always exploit the fact that an input regular language L is *not isolated*: its recognizer defines a *set* of regular languages from which L is built. This set has a structure upon which the algorithms are based. The covering problem takes this observation into account: an input of the problem is directly *any finite set* of regular languages. Given such a set $\mathbf{L}$, the problem asks to compute the "best possible approximation" (called *optimal cover*, hence the name "covering") of this set of languages by languages belonging to the investigated fragment. In particular, the separation problem is just the special case when the input set is of size 2.

The main advantage of the covering problem is that it comes with a generic framework and a generic methodology designed for solving it. This framework is our second contribution. It generalizes the original framework of Schützenberger for membership in a natural way and lifts all its benefits to a more general setting. In particular, we recover *constructiveness*: a solution to the covering problem associated to a particular fragment yields a generic way for building an actual optimal cover of the input set.

Finally, the relevance of our new framework is supported by the fact that we are able to obtain covering algorithms for the fragments that were already known to enjoy a decidable separation problem. In contrast to the previous algorithms, these more general ones are presented within a single unified framework. This is our third contribution. We present actual covering algorithms for four particular logics: first-order logic (FO), two-variables FO (FO^2) and two logics within the quantifier alternation hierarchy of FO ($\mathcal{B}\Sigma_1$ and Σ_2). As explained, the payoff is that we obtain *effective* solutions to the covering problem. Hence, we obtain an effective method for building separators in the weaker separation problem.

Historical note. As observed by Almeida [1], separation is tied to a purely algebraic problem of Henckell and Rhodes (see [11, 12]): computing the "pointlike sets of a given finite semigroup with respect to a variety $\mathbf{V}$". This can probably be lifted to covering. However, there are two main advantages to our approach. First, it is more general: pointlike sets are restricted to classes and inputs that are both more specific than ours. Second, covering admits a simple set theoretic definition that pointlike sets obfuscate with heavy terminology.

Organization. We define the covering problem in Section 2 (for arbitrary input sets of languages, *i.e.*, not necessarily made of regular languages). We present our framework for the particular case of regular inputs in Sections 3 and 4. Four examples of covering algorithms are presented in Section 5. Due to lack of space, proofs are deferred to the journal version.

2 The Covering Problem

In this section, we define the covering problem. For the whole paper, we fix a finite alphabet A and work with finite words over A (*i.e.*, elements of A^*). A *language* is a subset of A^*. Note that we restrict ourselves to words for the sake of simplifying the presentation. However, the covering problem makes sense for *any structure* (such as infinite words or trees).

We focus on two kinds of classes of languages. We say that a class of languages $\mathscr{C}$ is a *lattice* when it contains the empty and universal languages ($\emptyset$ and A^*) and it is closed under finite union and finite intersection: $K, L \in \mathscr{C}$ implies $K \cup L, K \cap L \in \mathscr{C}$. Furthermore, $\mathscr{C}$ is a *boolean algebra* when $\mathscr{C}$ is a lattice that is closed under complement: $L \in \mathscr{C}$ implies $\{w \in A^* \mid w \notin L\} \in \mathscr{C}$. The covering problem then comes into two variants:

- a variant that can be associated to any class of languages that is a lattice. We call this variant the *pointed covering problem.*
- a weaker variant that can be associated to any class of languages that is a boolean algebra. We call it the *covering problem.* While weaker than the first one, this variant enjoys simpler terminology, which makes it our choice when working with boolean algebras.

We now define these two variants. In the definition, we use the separation problem as a foundation to motivate and explain our design choices. As we explained, given a class of languages $\mathscr{C}$, solutions to membership and separation exploit the fact that the recognizer of an input regular language L recognizes a *set* of regular languages from which L is built. The covering problem is based on this observation: its input is any finite set of languages $\mathbf{L}$.

▶ Remark. A "set of languages" is a purely mathematical object. An actual input is a set of recognizing devices for these languages. In particular, it may happen that two such devices recognize the same language. Therefore our inputs are actually finite sets of languages *names* (which may contain "several copies" of the same language). This is harmless: two sets of names for the same underlying set of languages are equivalent for both covering problems.

2.1 The Covering Problem for Boolean Algebras

We begin with the simpler covering problem. Let $\mathscr{C}$ be a boolean algebra[1]. Given a finite set of languages names $\mathbf{L} = \{L_1, \dots, L_n\}$, a *$\mathscr{C}$-cover* of $\mathbf{L}$ is a finite set of languages $\mathbf{K} = \{K_1, \dots, K_m\}$ such that $K_i \in \mathscr{C}$ for all $i \leq n$ and:

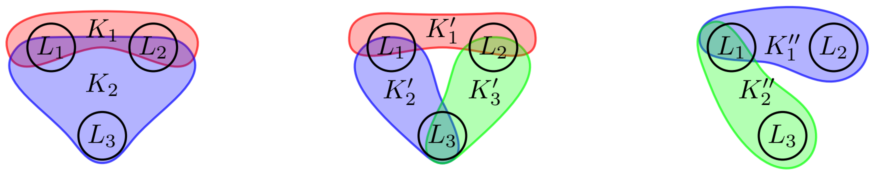
$$L_1 \cup \dots \cup L_n \subseteq K_1 \cup \dots \cup K_m.$$

Note that since $\mathscr{C}$ is a boolean algebra, there always exists a $\mathscr{C}$-cover of $\mathbf{L}$: the singleton $\{A^*\}$. When we have a $\mathscr{C}$-cover $\mathbf{K}$ of $\mathbf{L}$ in hand, our main interest will be to know how good $\mathbf{K}$ is at separating languages in $\mathbf{L}$: what languages in $\mathbf{L}$ are separated by unions of languages in $\mathbf{K}$? What are the "best $\mathscr{C}$-covers" of $\mathbf{L}$ (called optimal $\mathscr{C}$-covers)? This information is captured by a new object that we associate to any cover of $\mathbf{L}$, its imprint on $\mathbf{L}$.

Filterings and Imprints. Imprints are based on filterings. Given a finite set of names $\mathbf{L}$ and a language K, the *filtering of* $\mathbf{L}$ *by* K, measures the "interaction" between $\mathbf{L}$ and K. More precisely, the filtering of $\mathbf{L}$ by K, denoted by $\langle \mathbf{L} | K \rangle$, is defined as the following set:

$$\langle \mathbf{L} | K \rangle = \{L \in \mathbf{L} \mid L \cap K \neq \emptyset\} \subseteq \mathbf{L}\,.$$

[1] The problem actually makes sense for any class that contains the universal language and is closed under intersection. However, we need $\mathscr{C}$ to be a boolean algebra for the connection with separation.



Cover $\mathbf{K} = \{K_1, K_2\}$ $\mathcal{I}[\mathbf{L}](\mathbf{K}) =$ $\downarrow\{\{L_1, L_2, L_3\}\}$ | Cover $\mathbf{K}' = \{K'_1, K'_2, K'_3\}$ $\mathcal{I}[\mathbf{L}](\mathbf{K}') =$ $\downarrow\{\{L_1, L_2\}, \{L_1, L_3\}, \{L_2, L_3\}\}$ | Cover $\mathbf{K}'' = \{K''_1, K''_2\}$ $\mathcal{I}[\mathbf{L}](\mathbf{K}'') =$ $\downarrow\{\{L_1, L_2\}, \{L_1, L_3\}\}$

Figure 1 Some $\mathscr{C}$-covers of $\mathbf{L} = \{L_1, L_2, L_3\}$ and their imprint on $\mathbf{L}$.

▶ Remark. This notion is what makes the problem modular. It can be strengthened to define harder variants of the problem and accommodate future needs.

We may now define imprints. Given a subset E of $2^{\mathbf{L}}$, we write $\downarrow E$ to denote the *downset of* E, *i.e.*, the set $\downarrow E = \{\mathbf{H} \mid \exists \mathbf{H}' \in E \text{ such that } \mathbf{H} \subseteq \mathbf{H}'\}$. If $\mathbf{K}$ is a finite set of languages, the *imprint of* $\mathbf{K}$ *on* $\mathbf{L}$ is the set,

$$\mathcal{I}[\mathbf{L}](\mathbf{K}) \quad = \quad \downarrow\{\langle \mathbf{L}|K\rangle \mid K \in \mathbf{K}\} \subseteq 2^{\mathbf{L}} .$$

Note that we shall mainly use this definition when $\mathbf{K}$ is a $\mathscr{C}$-cover of $\mathbf{L}$. However, in some proofs, it will be convenient to have it for an arbitrary set of languages $\mathbf{K}$. We present examples of imprints when $\mathbf{K}$ is a $\mathscr{C}$-cover of $\mathbf{L}$ in Figure 1.

Let us make a few observations about imprints. An imprint on $\mathbf{L}$ is a subset of $2^{\mathbf{L}}$. Therefore, for a fixed *finite* set $\mathbf{L}$, there are *finitely many* possible imprints on $\mathbf{L}$, even though there are infinitely many finite sets $\mathbf{K}$ of languages. Another simple observation is that all imprints are closed under downset: $\mathcal{I}[\mathbf{L}](\mathbf{K}) = \downarrow\mathcal{I}[\mathbf{L}](\mathbf{K})$. Also notice that if $\mathbf{K}$ is a $\mathscr{C}$-cover of $\mathbf{L}$, its imprint captures separation-related information: if $\{L_1, L_2\} \notin \mathcal{I}[\mathbf{L}](\mathbf{K})$, then L_1 (resp. L_2) can be separated from L_2 (resp. L_1) by a union (in $\mathscr{C}$) of languages in $\mathbf{K}$.

▶ Remark. Imprints capture more than just separation-related information. From the separation point of view, the $\mathscr{C}$-covers $\mathbf{K}$ and $\mathbf{K}'$ of Figure 1 are equivalent: they cannot separate any pair of languages in $\mathbf{L}$. However, their imprints on $\mathbf{L}$ tell us that $\mathbf{K}'$ is "better" as it covers $\mathbf{L}$ without containing a language that intersects all languages in $\mathbf{L}$ at the same time.

Finally, observe that if $\mathbf{K}$ is a $\mathscr{C}$-cover of a finite set $\mathbf{L}$, then its imprint on $\mathbf{L}$ always contains some trivial elements. To any finite set of names $\mathbf{L}$, we associate the following set:

$$\mathcal{I}_{triv}[\mathbf{L}] = \downarrow\{\langle \mathbf{L}|\{w\}\rangle \mid w \in A^*\} = \{\mathbf{H} \subseteq \mathbf{L} \mid \cap_{H \in \mathbf{H}} H \neq \emptyset\} .$$

▶ **Fact 1.** *For any $\mathscr{C}$-cover $\mathbf{K}$ of $\mathbf{L}$, we have $\mathcal{I}_{triv}[\mathbf{L}] \subseteq \mathcal{I}[\mathbf{L}](\mathbf{K})$.*

Optimal $\mathscr{C}$-covers. We now use imprints to define our notion of "best" $\mathscr{C}$-cover of $\mathbf{L}$ which we call *optimal $\mathscr{C}$-covers*. A necessary (but not sufficient) property for a $\mathscr{C}$-cover of $\mathbf{L}$ to be optimal will be that $L_1, L_2 \in \mathbf{L}$ are $\mathscr{C}$-separable if and only if they can be separated by a union of languages in the $\mathscr{C}$-cover. Formally, we say that a $\mathscr{C}$-cover $\mathbf{K}$ of $\mathbf{L}$ is *optimal* when,

$$\mathcal{I}[\mathbf{L}](\mathbf{K}) \subseteq \mathcal{I}[\mathbf{L}](\mathbf{K}') \quad \text{for any } \mathscr{C}\text{-cover } \mathbf{K}' \text{ of } \mathbf{L}.$$

In general, there can be infinitely many optimal $\mathscr{C}$-covers of a given finite set of names $\mathbf{L}$. We now state that for any $\mathbf{L}$, there always exists an optimal $\mathscr{C}$-cover of $\mathbf{L}$. Note that the proof only requires $\mathscr{C}$ to be closed under finite intersection.

▶ **Lemma 2.** *For any finite set of languages names* $\mathbf{L}$*, there exists an optimal* $\mathscr{C}$*-cover of* $\mathbf{L}$*.*

Note that the proof of Lemma 2 is non-constructive. Given a finite set of names $\mathbf{L}$, computing an actual optimal $\mathscr{C}$-cover is a difficult problem in general. In fact, as seen in Theorem 4 below, this is more general than solving $\mathscr{C}$-separability for any pair of languages in $\mathbf{L}$. Before we present this theorem, let us make a key observation about optimal $\mathscr{C}$-covers.

By definition, given a boolean algebra $\mathscr{C}$ and a finite set of names $\mathbf{L}$, all optimal $\mathscr{C}$-covers of $\mathbf{L}$ have the same imprint on $\mathbf{L}$. Hence, this unique imprint on $\mathbf{L}$ is a *canonical* object for $\mathscr{C}$ and $\mathbf{L}$. We call it the *optimal imprint with respect to* $\mathscr{C}$ *on* $\mathbf{L}$ and we denote it by $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$:

$$\mathcal{I}_{\mathscr{C}}[\mathbf{L}] = \mathcal{I}[\mathbf{L}](\mathbf{K}) \quad \text{for any optimal } \mathscr{C}\text{-cover } \mathbf{K} \text{ of } \mathbf{L}.$$

We can now state the *covering problem.* We parametrize it by two classes of languages, a class $\mathscr{D}$ constraining the input, and a boolean algebra $\mathscr{C}$.

▶ **Definition 3.** The Covering problem for $\mathscr{C}$ inside $\mathscr{D}$ is as follows:

INPUT:	A finite set of languages $\mathbf{L} \subseteq \mathscr{D}$.
QUESTION:	Compute $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$.

As expected, we only consider the covering problem when the input class $\mathscr{D}$ is the class of regular languages (in particular we will often simply say "covering problem" for this particular variant). There are two stages when solving the covering problem.

1. *Stage One*: find an algorithm that, given a finite set of regular languages $\mathbf{L}$ as input, computes $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$ (we call such an algorithm a covering algorithm for $\mathscr{C}$). In Theorem 4 below, we prove that this generalizes separation as a *decision problem.*
2. *Stage Two*: find an algorithm that, given a finite set of regular languages $\mathbf{L}$ as input, computes an optimal $\mathscr{C}$-cover of $\mathbf{L}$ (*i.e.*, one whose imprint is $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$). We prove below that this generalizes separation as a *computational problem*: if one has an optimal $\mathscr{C}$-cover of $\mathbf{L}$, one may build a separator in $\mathscr{C}$ for any two separable languages in $\mathbf{L}$.

▶ **Theorem 4.** *Let* $\mathscr{C}$ *be a boolean algebra and let* $\mathbf{L}$ *be a finite set of languages names. Given any two language name' 's* $L_1, L_2 \in \mathbf{L}$*, the following properties are equivalent:*
1. L_1 *and* L_2 *are* $\mathscr{C}$*-separable.*
2. $\{L_1, L_2\} \notin \mathcal{I}_{\mathscr{C}}[\mathbf{L}]$.
3. *For any optimal* $\mathscr{C}$*-cover* $\mathbf{K}$ *of* $\mathbf{L}$*,* L_1 *and* L_2 *are* $\mathscr{C}$*-separable by a union of languages in* $\mathbf{K}$*.*

Theorem 4 will be proved in the journal version of this paper. It entails that 'covering' is a more general problem than 'separation'. It is actually *strictly* more general as $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$ captures more information than which pairs of languages in $\mathbf{L}$ are $\mathscr{C}$-separable.

2.2 The Pointed Covering Problem for Lattices

So far, we connected the separation problem to the more general *covering problem.* Unfortunately, while the definition of the covering problem makes sense for all lattices, the connection with separation stated in Theorem 4 requires the investigated class $\mathscr{C}$ to be a boolean algebra. When $\mathscr{C}$ is not closed under complement, the optimal imprint $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$ does not capture enough information to decide whether two languages in $\mathbf{L}$ are $\mathscr{C}$-separable.

▶ **Example 5.** Let $\mathscr{C}$ be the class of languages which are unions and intersections of languages of the form A^*aA^* for some $a \in A$. Observe that $L_1 = A^*aA^* \cap A^*bA^*$ is $\mathscr{C}$-separable from $L_2 = a^*$ (L_1 belongs to $\mathscr{C}$ and $L_1 \cap L_2 = \emptyset$). However, it can be verified that the optimal imprint with respect to $\mathscr{C}$ on $\{L_1, L_2\}$ is $\mathcal{I}_{\mathscr{C}}[\{L_1, L_2\}] = \{\emptyset, \{L_1\}, \{L_2\}, \{L_1, L_2\}\}$.

We solve this issue with a new problem generalizing separation for any lattice of languages $\mathscr{C}$: the *pointed $\mathscr{C}$-covering problem.* The main idea behind this new problem is to replace the notion of cover of a finite set of languages names $\mathbf{L}$ with a more general one: *pointed covers.* When a class of languages $\mathscr{C}$ is a lattice but not a boolean algebra (*i.e.*, $\mathscr{C}$ is *not closed under complement*), the associated separation problem is asymmetric: given $L_1, L_2 \subseteq A^*$, the two following problems are non-equivalent:

- finding $K_1 \in \mathscr{C}$ such that $L_1 \subseteq K_1$ and $K_1 \cap L_2 = \emptyset$.
- finding $K_2 \in \mathscr{C}$ such that $L_2 \subseteq K_2$ and $K_2 \cap L_1 = \emptyset$.

From the point of view of $\mathscr{C}$-covers, this means that we have to define a notion of "$\mathscr{C}$-cover of $\{L_1, L_2\}$" making a distinction between the languages used to cover L_1 and those used to cover L_2. This is what pointed $\mathscr{C}$-covers are designed for.

Pointed $\mathscr{C}$-covers. Let $\mathbf{L}$ be a finite set of names. An $\mathbf{L}$-*pointed set of languages* is a *finite* set $\mathbb{P} \subseteq \mathbf{L} \times 2^{A^*}$ (i.e., elements of $\mathbb{P}$ are pairs (L, K) where L is a name in $\mathbf{L}$ and K is an arbitrary language). Furthermore, we call *support* of $\mathbb{P}$ the set $\mathbf{K} = \{K \mid (L, K) \in \mathbb{P} \text{ for some } L \in \mathbf{L}\}$. In other words the support of $\mathbb{P}$ is the smallest set of languages such that $\mathbb{P} \subseteq \mathbf{L} \times \mathbf{K}$. Finally, when we have an $\mathbf{L}$-pointed set of languages $\mathbb{P}$ with support $\mathbf{K}$ in hand, for all $L \in \mathbf{L}$, we will denote by $\mathbb{P}(L) \subseteq \mathbf{K}$ the set of all $K \in \mathbf{K}$ such that $(L, K) \in \mathbb{P}$.

We may now define *pointed $\mathscr{C}$-covers.* Let $\mathscr{C}$ be a lattice. Given a finite set of languages names $\mathbf{L}$, a *pointed $\mathscr{C}$-cover* of $\mathbf{L}$ is an $\mathbf{L}$-pointed set of languages $\mathbb{P}$ such that all K in the support of $\mathbb{P}$ belong to $\mathscr{C}$ and for all $L \in \mathbf{L}$,

$$L \subseteq \bigcup_{K \in \mathbb{P}(L)} K \quad (\textit{i.e.}, \mathbb{P}(L) \text{ is a cover of } \{L\})$$

Note that since $\mathscr{C}$ is a lattice, we have $A^* \in \mathscr{C}$. Hence, for all finite sets $\mathbf{L}$, there always exists a pointed $\mathscr{C}$-cover of $\mathbf{L}$: the set $\{(L, A^*) \mid L \in \mathbf{L}\}$.

▶ Remark. Pointed $\mathscr{C}$-covers are more general than $\mathscr{C}$-covers: if $\mathbb{P}$ is a pointed $\mathscr{C}$-cover of $\mathbf{L}$, then the support $\mathbf{K}$ of $\mathbb{P}$ is a $\mathscr{C}$-cover of $\mathbf{L}$. Intuitively, pointed $\mathscr{C}$-covers capture more information: they record for each $L \in \mathbf{L}$ which languages in $\mathbf{K}$ are needed to cover L. We use this additional information to define a finer notion of optimality.

Pointed Imprints. We now generalize imprints to pointed covers with the notion of pointed imprint (also based on the notion of filtering which is unchanged). To define pointed imprints, we first have to generalize the notion of downset to our new setting. If $\mathbf{L}$ is a finite set of language names and $E \subseteq \mathbf{L} \times 2^{\mathbf{L}}$, we denote by $\downarrow E$ the set,

$$\downarrow E = \{(L, \mathbf{H}) \mid \text{there exists } (L, \mathbf{H}') \in E \text{ such that } \mathbf{H} \subseteq \mathbf{H}'\}$$

We may now define pointed imprints. Let $\mathbf{L}$ be a finite set of language names and let $\mathbb{P}$ be an $\mathbf{L}$-pointed set of languages. The *pointed imprint of* $\mathbb{P}$ *on* $\mathbf{L}$ is the set,

$$\mathcal{P}[\mathbf{L}](\mathbb{P}) \quad = \quad \downarrow\{(L, \langle \mathbf{L} | K \rangle) \mid (L, K) \in \mathbb{P}\} \subseteq \mathbf{L} \times 2^{\mathbf{L}}$$

This new notion of pointed imprint has similar properties to those of the original notion of imprint. For a fixed $\mathbf{L}$, any pointed imprint on $\mathbf{L}$ is a subset of $\mathbf{L} \times 2^{\mathbf{L}}$, so there are finitely many pointed imprints on $\mathbf{L}$. Furthermore, pointed imprints are closed under downset.

Moreover, as for imprints, pointed imprints contain some trivial elements. If $\mathbf{L}$ is a finite set of languages, we let

$$\mathcal{P}_{triv}[\mathbf{L}] = \downarrow\{(L, \langle \mathbf{L} | \{w\} \rangle) \mid L \in \mathbf{L} \text{ and } w \in L\} = \{(L, \mathbf{H}) \mid (\cap_{H \in \mathbf{H}} H) \cap L \neq \emptyset\}$$

▶ **Fact 6.** *For any pointed $\mathscr{C}$-cover $\mathbb{P}$ of $\mathbf{L}$, we have $\mathcal{P}_{triv}[\mathbf{L}] \subseteq \mathcal{P}[\mathbf{L}](\mathbb{P})$.*

Optimal Pointed $\mathscr{C}$-Covers. We can now define optimal pointed $\mathscr{C}$-covers. The definition is similar to that of optimal $\mathscr{C}$-covers. We say that a pointed $\mathscr{C}$-cover $\mathbb{P}$ of $\mathbf{L}$ is *optimal* when,

$$\mathcal{P}[\mathbf{L}](\mathbb{P}) \subseteq \mathcal{P}[\mathbf{L}](\mathbb{P}') \quad \text{for any pointed } \mathscr{C}\text{-cover } \mathbb{P}' \text{ of } \mathbf{L}\,.$$

▶ **Lemma 7.** *For any finite set of languages names $\mathbf{L}$, there exists an optimal pointed $\mathscr{C}$-cover of $\mathbf{L}$.*

As Lemma 2, Lemma 7 is based on closure under intersection. We now generalize the notion of optimal imprint. By definition, all optimal pointed $\mathscr{C}$-covers of $\mathbf{L}$ share the same pointed imprint on $\mathbf{L}$. Hence, this unique pointed imprint is a *canonical object* for $\mathscr{C}$ and $\mathbf{L}$. We call it the *optimal pointed imprint with respect to $\mathscr{C}$ on* $\mathbf{L}$ denoted by $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$:

$$\mathcal{P}_{\mathscr{C}}[\mathbf{L}] = \mathcal{P}[\mathbf{L}](\mathbf{K}) \quad \text{for any optimal pointed } \mathscr{C}\text{-cover } \mathbf{K} \text{ of } \mathbf{L}\,.$$

We are now ready to state the pointed covering problem. As before, it is parametrized by a class $\mathscr{D}$ constraining the input, and a lattice $\mathscr{C}$.

▶ **Definition 8.** The Pointed covering problem for $\mathscr{C}$ inside $\mathscr{D}$ is as follows:

INPUT:	A finite set of languages $\mathbf{L} \subseteq \mathscr{D}$.
QUESTION:	Compute $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$.

Similarly to the covering problem, there are two stages when solving the pointed covering problem for a given lattice $\mathscr{C}$. The first one is to find an algorithm that computes $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$ from $\mathbf{L}$ and the second one is to find a generic method for constructing optimal pointed $\mathscr{C}$-covers. We now make the connection with the $\mathscr{C}$-separation problem in the following theorem.

▶ **Theorem 9.** *Let $\mathscr{C}$ be a lattice and let $\mathbf{L}$ be a finite set of languages. Given any two languages $L_1, L_2 \in \mathbf{L}$, the following properties are equivalent:*

1. *L_1 is $\mathscr{C}$-separable from L_2.*
2. *$(L_1, \{L_2\}) \notin \mathcal{P}_{\mathscr{C}}[\mathbf{L}]$.*
3. *For any optimal pointed $\mathscr{C}$-cover $\mathbb{P}$ of $\mathbf{L}$, the language $\bigcup_{K \in \mathbb{P}(L_1)} K$ separates L_1 from L_2.*

Let us make two remarks. The first one is that for any lattice $\mathscr{C}$, pointed covering is more general than covering. The second is that while this relation can be strict (see Example 5), this only happens when the class $\mathscr{C}$ is not closed under complement: if $\mathscr{C}$ is a boolean algebra, then the two problems are equivalent. In other words, when $\mathscr{C}$ is a boolean algebra, there is no point in considering pointed covering: the covering problem (which relies on simpler terminology) suffices. We refer to the journal version of this paper for details.

Now that we have defined both covering problems, the remaining sections are devoted to presenting their benefits. In particular, we present a general methodology for regular inputs in Sections 3 and 4 and use it in Section 5 on specific examples. Note that in contrast to this section which was generic to all types of structures and inputs, the remainder of the paper is specific to words and regular languages: we will rely on the fact that our inputs are sets of regular languages of finite words in our methodology.

3 Tame Sets of Languages

We now present a special class of input sets for the covering problem that we call the class of *tame* sets of languages names. A tame set contains only regular languages and has a specific algebraic structure (which is connected to language concatenation). While not all finite sets of regular languages are tame, we will be able to restrict our algorithms to such inputs without loss of generality. This restriction is central: we rely heavily on the properties of tame inputs in all our algorithms. The typical example of a tame set is the following.

▶ **Example 10.** Given a nondeterministic finite automaton (NFA) $\mathscr{A} = (A, Q, I, F, \delta)$, the set $\{L_{q,r} \mid (q, r) \in Q^2\}$ is tame (where $L_{q,r}$ is a name for the language $\{w \mid q \xrightarrow{w} r\}$).

3.1 Definition

A finite set of languages names is said to be *tame* if it can be given a *partial semigroup structure*. Let us first define *partial semigroups*. A partial semigroup is a set S equipped with a partial multiplication (*i.e.*, st may not be defined for all $s, t \in S$) such that for all $r, s, t \in S$, if rs and st are both defined, then $(rs)t$ and $r(st)$ are defined and equal.

We may now define tame sets. Let $\mathbf{L}$ be a finite set of languages names. A *tame multiplication* for $\mathbf{L}$ is a partial semigroup multiplication "$\odot$" (we use this notation to avoid confusion with language concatenation) that satisfies the following properties:

1. for all $L, L' \in \mathbf{L}$, if $L \odot L'$ is defined then $LL' \subseteq L \odot L'$.
2. for all $H \in \mathbf{L}$ and all words $w \in H$, if w may be decomposed as $w = uu'$, then there exist $L, L' \in \mathbf{L}$ such that $u \in L$, $u' \in L'$ and $H = L \odot L'$.

We say that a finite set of languages names $\mathbf{L}$ is *tame* if it can be equipped with a *tame multiplication*. Note that when working with tame sets, we will implicitly assume that we have a tame multiplication "$\odot$" for this set. Furthermore, since $\mathbf{L}$ is a finite partial semigroup, it is known that there exists an integer $\omega(\mathbf{L})$ (denoted by ω when $\mathbf{L}$ is understood) such that if $L \odot L$ is defined, then L^ω is defined and idempotent (*i.e.*, $L^\omega \odot L^\omega = L^\omega$).

An important observation is that tame sets of languages names may only contain *regular languages*, as stated in the following lemma (proved in the journal version).

▶ **Lemma 11.** *Any language in a tame set of languages is regular.*

Unfortunately, the converse of Lemma 11 is not true: there are finite sets of regular languages that are not tame. For example, the set $\mathbf{L} = \{\{ab\}\}$ fails Condition 2. However, this issue is easily solved with the following proposition.

▶ **Proposition 12.** *Let* $\mathbf{H} = \{H_1, \dots, H_n\}$ *be a finite set of languages given by* n NFAs $\mathscr{A}_1, \dots, \mathscr{A}_n$. *There exists a tame set of languages names* $\mathbf{L}$ *such that for any lattice* $\mathscr{C}$,

- $\mathcal{I}_{\mathscr{C}}[\mathbf{H}]$ *(resp.* $\mathcal{P}_{\mathscr{C}}[\mathbf{H}]$*) can be computed from* $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$ *(resp.* $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$*).*
- *any optimal (pointed)* $\mathscr{C}$*-cover of* $\mathbf{L}$ *is an optimal (pointed)* $\mathscr{C}$*-cover of* $\mathbf{H}$*.*
- $\mathbf{L}$ *and its tame multiplication can be computed from* $\mathscr{A}_1, \dots, \mathscr{A}_n$ *in polynomial time and has size* $|\mathscr{A}_1|^2 + \cdots + |\mathscr{A}_n|^2$ *(where* $|\mathscr{A}_i|$ *stands for the number of states of* $\mathscr{A}_i$*).*

Proposition 12 is proved in the journal version (the construction is based on Example 10). From now on, we will assume that our inputs are tame. We finish the section by explaining the benefits of considering tame inputs in the covering and pointed covering problems.

3.2 Tame Sets of Languages and the Covering Problems

As explained, we will restrict our inputs to tame sets. We now have to explain the benefits of such a restriction. In order to get these benefits, we need the investigated class $\mathscr{C}$ to satisfy a new property in addition to being a boolean algebra or a lattice. The *left quotient* of a language L by a word w is the language $w^{-1}L = \{u \in A^* \mid wu \in L\}$. The *right quotient* Lw^{-1} is defined symmetrically. A class of languages is a *quotienting boolean algebra* if it is a boolean algebra of regular languages closed under left and right quotient. A *quotienting lattice* is a lattice of regular languages closed under left and right quotients.

When $\mathbf{L}$ is tame, the partial semigroup multiplication $\odot$ over $\mathbf{L}$ can be extended as a semigroup multiplication over $2^{\mathbf{L}}$: $\mathbf{S} \odot \mathbf{R} = \{S \odot R \mid S \in \mathbf{S},\ R \in \mathbf{R} \text{ and } S \odot R \text{ is defined}\}$. Hence, $2^{\mathbf{L}}$ is a semigroup and $\mathbf{L} \times 2^{\mathbf{L}}$ a partial semigroup. It turns out that when $\mathscr{C}$ is a quotienting lattice these structures are transferred to $\mathcal{I}_{\mathscr{C}}[\mathbf{L}] \subseteq 2^{\mathbf{L}}$ and $\mathcal{P}_{\mathscr{C}}[\mathbf{L}] \subseteq \mathbf{L} \times 2^{\mathbf{L}}$.

▶ **Lemma 13.** *Let $\mathscr{C}$ be a quotienting lattice and let $\mathbf{L}$ be a tame set of languages. Then the two following properties holds:*

1. *$\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$ is closed under multiplication: for all $(L_1, \mathbf{L}_1)$, $(L_2, \mathbf{L}_2)$ in $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$, if $L_1 \odot L_2$ is defined, then $(L_1 \odot L_2, \mathbf{L}_1 \odot \mathbf{L}_2) \in \mathcal{P}_{\mathscr{C}}[\mathbf{L}]$.*
2. *$\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$ is closed under multiplication: for all $\mathbf{L}_1$ and $\mathbf{L}_2$ in $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$, $\mathbf{L}_1 \odot \mathbf{L}_2 \in \mathcal{I}_{\mathscr{C}}[\mathbf{L}]$.*

Lemma 13 will be proved in the full version. Let us explain why it is crucial. We do it in the setting of the covering problem, which is simpler. We start with the following statement.

▶ **Lemma 14.** *Let $\mathbf{L}$ be a tame set of languages and let K_1, K_2 be two languages, then $\langle \mathbf{L} | K_1 \rangle \odot \langle \mathbf{L} | K_2 \rangle = \langle \mathbf{L} | K_1 K_2 \rangle$.*

Let $\mathscr{C}$ be a boolean algebra and $\mathbf{L}$ be a finite set of names. A natural method for building an optimal $\mathscr{C}$-cover $\mathbf{K}$ of $\mathbf{L}$ is to start from $\mathbf{K} = \mathcal{I}_{triv}[\mathbf{L}]$ and to add new languages K in $\mathscr{C}$ to $\mathbf{K}$ until $\mathbf{K}$ covers $\mathbf{L}$. By definition of imprints, for $\mathbf{K}$ to be optimal, we need all such candidate languages K to satisfy $\langle \mathbf{L} | K \rangle \in \mathcal{I}_{\mathscr{C}}[\mathbf{L}]$. It follows from Lemma 13 and Lemma 14 that when $\mathscr{C}$ is a quotienting boolean algebra and $\mathbf{L}$ is tame, these K may be built with concatenation: if we already have K_1 and K_2 such that $\langle \mathbf{L} | K_1 \rangle, \langle \mathbf{L} | K_2 \rangle \in \mathcal{I}_{\mathscr{C}}[\mathbf{L}]$, then we may add $K_1 K_2$ as well since by Lemmas 13 and 14, $\langle \mathbf{L} | K_1 K_2 \rangle = \langle \mathbf{L} | K_1 \rangle \odot \langle \mathbf{L} | K_2 \rangle \in \mathcal{I}_{\mathscr{C}}[\mathbf{L}]$.

This is central for classes of languages defined through logic (such as first-order logic). Indeed, concatenation is a fundamental process for building new languages in such classes.

4 General Approach

In this section, we present a natural methodology for attempting to solve the covering or pointed covering problem for a particular input class $\mathscr{C}$. This is the methodology that we use for all examples of Section 5.

Let $\mathscr{C}$ be a quotienting boolean algebra or a quotienting lattice. Recall that since we restrict ourselves to tame sets, the two objectives of the covering (resp. pointed covering) problem are as follows. Given as input a tame set $\mathbf{L}$,

1. we want an algorithm that computes $\mathcal{I}_{\mathscr{C}}[\mathbf{L}]$ (resp. $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$).
2. we want an algorithm that computes optimal $\mathscr{C}$-covers (resp. pointed $\mathscr{C}$-covers).

We now detail our methodology for the pointed covering problem (the case of the weaker covering problem is similar, see Section 5). This methodology consists in three steps.

Step 1: Presentation of the Pointed Covering Algorithm. The first step presents a solution to stage one: an algorithm that takes as input a tame set $\mathbf{L}$ and computes $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$. This step only presents the algorithm: the second and third steps are devoted to its proof.

A key point is that pointed covering algorithms are designed as *lowest fixpoint algorithms*. Since $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$ is a pointed imprint on $\mathbf{L}$, we have $\mathcal{P}_{triv}[\mathbf{L}] \subseteq \mathcal{P}_{\mathscr{C}}[\mathbf{L}]$ (Fact 6). All our algorithms start from $\mathcal{P}_{triv}[\mathbf{L}]$, and then add new elements using finitely many operations until a fixpoint is reached. Among these operations, some are specific to the particular quotienting lattice $\mathscr{C}$ that we consider, and some are *generic* to all quotienting lattices. In particular, the set of operations that we use will always include downset and multiplication (see Lemma 13). To sum up, our algorithms compute $\mathcal{P}_{\mathscr{C}}[\mathbf{L}]$ as a the smallest set $Sat_{\mathscr{C}}(\mathbf{L}) \subseteq \mathbf{L} \times 2^{\mathbf{L}}$ (*Sat* means 'saturation'), containing $\mathcal{P}_{triv}[\mathbf{L}]$ and closed under the following operations:

1. Downset: $Sat_{\mathscr{C}}(\mathbf{L}) = \downarrow Sat_{\mathscr{C}}(\mathbf{L})$.
2. Multiplication: if $(L, \mathbf{H}), (L', \mathbf{H}') \in Sat_{\mathscr{C}}(\mathbf{L})$, then $(L \odot L', \mathbf{H} \odot \mathbf{H}') \in Sat_{\mathscr{C}}(\mathbf{L})$ (if defined).
3. $\cdots$ (additional operation(s) specific to $\mathscr{C}$).

Step 2: Soundness. The second step is devoted to proving that the covering algorithm of Step 1 is sound, *i.e.*, that $Sat_{\mathscr{C}}(\mathbf{L}) \subseteq \mathcal{P}_{\mathscr{C}}[\mathbf{L}]$: for any pointed $\mathscr{C}$-cover $\mathbb{P}$ of $\mathbf{L}$, $Sat_{\mathscr{C}}(\mathbf{L}) \subseteq \mathcal{P}[\mathbf{L}](\mathbb{P})$. This is the "easy" direction and it involves Ehrenfeucht-Fraïssé arguments.

Step 3: Completeness. The third step is devoted to proving that the covering algorithm of Step 1 is complete, *i.e.*, that $\mathcal{P}_{\mathscr{C}}[\mathbf{L}] \subseteq Sat_{\mathscr{C}}(\mathbf{L})$. While usually difficult, this proof is of particular interest as it yields a solution to second stage of the pointed covering problem as a byproduct: an algorithm that computes optimal pointed $\mathscr{C}$-covers.

The proof of this step should be presented as a generic construction for building an actual pointed $\mathscr{C}$-cover $\mathbb{P}$ of $\mathbf{L}$ whose imprint on $\mathbf{L}$ is included in $Sat_{\mathscr{C}}(\mathbf{L})$. This proves that $\mathcal{P}_{\mathscr{C}}[\mathbf{L}] \subseteq \mathcal{P}[\mathbf{L}](\mathbb{P}) \subseteq Sat_{\mathscr{C}}(\mathbf{L})$, and therefore completeness. However, by combining this with the knowledge that the algorithm is also sound (this is proved in Step 2), we obtain that $\mathcal{P}_{\mathscr{C}}[\mathbf{L}] = \mathcal{P}[\mathbf{L}](\mathbb{P})$. In other words the proof builds an *optimal pointed $\mathscr{C}$-cover* $\mathbb{P}$ of $\mathbf{L}$.

5 Examples of Covering Algorithms

We now present examples of covering algorithms for several classical logical fragments, all based on first-order logic on words. Let us first briefly recall the definition of first-order logic over finite words. A word is viewed as logical structure made of a sequence of positions labeled over A. In first-order logic over words (FO), for each $a \in A$, one is allowed to use a unary predicate "$a(x)$" which selects positions x labeled with an a, as well as a binary predicate "$<$" for the linear order. A language L is said to be *first-order definable* if there is an FO sentence φ such that $L = \{w \mid w \models \varphi\}$. Also denote by FO the class of all first-order definable languages. We present algorithms for FO itself and its fragments $\mathcal{B}\Sigma_1$, FO^2, Σ_2.

Note that we only present Step 1 of our methodology in the main text, *i.e.*, algorithms without their proofs. An important remark is that these proofs are all difficult: while we have a generic template, proving a covering algorithm always requires arguments specific to the investigated class. We present proofs for $\mathcal{B}\Sigma_1$, FO^2 and Σ_2 in the full version of this paper. The proof for FO is omitted as it is close to proof of [28] (which is based on a prototype of the present framework). On the other hand, the algorithms and proofs for $\mathcal{B}\Sigma_1$, FO^2 and Σ_2 are new.

First-Order Logic: FO. The first algorithm that we present is for FO itself, which is among the most famous classes of regular languages in the literature. The decidability of the membership problem for FO was proved by Schützenberger, McNaughton and Papert [30, 14] and the result is among those that started this line of research. Separation was later proved to be decidable as well [11, 12, 28]. As explained the covering algorithm is a generalization of that of [28] (which is based on a prototype of this framework). As FO is known to be a quotienting boolean algebra, we use the covering problem.

▶ **Theorem 15.** *Let* $\mathbf{L}$ *be a tame set of languages. Then* $\mathcal{I}_{\mathrm{FO}}[\mathbf{L}]$ *is the smallest subset of* $2^{\mathbf{L}}$ *containing* $\mathcal{I}_{triv}[\mathbf{L}]$*, closed under downset, multiplication and such that for all* $\mathbf{S} \in \mathcal{I}_{\mathrm{FO}}[\mathbf{L}]$*, we have* $\mathbf{S}^{\omega} \cup \mathbf{S}^{\omega+1} \in \mathcal{I}_{\mathrm{FO}}[\mathbf{L}]$*.*

Boolean Combinations of Σ_1: $\mathcal{B}\Sigma_1$. The next class that we use as an example is $\mathcal{B}\Sigma_1$, which is the restriction of FO to sentences that are boolean combinations of Σ_1 sentences. A sentence is Σ_1 if its prenex normal form uses only existential quantifiers. The class $\mathcal{B}\Sigma_1$ is famous in the literature. the decidability of $\mathcal{B}\Sigma_1$-membership was proved by Simon [31]. $\mathcal{B}\Sigma_1$-separation is also known to be decidable [8, 23]. As $\mathcal{B}\Sigma_1$ is known to be a quotienting boolean algebra, we use the covering problem. Given a word $w \in A^*$, we denote by $\mathbf{alph}(w)$ the set of letters occurring in w, i.e. the smallest subset of B of A such that $w \in B^*$.

▶ **Theorem 16.** *Let* $\mathbf{L}$ *be a tame set of languages.* $\mathcal{I}_{\mathcal{B}\Sigma_1}[\mathbf{L}]$ *is the smallest subset of* $2^{\mathbf{L}}$ *containing* $\mathcal{I}_{triv}[\mathbf{L}]$*, closed under downset, multiplication and such that for all* $B \subseteq A$*, if* $\mathbf{H} = \{L \in \mathbf{L} \mid \exists w \in L,\ s.t.\ \mathbf{alph}(w) = B\}$*, then* $\mathbf{H}^{\omega} \in \mathcal{I}_{\mathcal{B}\Sigma_1}[\mathbf{L}]$*.*

Two-variable First-Order Logic: FO^2. The logic FO^2 is the restriction of FO to sentences that use at most two distinct variables (which may be reused). That the associated membership problem is decidable is due to Thérien and Wilke [34]. The separation problem was proved to be decidable in [23]. As FO^2 is known to be a quotienting boolean algebra, we use the covering problem. Our algorithm requires the input to satisfy a new condition in addition to being tame: *alphabet compatibility* (this may be assumed without loss of generality, as will be shown in the full version). A set $\mathbf{L}$ is said to be *alphabet compatible* if for all languages $L \in \mathbf{L}$, there exists a unique $B \subseteq A$ such that for any $w \in L$, $\mathbf{alph}(w) = B$. Note that when $\mathbf{L}$ is alphabet compatible, then $\mathbf{alph}(L)$ is well-defined for all $L \in \mathbf{L}$ as this unique alphabet.

▶ **Theorem 17.** *Let* $\mathbf{L}$ *be a tame and alphabet compatible set of languages.* $\mathcal{I}_{\mathrm{FO}^2}[\mathbf{L}]$ *is the smallest subset of* $2^{\mathbf{L}}$ *containing* $\mathcal{I}_{triv}[\mathbf{L}]$*, closed under downset, multiplication and such that for all* $B \subseteq A$ *and* $\mathbf{S}, \mathbf{T} \in \mathcal{I}_{\mathrm{FO}^2}[\mathbf{L}]$ *containing* S, T *with* $\mathbf{alph}(S) = \mathbf{alph}(T) = B$*,*

$$\mathbf{S}^{\omega} \odot \langle \mathbf{L} | B^* \rangle \odot \mathbf{T}^{\omega} \in \mathcal{I}_{\mathrm{FO}^2}[\mathbf{L}].$$

One Quantifier Alternation: Σ_2. Our third example is Σ_2, which is the restriction of FO to sentences whose prenex normal form have a quantifier prefix of the form '$\exists^*\forall^*$'. It was proved that Σ_2-membership is decidable in [3, 21] and the same was proved for separation in [25]. As Σ_2 is a quotienting lattice but not a boolean algebra, we use the pointed covering problem. Our algorithm requires the input to be tame and alphabet compatible.

▶ **Theorem 18.** *Let* $\mathbf{L}$ *be a tame and alphabet compatible set of languages* $\mathcal{P}_{\Sigma_2}[\mathbf{L}]$ *is the smallest subset of* $\mathbf{L} \times 2^{\mathbf{L}}$ *containing* $\mathcal{P}_{triv}[\mathbf{L}]$*, closed under downset, multiplication and such that for any* $B \subseteq A$*, and* $(S, \mathbf{S}) \in \mathcal{P}_{\Sigma_2}[\mathbf{L}]$ *satisfying* $\mathbf{alph}(S) = B$ *and* $S \odot S$ *is defined,*

$$(S^{\omega}, \mathbf{S}^{\omega} \odot \langle \mathbf{L} | B^* \rangle \odot \mathbf{S}^{\omega}) \in \mathcal{P}_{\Sigma_2}[\mathbf{L}].$$

6 Conclusion

We introduced the covering and pointed covering problems which are designed to investigate quotienting boolean algebras and quotienting lattices respectively. We also presented a methodology outlining how these problems should be approached. Furthermore, we presented four examples of algorithms for the instances associated to FO, $\mathcal{B}\Sigma_1$, FO^2 and Σ_2.

It is worth noting that while our examples include the most significant logics for which separation is known to be decidable, an important one is missing: Σ_3. This is not surprising as the algorithm of [22] considers an *ad hoc* problem which is associated to two logics at the same time: Σ_2 and Σ_3. However, it is possible to generalize this result as well within our framework: this is where the modularity of our problems comes into play. Using a stronger notion of filtering, one can reformulate and generalize the problem of [22] as an instance of the pointed covering problem (we leave the presentation of this instance for further work).

Our results raise several questions. The most natural is to apply our framework to classes for which no membership or separation algorithm is known yet. Another one is related to the classical membership algorithms. These algorithms are usually stated as equations on the syntactic monoid of the language which share similarities with fixpoint operations of our (pointed) covering algorithms. An interesting question would be to find a criterion under which membership equations can be lifted as a fixpoint operation for the covering problem.

References

1 Jorge Almeida. Some algorithmic problems for pseudovarieties. *Publicationes Mathematicae Debrecen*, 54:531–552, 1999.

2 Jorge Almeida, José C. Costa, and Marc Zeitoun. Closures of regular languages for profinite topologies. *Semigroup Forum*, 89(1):20–40, 2014.

3 Mustapha Arfi. Polynomial operations on rational languages. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS'87)*, pages 198–206, 1987.

4 Janusz A. Brzozowski and Imre Simon. Characterizations of locally testable events. *Discrete Mathematics*, 4(3):243–271, 1973.

5 Julius R. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

6 Julius R. Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of ScienceProceeding of the 1960 International Congress*, volume 44, pages 1–11. Elsevier, 1966.

7 Thomas Colcombet. Green's relations and their use in automata theory. In *Proceedings of Language and Automata Theory and Applications, 5th International Conference (LATA'11)*, pages 1–21, 2011.

8 Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, pages 150–161, 2013.

9 Volker Diekert and Paul Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

10 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98(1):21–51, 1961.

11 Karsten Henckell. Pointlike sets: the finest aperiodic cover of a finite semigroup. *J. Pure Appl. Algebra*, 55:85–126, 1988.

12 Karsten Henckell, John Rhodes, and Benjamin Steinberg. Aperiodic pointlikes and beyond. *Internat. J. Algebra Comput.*, 20(2):287–305, 2010.

13 Robert McNaughton. Algebraic decision procedures for local testability. *Mathematical Systems Theory*, 8(1):60–76, 1974.

14 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata*. The MIT Press, 1971.

15 Jean-Éric Pin. Finite semigroups and recognizable languages: An introduction. In *Semigroups, Formal Languages and Groups*, pages 1–32. Springer-Verlag, 1995.

16 Jean-Éric Pin. Syntactic semigroups. In *Handbook of Formal Languages*, pages 679–746. Springer-Verlag, 1997.

17 Jean-Éric Pin. Bridges for concatenation hierarchies. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming, ICALP'98*, Lecture Notes in Computer Science, pages 431–442, Berlin, Heidelberg, 1998. Springer-Verlag.

18 Jean-Éric Pin. Theme and variations on the concatenation product. In *Proceedings of the 4th International Conference on Algebraic Informatics, CAI'11*, Lecture Notes in Computer Science, pages 44–64, Berlin, Heidelberg, 2011. Springer-Verlag.

19 Jean-Éric Pin. The dot-depth hierarchy, 45 years later. In *WSPC Proceedings*, 2016. To appear.

20 Jean-Éric Pin. Mathematical foundations of automata theory. In preparation, 2016. URL: `https://www.irif.univ-paris-diderot.fr/~jep/MPRI/MPRI.html`.

21 Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997.

22 Thomas Place. Separating regular languages with two quantifiers alternations. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'15)*, pages 202–213, 2015.

23 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science*, MFCS'13, pages 729–740, 2013.

24 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. *Logical Methods in Computer Science*, 10(3), 2014.

25 Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In *Automata, Languages, and Programming - 41st International Colloquium (ICALP'14)*, pages 342–353, 2014.

26 Thomas Place and Marc Zeitoun. Separation and the successor relation. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS'15)*, pages 662–675, 2015.

27 Thomas Place and Marc Zeitoun. The tale of the quantifier alternation hierarchy of first-order logic over words. *SIGLOG news*, 2(3):4–17, 2015.

28 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. *Logical Methods in Computer Science*, 12(1), 2016.

29 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Bull. Amer. Math. Soc.*, 74(5):1025–1029, 09 1968.

30 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

31 Imre Simon. Piecewise testable events. In *Proc. of the 2nd GI Conf. on Automata Theory and Formal Languages*, pages 214–222, 1975.

32 James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

33 Denis Thérien and Alex Weiss. Graph congruences and wreath products. *J. Pure Appl. Algebra*, 36:205–215, 1985.

34 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC'98)*, pages 234–240, 1998.

35 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of formal languages*. Springer, 1997.

36 Boris A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 149:326–329, 1961. In Russian.

37 Pascal Weil. Concatenation product: a survey. In *Formal Properties of Finite Automata and Applications*, volume 386 of *Lecture Notes in Computer Science*, pages 120–137. Springer-Verlag, Berlin, Heidelberg, 1989.

38 Thomas Wilke. Classifying discrete temporal properties. In *Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science, STACS'99*, Lecture Notes in Computer Science, pages 32–46, Berlin, Heidelberg, 1999. Springer-Verlag.

39 Yechezkel Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972.

On the Complexity of Branching Games with Regular Conditions[*]

Marcin Przybyłko[1] and Michał Skrzypczak[2]

1 **University of Warsaw, University of New Caledonia**
`M.Przybylko@mimuw.edu.pl`

2 **University of Warsaw**
`M.Skrzypczak@mimuw.edu.pl`

Abstract

Infinite duration games with regular conditions are one of the crucial tools in the areas of verification and synthesis. In this paper we consider a branching variant of such games – the game contains branching vertices that split the play into two independent sub-games. Thus, a play has the form of an infinite tree. The winner of the play is determined by a winning condition specified as a set of infinite trees. Games of this kind were used by Mio to provide a game semantics for the probabilistic μ-calculus. He used winning conditions defined in terms of parity games on trees. In this work we consider a more general class of winning conditions, namely those definable by finite automata on infinite trees. Our games can be seen as a branching-time variant of the stochastic games on graphs.

We address the question of determinacy of a branching game and the problem of computing the optimal game value for each of the players. We consider both the stochastic and non-stochastic variants of the games. The questions under consideration are parametrised by the family of strategies we allow: either mixed, behavioural, or pure.

We prove that in general, branching games are not determined under mixed strategies. This holds even for topologically simple winning conditions (differences of two open sets) and non-stochastic arenas. Nevertheless, we show that the games become determined under mixed strategies if we restrict the winning conditions to open sets of trees. We prove that the problem of comparing the game value to a rational threshold is undecidable for branching games with regular conditions in all non-trivial stochastic cases. In the non-stochastic cases we provide exact bounds on the complexity of the problem. The only case left open is the 0-player stochastic case, i.e. the problem of computing the measure of a given regular language of infinite trees.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases stochastic games, meta-parity games, infinite trees, regular languages, parity automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.78

1 Introduction

Since the seminal works of Büchi and Landweber [3], and of McNaughton [17], the infinite duration games are widely used to model interaction between a system and an environment. One of the fundamental questions about such games is the question of determinacy – does always one of the players has a winning strategy? In a more general case of valued zero-sum

[*] The authors were supported by the Polish National Science Centre grant no. 2014-13/B/ST6/03595.

games, determinacy amounts to the equality

$$\sup_{\sigma} \inf_{\pi} \ val(\sigma, \pi) = \inf_{\pi} \sup_{\sigma} \ val(\sigma, \pi), \tag{1}$$

where σ and π range over strategies of the respective players. It is often crucial to provide a specific information about the strategies that are enough to win a given game. Büchi and Landweber proved that if the winning condition of a game is a regular language of infinite words then the game is determined under finite memory strategies. Further results have established more precise bounds for the amount of memory needed [9, 10]. Also, the stochastic variant of the question was considered [7].

In this work we study a branching variant of stochastic games on graphs – a variant called *branching games* (also known as *tree games*, c.f. [20, Chapter 4]), played on *branching boards*. A play of a branching game consists of a number of threads, each thread develops independently. When a thread reaches a vertex marked as *branching*, it is split into two separate threads. Thus, a play of a branching game has the shape of an infinite tree. The winner of the play is determined by a winning condition specified as a regular set of infinite trees. Since the choices made by players in separate threads are independent, branching games are not games of perfect information, nor of perfect recall in the meaning of [2]. Games of this kind were used by Mio [21] to provide a game semantics for the probabilistic μ-calculus, with the *meta-parity* winning condition defined in terms of parity games on trees. The author called them *meta-parity games* and established their pure determinacy subject to some set-theoretic assumptions, that were later eliminated [13]. This result is interesting, because, as the author notices in conclusion, determinacy results for imperfect information games are not frequent in game theory.

In this article we address the question of determinacy of branching games and the problem of computing the game value for a more general class of winning conditions than the meta-parity conditions studied by Mio, namely those definable by finite automata on infinite trees. We believe that this extension is motivated by the role tree automata play in verification theory. Recall that these automata, introduced by Rabin in his proof of decidability of the Monadic Second-Order theory of the full k-ary tree [23], constitute a general formalism subsuming most of temporal logics of programs. The first step was made by the first author who extended the results of Mio [21] to winning conditions given by game automata [22].

We consider both the stochastic and non-stochastic variants of the games. The questions under consideration are parametrised by the family of strategies we allow, either mixed, behavioural, or pure. The goal of this work is to provide answers to two questions:

- When a branching game is determined in the sense of (1)?
- When the optimal value for a given player can be effectively computed?

Both questions can be asked for stochastic and non-stochastic variants of games, which usually yields different answers, see e.g. [7]; and for different sets of allowed strategies. The distinction between the sets of pure, behavioural, and mixed strategies can significantly alter the techniques and expected outcomes, see e.g. [5].

The answers we provide create an almost complete picture from the point of view of topological complexity of sets:

- non-stochastic branching games are not determined under pure nor behavioural strategies even for winning conditions that are topologically both closed and open,
- non-stochastic branching games are not determined under mixed strategies for winning conditions that are a difference of two open sets,
- non-stochastic branching games are determined under mixed strategies for winning conditions that are open (equivalently, closed) sets,

- the problem of comparing the value of a branching game to a rational number is undecidable in all the non-trivial stochastic cases,
- in the non-stochastic case, when we ask about the existence of a pure winning strategy, the problem is decidable and we provide precise bounds on its complexity.

The only remaining question is whether the value of a 0-player stochastic branching game can be effectively computed. This is equivalent to asking about computability of the measure of a given regular language of infinite trees.

Although the results of this paper show intractability of branching games, it is still possible that they are determined for a reasonable class of winning conditions. These ideas are discussed in Conclusions.

1.1 Related work

It is known that games with arbitrary pay-off functions are not determined. The celebrated result of Martin [15] states that *Gale-Steward games* with Borel payoffs are determined under pure strategies. His later result establishes an analogous result for the so-called *Blackwell games* and mixed strategies (cf. [16]). In this work we show that branching games are determined only for topologically simplest winning conditions – open or closed sets; even allowing a difference of two open sets leads to indeterminacy.

Since the branching games are not games of *perfect recall* the Kuhn's theorem (cf. e.g. [2]) does not hold; and the behavioural strategies are weaker than the mixed strategies. An example of such a situation was provided by Mio in [20, Chapter 4]. Therefore, there are three variants of the question of determinacy: pure, behavioural, and mixed determinacy.

The concept of branching games is a natural extension of the *meta-parity games* introduced by Mio [21] to provide a game semantics for the probabilistic μ-calculus. The first author proved in [22] that non-stochastic branching games with winning conditions given by the so-called *game automata* are determined under pure strategies. In this work we study determinacy of branching games from the perspective of the topological complexity of the winning condition.

Recently Asarin et al. considered the so-called *entropy games*, cf. [1], which can be easily embedded in our framework. In the authors' own words, an entropy game is played on a finite arena by two-and-a-half players: Despot, Tribune, and the non-deterministic People. The pay-off function is the entropy of the language formed by paths of the resulting tree. The authors of [1] prove that the entropy games are determined under pure strategies and can be solved in $NP \cap coNP$, extending the class of objectives for which branching games are determined.

The question of computing the coin-flipping measure of a given regular languages of infinite trees is one of the crucial open problems about probabilistic logics on infinite trees. Chen, Dräger, and Kiefer proved in [8] that the problem is decidable for regular languages recognisable by deterministic automata. The result was later strengthened by Michalewski and Mio [19] to the so-called game automata. The question of computing the value of a branching game is a natural extension of the above problem obtained by allowing interplay between the players. The first author implicitly provided bounds on the complexity of the problem in the non-stochastic case. In this paper we complete those bounds by proving *2-EXP*-hardness of the problem. Additionally, we prove that the problem becomes undecidable if any form of randomisation is allowed (either by considering randomised or behavioural strategies; or by adding stochastic positions). The only remaining open question is the original one – when there are only stochastic positions and no players.

Branching games fall into a general category of games of *imperfect information*, i.e. games where the full information about the state of the game is not assumed: the definitions assure that the players have no information about the execution of separate threads. The area of imperfect information games is rich and not fully understood, see e.g. [7, 4, 6], see also [5]. In this context, branching games with regular objectives can be seen as a natural extension of *imperfect information games with ω-regular objectives* to the branching-time case.

2 Definitions

In this section we will define the objects studied in the paper. The crucial definitions are those of a *branching game* and *game values*. By ω we denote the set of natural numbers and $\mathbb{R}$ stands for the set of reals.

Words and trees. An *alphabet* Γ is a finite non-empty set. A *word* over Γ is any, possibly infinite, sequence $w = a_0 a_1 \cdots a_n \cdots$ where $a_i \in \Gamma$. By $w[i]$ we denote the i-th letter of w, i.e. a_i. ε stands for the empty sequence. Words are either *finite* (Γ^*) or *infinite* (Γ^ω). $|w|$ is the length of a finite word w. The prefix order on words is denoted $\sqsubseteq$.

A *tree* over an alphabet Γ is any partial function $t\colon \{\mathtt{L},\mathtt{R}\}^* \rightharpoonup \Gamma$ with a non-empty prefix-closed domain $\mathit{Dom}(t) \subseteq \{\mathtt{L},\mathtt{R}\}^*$. The elements $d \in \{\mathtt{L},\mathtt{R}\}$ are called *directions*, $\bar{d}$ is the direction opposite to d. Elements of the set $\{\mathtt{L},\mathtt{R}\}^*$ are called *nodes*. We say that a node u of a tree t is *fully branching* if it has two children in the tree, is *uniquely branching* if it has exactly one child in the tree. The set $\mathcal{T}_\Gamma$ is the set of all trees over an alphabet Γ. This set can naturally be enhanced with a topology in such a way that it becomes a homeomorphic copy of the Cantor set [25]. We say that a tree t_1 is a *prefix* of a tree t_2 if $t_1 \subseteq t_2$, i.e. $\mathit{Dom}(t_1) \subseteq \mathit{Dom}(t_2)$ and for every $u \in \mathit{Dom}(t_1)$ we have $t_1(u) = t_2(u)$.

Regular languages. In this work we use the standard notions of non-deterministic and alternating parity automata over infinite trees. Together with Monadic Second-Order logic, these automata form equivalent formalisms for defining *regular* languages of infinite trees. For an introduction to this area see for instance [24].

Branching games. This paper is about branching games. The two adversaries of our games are called Eve and Adam (or shortly E and A). Since we consider stochastic games, we additionally introduce *Nature* denoted $\mathcal{N}$. A *branching board* is a tuple $\mathtt{B} = \langle V, \Gamma, s_\mathtt{L}, s_\mathtt{R}, \rho, \eta, \lambda, v_\mathrm{I}\rangle$, where V is the set of *vertices*; Γ is the *alphabet*; $s_\mathtt{L}, s_\mathtt{R}\colon V \to V$ are the *successor functions*; $\lambda\colon V \to \Gamma$ is the *labelling* of the vertices; $\rho\colon V \to \{A, E, \mathcal{N}, \mathcal{B}\}$ is a partition of the vertices between Adam's, Eve's, *Nature*'s, and branching vertices; $\eta\colon \rho^{-1}\big(\{\mathcal{N}\}\big) \to \mathit{Dist}(\{\mathtt{L},\mathtt{R}\})$ maps *Nature*'s vertices to random distributions over the successors; $v_\mathrm{I} \in V$ is the *initial vertex*. We extend the assignment s to arbitrary sequences of directions in the natural way: $s_\varepsilon(v) = v$ and $s_{u\cdot d}(v) = s_d\big(s_u(v)\big)$.

For $P \in \{A, E, \mathcal{N}, \mathcal{B}\}$, by V_P we denote the set of vertices belonging to P, i.e. $\rho^{-1}\big(\{P\}\big)$. We say that $\mathtt{B}$ is *finitary* if the set of vertices V is finite and the values used to define η are rational. For $\mathcal{P} \subseteq \{A, E, \mathcal{N}, \mathcal{B}\}$ we say that $\mathtt{B}$ is $\mathcal{P}$-*branching* if $\mathit{Range}(\rho) \subseteq \mathcal{P}$. Every board $\mathtt{B}$ defines the tree $t^\lambda_\mathtt{B}\colon \{\mathtt{L},\mathtt{R}\}^* \to \Gamma$ as the unfolding of the adequate labelled sub-graph of the board, i.e. $t^\lambda_\mathtt{B}(u) = \lambda\big(s_u(v_\mathrm{I})\big)$. $\mathtt{B}$ is *non-stochastic* if it is $\{E, A, \mathcal{B}\}$-branching.

Intuitively, a play over a branching board $\mathtt{B}$ proceeds in threads, each thread has one token located in a vertex of the board. Initially, there is one thread with the token located in v_I. Consider a thread with a token located in a vertex v. If $\rho(v) = \mathcal{B}$ then the thread is duplicated into two separate threads with tokens located in $s_{\mathtt{L}}(v)$ and $s_{\mathtt{R}}(v)$. If $\rho(v) = \mathcal{N}$ then the token is moved either to $s_{\mathtt{L}}(v)$ or to $s_{\mathtt{R}}(v)$ depending on an independent random event with distribution $\eta(v)$. If $\rho(v) \in \{E, A\}$ then the respective player can make her/his choice depending on the history of the current thread. However, she/he cannot take into account positions of tokens from other threads in the current play. After all the threads moved infinitely many times, a tree-like play has been created. The winning condition of a branching game will indicate which plays are winning for which player. Figure 1 depicts a branching board and a play on this board.

We will now formalise the notions of a play and a pure strategy of a player. Consider a non-empty set $\mathcal{P} \subseteq \{A, E, \mathcal{N}, \mathcal{B}\}$. We say that a tree $t \subseteq t^{\lambda}_{\mathtt{B}}$ is *$\mathcal{P}$-branching* if it is fully branching in the nodes $u \in \{\mathtt{L},\mathtt{R}\}^*$ such that $\rho\big(s_u(v_{\mathrm{I}})\big) \in \mathcal{P}$ and uniquely branching in the remaining nodes. A *play* on a board $\mathtt{B}$ is a tree $t \subseteq t^{\lambda}_{\mathtt{B}}$ that is $\{\mathcal{B}\}$-branching. The set of all plays on a board $\mathtt{B}$ is denoted $\mathrm{plays}(\mathtt{B})$. For $P \in \{E, A, \mathcal{N}\}$ we say that a tree $t \subseteq t^{\lambda}_{\mathtt{B}}$ is a *pure strategy* of P over $\mathtt{B}$ if t is $\big(\{E, A, \mathcal{N}, \mathcal{B}\} \setminus \{P\}\big)$-branching. The set of pure strategies of P over $\mathtt{B}$ is denoted $\Sigma^P_{\mathtt{B}}$. Notice that the sets $\mathrm{plays}(\mathtt{B})$ and $\Sigma^P_{\mathtt{B}}$ for $P \in \{E, A, \mathcal{N}\}$ are closed sets of Γ-labelled trees. If V is finite then all these sets are regular.

Given three pure strategies $\sigma \in \Sigma^E_{\mathtt{B}}$, $\pi \in \Sigma^A_{\mathtt{B}}$, and $\eta \in \Sigma^{\mathcal{N}}_{\mathtt{B}}$ the *play resulting from σ, π, and η* (denoted $\mathrm{eval}_{\mathtt{B}}(\sigma, \pi, \eta)$) is the tree $\sigma \cap \pi \cap \eta \in \mathrm{plays}(\mathtt{B})$. Thus, $\mathrm{eval}_{\mathtt{B}} \colon \Sigma^E_{\mathtt{B}} \times \Sigma^A_{\mathtt{B}} \times \Sigma^{\mathcal{N}}_{\mathtt{B}} \to \mathrm{plays}(\Gamma)$. Notice that the function $\mathrm{eval}_{\mathtt{B}}$ is continuous.

Measure theory. For an introduction to measure theory we refer to [14, Chapter 17]. Measure properties of regular sets of trees are discussed in [13]. Let μ be a Borel measure on a topological space X. We say that μ is a *probability measure* if $\mu(X) = 1$. A function $f \colon X \to \mathbb{R}$ is μ-measurable if the pre-image of any measurable set in $\mathbb{R}$ is μ-measurable in X. $f \colon X \to \mathbb{R}$ is *universally measurable* if it is μ-measurable for every Borel measure μ on X. If $f \colon X \to \mathbb{R}$ is μ-measurable then by $\int_X f(x)\ \mu(\mathrm{d}x)$ we denote the integral of f with respect to the measure μ.

Branching games. A *branching game* is a pair $G = \langle \mathtt{B}, \Phi\rangle$ where $\mathtt{B}$ is a branching board and Φ is a universally measurable bounded real function $\Phi \colon \mathrm{plays}(\mathtt{B}) \to \mathbb{R}^+$. The notions of a *$\mathcal{P}$-branching* game and a *finitary* game refer to the respective properties of the board.

Mixed strategies. A mixed strategy of a player $P \in \{E, A\}$ is a Borel probability measure over the set $\Sigma^P_{\mathtt{B}}$. The set of all mixed strategies of P is denoted by $\Sigma^{MP}_{\mathtt{B}}$.

There is a natural way of defining a Borel probability measure $\eta^*_{\mathtt{B}}$ on the set $\Sigma^{\mathcal{N}}_{\mathtt{B}}$ of strategies of $\mathcal{N}$. This measure represents the intuition, that after a sequence of directions $u \in \{\mathtt{L},\mathtt{R}\}$ corresponding to a vertex $v = s_u(v_{\mathrm{I}}) \in V$ such that $\rho(v) = \mathcal{N}$, *Nature* chooses to move to a direction $d \in \{\mathtt{L},\mathtt{R}\}$ with the probability $\eta(v)(d)$ and is called *behavioural.*

Behavioural strategies. We say that a mixed strategy of P is *behavioural* if it is a "coin flipping" measure, i.e. a measure induced by supplying some of the nodes of $t^{\lambda}_{\mathtt{B}}$ corresponding to vertices of P with a probability distribution over the successors. To produce a pure strategy from a behavioural one, the directions are chosen independently according to the fixed probability distributions.

More formally, a mixed strategy τ of P is *behavioural* if it is, as a measure over $\Sigma^P_{\mathtt{B}}$, the measure $\eta^*_{\mathtt{B}'}$ for some (possibly not finitary) board $\mathtt{B}'$. The set of all behavioural strategies of P is denoted by $\Sigma^{BP}_{\mathtt{B}}$. Clearly we can treat every pure strategy in $\Sigma^P_{\mathtt{B}}$ as a Dirac delta function in $\Sigma^{MP}_{\mathtt{B}}$ (in fact in $\Sigma^{BP}_{\mathtt{B}}$). Thus, we can assume that $\Sigma^P_{\mathtt{B}} \subseteq \Sigma^{BP}_{\mathtt{B}} \subseteq \Sigma^{MP}_{\mathtt{B}}$.

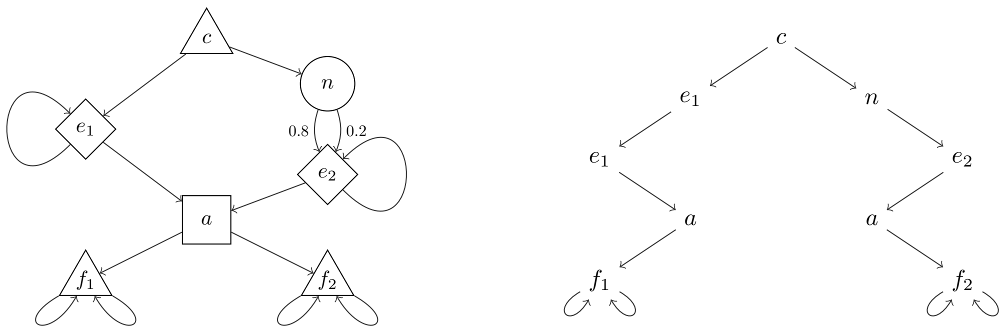


Figure 1 An example of a branching board and a play on this board. We denote Eve's, Adam's, *Nature*'s, and branching vertices by diamonds, squares, circles, and triangles respectively. *Nature*'s vertices are equipped with a probability distribution over the successors. The successors L and R agree with the directions on the picture, i.e. L moves to the left.

Strategies as functions. There is a different way to define the three types of strategies that may give more intuition to the behaviour and expressive power of the strategies. A pure strategy $\sigma \in \Sigma^P$ can be seen as a function $\sigma\colon \{\mathtt{L},\mathtt{R}\}^* \to \{\mathtt{L},\mathtt{R}\}$; a behavioural strategy $\sigma_b \in \Sigma^{BP}$ as a function $\sigma_b\colon \{\mathtt{L},\mathtt{R}\}^* \to \mu(\{\mathtt{L},\mathtt{R}\})$; and a mixed strategy $\sigma_m \in \Sigma^{MP}$ as a measure $\sigma_m \in \mu(\Sigma^P)$, where $\mu(X)$ dentotes some Borel probability mesure on the set X.

An example. Figure 1 depicts a branching board $\mathtt{B}$ and a play t on this board. We identify the vertices with their labels. A pure strategy of Adam can make different choices in a depending on the history of the thread that lead to this vertex (there are infinitely many such histories). A pure strategy of Eve can make different choices in e_2 depending on the edge taken by *Nature* in n. A mixed strategy of Eve can synchronise: with probability $\frac{1}{2}$ move to L in both vertices e_1, e_2; and with probability $\frac{1}{2}$ move to R in both of them. A behavioural strategy cannot make such a synchronisation: the probability distribution over the successors depends only on the history of the current thread.

Values of strategies. Assume that $\sigma_m \in \Sigma_{\mathtt{B}}^{ME}$ and $\pi_m \in \Sigma_{\mathtt{B}}^{MA}$ are two mixed strategies of the respective players. Our aim is to define the value $val_{\mathrm{G}}(\sigma_m, \pi_m)$. Intuitively, $val_{\mathrm{G}}(\sigma_m, \pi_m)$ should be the expected value of $\Phi\big(\mathrm{eval}_{\mathrm{G}}(\sigma, \pi, \eta)\big)$ where the pure strategies σ, π, and η are chosen according to the probability distributions σ_m, π_m, and $\eta_{\mathtt{B}}^*$ respectively. This is formalised as follows.

$$val_{\mathrm{G}}(\sigma_m, \pi_m) \overset{\mathrm{def}}{=} \int_{\Sigma_{\mathtt{B}}^E, \Sigma_{\mathtt{B}}^A, \Sigma_{\mathtt{B}}^N} \Phi\big(\mathrm{eval}_{\mathrm{G}}(\sigma, \pi, \eta)\big)\ \sigma_m(\mathrm{d}\sigma)\, \pi_m(\mathrm{d}\pi)\, \eta_{\mathtt{B}}^*(\mathrm{d}\eta) \tag{2}$$

If σ and π are pure strategies and the board is non-stochastic then $val_{\mathrm{G}}(\sigma, \pi) = \Phi(\pi \cap \sigma)$.

Values of a game. The aim of Eve in a branching game is to maximise the value $val_{\mathrm{G}}(\sigma, \pi)$. Let us define the *partial values* of the game. Consider $X \in \{\varepsilon, B, M\}$ (i.e. X stands for respectively *pure*, *behavioural*, and *mixed* strategies). The X *value of* G *for Eve* (resp. Adam) is defined as

$$val_{\mathrm{G}}^{XE} \overset{\mathrm{def}}{=} \sup_{\sigma \in \Sigma_{\mathtt{B}}^{XE}} val_{\mathrm{G}}(\sigma) \quad \text{where} \quad val_{\mathrm{G}}(\sigma) \overset{\mathrm{def}}{=} \inf_{\pi \in \Sigma_{\mathtt{B}}^{A}} val_{\mathrm{G}}(\sigma, \pi),$$

$$val_{\mathrm{G}}^{XA} \overset{\mathrm{def}}{=} \inf_{\pi \in \Sigma_{\mathtt{B}}^{XA}} val_{\mathrm{G}}(\pi) \quad \text{where} \quad val_{\mathrm{G}}(\pi) \overset{\mathrm{def}}{=} \sup_{\sigma \in \Sigma_{\mathtt{B}}^{E}} val_{\mathrm{G}}(\sigma, \pi).$$

Notice, that the second inf/sup is taken over the pure strategies of the opponent. This is explained by the following simple lemma.

▶ **Lemma 1.** *Let* G *be a branching game. If* σ_m *is Eve's mixed strategy then*

$$\inf_{\pi_m \in \Sigma_{\mathtt{B}}^{MA}} val_{\mathrm{G}}(\sigma_m, \pi_m) = \inf_{\pi_b \in \Sigma_{\mathtt{B}}^{BA}} val_{\mathrm{G}}(\sigma_m, \pi_b) = \inf_{\pi \in \Sigma_{\mathtt{B}}^{A}} val_{\mathrm{G}}(\sigma_m, \pi)$$

Dually, the same holds for mixed strategies of Adam if we replace inf *with* sup *and* A *with* E*.*

Determinacy. As a simple consequence of Lemma 1 we obtain the following inequalities

$$val_{\mathrm{G}}^{A} \geq val_{\mathrm{G}}^{BA} \geq val_{\mathrm{G}}^{MA} \geq val_{\mathrm{G}}^{ME} \geq val_{\mathrm{G}}^{BE} \geq val_{\mathrm{G}}^{E}. \qquad (3)$$

The first two (resp. the last two) inequalities hold by the fact that we take inf (resp. sup) over greater (reps. smaller) sets of strategies. The third inequality holds by Lemma 1 and the fact that $\inf_x \sup_y f(x,y) \geq \sup_y \inf_x f(x,y)$.

We will say that a branching game G is *determined*

- *under pure strategies* if $val_{\mathrm{G}}^{A} = val_{\mathrm{G}}^{E}$,
- *under behavioural strategies* if $val_{\mathrm{G}}^{BA} = val_{\mathrm{G}}^{BE}$,
- *under mixed strategies* if $val_{\mathrm{G}}^{MA} = val_{\mathrm{G}}^{ME}$.

Clearly, Equation (3) shows that pure determinacy implies behavioural determinacy and behavioural determinacy implies mixed determinacy. In general, the opposite implications do not hold. The questions of determinacy of branching games are discussed in Section 3.

Regular branching games. The following theorem implies that we can take as Φ an indicator of a regular language of trees $L \subseteq \mathrm{plays}(\mathtt{B})$, i.e. $\Phi(t) = 1$ if $t \in L$ and $\Phi(t) = 0$ otherwise. In that case we say that a game G has L *as a winning condition* and we write $G = \langle \mathtt{B}, L\rangle$ instead of $G = \langle \mathtt{B}, \Phi\rangle$.

▶ **Theorem 2** (Michalewski et al. [13]). *Every regular language* L *of infinite trees is universally measurable, i.e. for every Borel measure* μ *on the set of trees, we know that* L *is* μ*-measurable.*

3 Determinacy

In this section we study determinacy of branching games in the three variants: pure, behavioural, and mixed; see (3). We will show that for general regular winning conditions all three variants fail. However, when we restrict to closed regular winning sets we can recover the mixed determinacy.

Notice that if a branching game is not branching, i.e. it is a $\{E, A, \mathcal{N}\}$-branching game then the determinacy is well-understood [16, 7]. Similarly, if there are no positions of one of the players then the game is purely determined by Lemma 1. Therefore, the simplest case specific for the branching games are the $\{E, A, \mathcal{B}\}$-branching games.

3.1 Behavioural indeterminacy

We start by proving the following theorem.

▶ **Theorem 3.** *There is a* $\{E, A, \mathcal{B}\}$*-branching game* G *with a regular winning condition that is both closed and open such that* G *is not determined under behavioural strategies.*

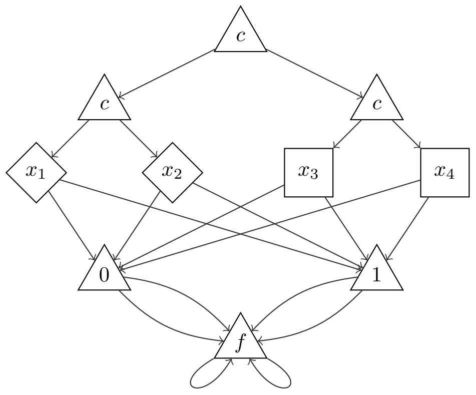


Figure 2 A branching board that is not determined under behavioural strategies.

The board B of the game G is depicted in Figure 2. A play $t \in \text{plays}(\mathtt{B})$ over B starts by splitting into four separate threads by the $\mathcal{B}$-vertices labelled with c. Then, each of the players can perform two separate choices, E in the two vertices labelled x_1 and x_2, and A in the two vertices labelled x_3 and x_4. Their choices lead to vertices labelled by either 0 or 1. The rest of the play stays forever in the branching vertex labelled by f. For $i = 1, 2, 3, 4$ let $x_i(t) \in \{0, 1\}$ be label chosen by the respective player in the vertex labelled by x_i, i.e. the label of the unique child of the unique node labelled by x_i in t. Consider a winning set $L \subseteq \text{plays}(\mathtt{B})$ defined as follows

$$L \overset{\text{def}}{=} \{t \in \text{plays}(\mathtt{B}) \mid x_1(t) = x_2(t) = x_3(t) = x_4(t) \ \vee \ x_3(t) \neq x_4(t)\} \tag{4}$$

In other words, Eve wins a play t if either Adam has chosen two different labels in x_3 and x_4 or all the chosen labels are equal. Since the vertices labelled x_i lie at a fixed depth of every play $t \in \text{plays}(\mathtt{B})$, L is a closed and open regular language of infinite trees.

▶ **Example 4.** The game $\mathrm{G} = \langle \mathtt{B}, L\rangle$ has the following partial values:

$$val_{\mathrm{G}}^{A} = 1; \qquad val_{\mathrm{G}}^{BA} = \frac{3}{4}; \qquad val_{\mathrm{G}}^{MA} = \frac{1}{2} = val_{\mathrm{G}}^{ME}; \qquad val_{\mathrm{G}}^{BE} = \frac{1}{4}; \qquad val_{\mathrm{G}}^{E} = 0.$$

We first argue about the pure values – a pure strategy over the board from Figure 2 needs to declare in advance the two values $x_i(t)$ and $x_{i+1}(t)$ for $i = 1, 3$ depending on the player.

If such a strategy is fixed, the opponent can choose his values in such a way to win.

We now consider the mixed value. Let σ_m randomly choose with equal probability between the following two pure strategies σ_i for $i = 0, 1$: the strategy σ_i satisfies $x_1(\sigma_i) = x_2(\sigma_i) = i$. π_m is defined analogously. It is easy to check that these strategies are optimal and witness that the mixed value of the game is $\frac{1}{2}$ for both players.

Consider a behavioural strategy σ_b of Eve (the case of Adam is entirely dual). Such a strategy can be described by two independent random choices:

1. σ_b chooses x_1 to be 0 with probability p_1,
2. σ_b chooses x_2 to be 0 with probability p_2.

Thus, each behavioural strategy of Eve is characterised by a pair of numbers $p_1, p_2 \in [0, 1]$. A simple computation shows that no matter how Eve chooses her values p_1, p_2, Adam can find a counter-strategy guaranteeing the value of at most $\frac{1}{4}$.

Since $val_{\mathrm{G}}^{BA} = \frac{3}{4} \neq \frac{1}{4} = val_{\mathrm{G}}^{BE}$ the proof of Theorem 3 is concluded.

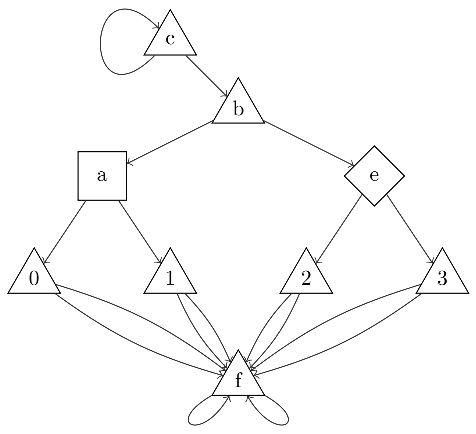


Figure 3 A branching board that is not determined under mixed strategies.

3.2 Mixed indeterminacy

We will now show that the mixed determinacy fails for relatively simple regular sets, as expressed by the following theorem.

▶ **Theorem 5.** *There is a* $\{E, A, \mathcal{B}\}$*-branching game with a regular winning set being a difference of two open sets that is not determined under mixed strategies.*

To prove this theorem we will encode the following game as a $\{E, A, \mathcal{B}\}$-branching game G. Assume that ∞ is an additional symbol such that for every $n \in \omega$ we have $n < \infty$.

▶ **Example 6** (Folklore)**.** Consider the following game: Adam and Eve simultaneously and independently choose two numbers: Eve chooses $e \in \omega \cup \{\infty\}$, Adam chooses $a \in \omega \cup \{\infty\}$. Eve wins if $e < \infty$, and either $a = \infty$ or $a \leq e$.

It is easy to see that this game is not determined under mixed strategies. Intuitively, it follows from the fact that both players try to choose a finite number as big as possible.

The board $\mathtt{B}$ of the game G is depicted in Figure 3. A play $t \in \mathrm{plays}(\mathtt{B})$ consists of infinitely many independent sub-games that start in the vertices labelled by b. More precisely, the k-th sub-game starts in the node $\mathtt{L}^k\mathtt{R}$ in the tree t. Such a sub-game is split into two independent choices: Adam chooses a label, either 0 or 1, for the successor of the node labelled by a; Eve chooses a label, either 2 or 3, for the successor of the node labelled by e.

Let $a(t)$ (resp. $e(t)$) be the smallest number $k \in \omega$ such that Eve (resp. Adam) has chosen an odd label in the k-th sub-game, i.e. $\mathtt{L}^k\mathtt{RLR} \in \mathit{Dom}(t)$ (resp. $\mathtt{L}^k\mathtt{RRR} \in \mathit{Dom}(t)$). If no such number exists then $a(t)$ (resp. $e(t)$) equals ∞.

Let the winning condition L of the game G be defined as follows

$$L \overset{\text{def}}{=} \big\{t \in \mathrm{plays}(\mathtt{B}) \mid e(t) < \infty \text{ and not } (e(t) < a(t) < \infty)\big\}. \tag{5}$$

It is easy to see that L is a regular language of infinite trees (to compare $a(t)$ with $e(t)$ it is enough to notice that each of these values corresponds to a node on the left-most branch of the play t). Moreover, both the conditions $e(t) < \infty$ and $e(t) < a(t) < \infty$ are open sets of plays.

Hence, the game $\mathrm{G} = \langle \mathtt{B}, L\rangle$ is a game as required in Theorem 5. Moreover, there is a clear correspondence between the pure strategies in G and the pure strategies in the game from Example 6. This correspondence extends to the mixed strategies what implies the following claim.

▶ **Claim 7.** *We have that* $val^{MA}_{\mathrm{G}} = 1$ *and* $val^{ME}_{\mathrm{G}} = 0$.

This concludes the proof of Theorem 5.

3.3 Mixed determinacy for closed sets

In this section we use Glicksberg's minimax theorem to prove that if a winning condition is a closed set of plays then the game is determined under mixed strategies.

▶ **Theorem 8.** *If* $\mathrm{G} = \langle \mathtt{B}, L\rangle$ *is a* $\{E, A, \mathcal{N}, \mathcal{B}\}$*-branching game and* L *is an arbitrary closed subset of* $\mathrm{plays}(\mathtt{B})$ *then* G *is determined under mixed strategies.*

Before we recall the statement of Glicksberg's minimax theorem, let us introduce some relevant notions. Assume that X is a metrisable topological space. We say that a function $f\colon X \to \mathbb{R}$ is *upper semi-continuous* if for every $x_0 \in X$ we have $\limsup_{x\to x_0} \leq f(x_0)$. Clearly, if $C \subseteq X$ is a closed subset of X then the characteristic function of C is upper semi-continuous. Also, a composition of a continuous function and an upper semi-continuous function is upper semi-continuous.

▶ **Theorem 9** (Glicksberg's minimax theorem [12], see also [18, pages 299–306])**.** *Let* A*,* B *be compact metrisable spaces and* $f\colon A \times B \to \mathbb{R}$ *be an upper semi-continuous function. Then the following holds*

$$\sup_{\mu} \inf_{\nu} \int_{A,B} f(a,b)\ \mu(\mathrm{d}a)\,\nu(\mathrm{d}b) = \inf_{\nu} \sup_{\mu} \int_{A,B} f(a,b)\ \mu(\mathrm{d}a)\,\nu(\mathrm{d}b), \tag{6}$$

where μ*,* ν *range over the Borel probability measures on the sets* A*,* B *respectively.*

It remains to prove that if $\mathrm{G} = \langle \mathtt{B}, L\rangle$ with $L \subseteq \mathrm{plays}(\mathtt{B})$ closed then the function $val_{\mathrm{G}}\colon \Sigma^{E}_{\mathtt{B}} \times \Sigma^{A}_{\mathtt{B}} \to \mathbb{R}$ is upper semi-continuous. This function can be written as a composition of two functions. The first one maps a pair of pure strategies (σ, π) to a measure on $\mathrm{plays}(\mathtt{B})$ defined as $\mu_{(\sigma,\pi)}(T) \overset{\text{def}}{=} \nu^{*}_{\mathtt{B}}\big(\{t \in \Sigma^{\mathcal{N}}_{\mathtt{B}} \mid \sigma \cap \pi \cap t \in T\}\big)$, i.e. the $\nu^{*}_{\mathtt{B}}$ measure of the pre-image of the set T under the function that intersects the three strategies. This mapping is continuous, as proved by Mio in [20, Lemma 4.1.4]. The second one applies the measure $\mu_{(\sigma,\pi)}$ to the winning set $L \subseteq \mathrm{plays}(\mathtt{B})$. For a closed set L this function is upper semi-continuous by [14, Corollary 17.21].

4 Computing game values

In this section we will discuss the computational complexity of determining the partial values of branching games. To be more precise, we consider the following family of problems, parametrised by the set of available positions $\mathcal{P} \subseteq \{A, E, \mathcal{N}, \mathcal{B}\}$ and the type of the value $V \in \{val^{A}, val^{BA}, val^{MA}, val^{ME}, val^{BE}, val^{E}\}$.

▶ **Problem 10** (The value V of a regular $\mathcal{P}$-branching game)**.**

- **Input:** *A finitary* $\mathcal{P}$*-branching game* G *with the winning condition given by a non-deterministic tree automaton.*
- **Output:** *Does* $V > \frac{1}{2}$?

4.1 The non-stochastic case

If no random choice is involved, i.e. the board has no *Nature*'s positions and we consider pure strategies, the values belong to the set $\{0, 1\}$ and we can compute them, as expressed by the following theorem.

▶ **Theorem 11.** *The value val^E problem of a regular $\{A, E, \mathcal{B}\}$-branching game is in 2-EXP, the value val^A problem of a regular $\{A, E, \mathcal{B}\}$-branching game is EXP-complete.*

Moreover, the value val^E problem of a regular $\{A, E, \mathcal{B}\}$-branching game is 2-EXP-complete if the winning condition is given by an alternating tree automaton.

This theorem follows from the constructions in [22], performed in a bit different language. The asymmetry in this theorem comes from the fact that in Problem 10 we assume that the winning condition of a game is given as a non-deterministic automaton. In this work we strengthen the second part of the above theorem by proving that the value val^E problem of a regular $\{A, E, \mathcal{B}\}$-branching game is *2-EXP*-hard also for non-deterministic automata. This is achieved by using the completeness result from [22] together with the following reduction. It is somehow surprising to notice that in the context of branching games one can de-alternate an automaton in polynomial time.

▶ **Theorem 12.** *There exists a polynomial time reduction that inputs a $\{A, E, \mathcal{B}\}$-branching game G with the winning condition given as an alternating tree automaton and constructs a $\{A, E, \mathcal{B}\}$-branching game G' with the winning condition given by a non-deterministic tree automaton, such that $val^E_{\mathrm{G}} = val^E_{\mathrm{G}'}$.*

The proof is straightforward, its main idea is to split the alternation of the given automaton into two parts: the choices of Adam and the choices of Eve. In the game G' the former choices will be done explicitly on the board while the latter choices will be performed by the non-deterministic automaton that recognises the winning condition of G'.

▶ **Corollary 13.** *val^E problem of a regular $\{A, E, \mathcal{B}\}$-branching game is 2-EXP-complete.*

4.2 The stochastic cases

The above decidability results hold for non-stochastic games and pure strategies. Restoring any of those features yields undecidability, as expressed by the two theorems of this section.

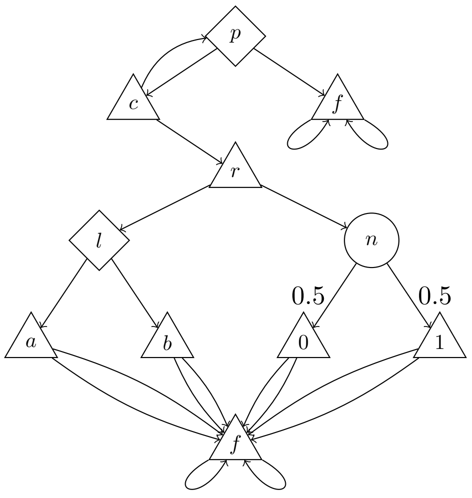
▶ **Theorem 14.** *For every $V \in \{val^A, val^{BA}, val^{MA}, val^{ME}, val^{BE}, val^E\}$ and $P \in \{E, A\}$, the value V problem of a regular $\{P, \mathcal{N}, \mathcal{B}\}$-branching game is undecidable.*

Observe that by Lemma 1 a $\{P, \mathcal{N}, \mathcal{B}\}$-branching game is determined under pure strategies. It means that all the six partial values are the same for such games. Thus, by the symmetry we can assume that $P = E$ and $V = val^E$.

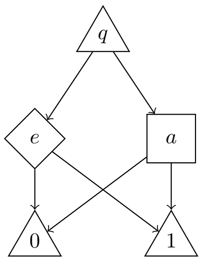
To prove Theorem 14 we reduce the following undecidable problem, cf. [11]. It can be shown that the word problem is undecidable even if we restrict our attention to a two-letters alphabet and the so-called *very simple* automata: a non-deterministic automaton is *very simple* if from every state and every letter there are exactly two possible transitions leading to two distinct states.

▶ **Problem 15** (Word problem for VSNA).

- **Input:** *A very simple non-deterministic automaton $\mathcal{A}$ on finite words over $\{a, b\}$.*
- **Output:** *Does there exist a finite word such that more than half of the runs of $\mathcal{A}$ on this word is accepting?*



(a) A branching board used in the proof of Theorem 14.



(b) A gadget used in the proof of Theorem 16. to replace *Nature*'s vertex in the board

Figure 4 Boards used in undecidability proofs.

We will now sketch the proof of Theorem 14. Let us take a very simple non-deterministic automaton $\mathcal{A}$ and assume that the two transitions over a letter $l \in \{a, b\}$ from a state $q \in Q^{\mathcal{A}}$ lead to the states $\delta_0(q, l)$ and $\delta_1(q, l)$. Assume that $l_0, l_1, \ldots, l_k$ is a sequence of letters $l_i \in \{a, b\}$ and $n_0, n_1, \ldots, n_k$ is a sequence of numbers $n_i \in \{0, 1\}$. These two sequences allow us to naturally define a run $\rho = \mathrm{run}(\vec{l}, \vec{n})$ of $\mathcal{A}$ over the word $l_0, \ldots, l_k$ that follows the respective transitions of $\mathcal{A}$: $\rho[0] = q_{\mathrm{I}}$ and $\rho[i + 1] = \delta_{n_i}(\rho[i], l_i)$.

Consider the board $\mathtt{B}$ depicted on Figure 4a. A play on this board consists of a sequence of decisions made by Eve, whether to move from the vertex labelled l to a or to b. At every moment Eve can stop this sequence by choosing the right successor of the vertex labelled p. For every choice of a or b by Eve, the *Nature* simultaneously chooses a number 0 or 1. Thus, a play t results in two finite or infinite sequences of the same length: $l_0, l_1, \ldots$ with $l_i \in \{a, b\}$ and $n_0, n_1, \ldots$ with $n_i \in \{0, 1\}$. Consider the following winning condition

$$L \overset{\text{def}}{=} \{t \in \mathrm{plays}(\mathtt{B}) \mid \text{the sequences } \vec{l} \text{ and } \vec{n} \text{ are finite and } \mathrm{run}(\vec{l}, \vec{n}) \text{ is accepting}\}. \tag{7}$$

Now let $\mathrm{G} = \langle \mathtt{B}, L\rangle$. It is easy to see that the winning condition L can be represented as a regular language of infinite trees. A pure strategy of Eve in G either never moves from the vertex labelled p to the vertex labelled f (in that case its value is 0) or in the opposite case it corresponds to a finite word $l_0, l_1, \ldots, l_k$. The value of such a strategy is the probability that the choices of *Nature* will represent an accepting run of $\mathcal{A}$ over the word $\vec{l}$. Thus, Eve has a pure strategy σ with $val^{E}_{\mathrm{G}}(\sigma) > \frac{1}{2}$ if and only if more than half of the runs of $\mathcal{A}$ over the word $\vec{l}$ produced by σ is accepting.

To complete the landscape of decidability we state.

▶ **Theorem 16.** *For every $V \in \{val^{BA}, val^{MA}, val^{ME}, val^{BE}\}$ the value V problem of a regular $\{E, A, \mathcal{B}\}$-branching game is undecidable.*

The theorem follows from the fact that the game used in the proof of Theorem 14 can be simulated on the board with *Nature*'s position replaced by the gadget depicted in Figure 4b.

5 Conclusions

In this work we have studied questions of determinacy and decidability of regular branching games. We have shown that the games are not determined even for topologically simple regular conditions. In the case of mixed determinacy, the frontier lies in the first level of the difference hierarchy of closed sets. Additionally, we have shown that the question whether the value of a given game is greater than a fixed threshold is undecidable in all non-trivial stochastic cases. In the non-stochastic cases (i.e. when the board is non-stochastic and we ask about pure strategies) we have given exact bounds on the complexity of the problem. The only remaining case is the 0-player stochastic case, i.e. the problem of computing the measure of a regular language of infinite trees.

Further work. It seems interesting to understand for which classes of regular winning conditions, the branching games are determined. It was proved by the first author in [22] that the non-stochastic branching games with winning conditions given by *game automata* are determined under pure strategies. We believe that the proof can be naturally extended to the stochastic case. However, there are regular languages of infinite trees L that are not recognisable by game automata, but still all the branching games with the winning condition L are purely determined. The characterisation of such objectives poses an interesting research direction as it could give a broader class of games with decidable value problem.

On the frontier of mixed determinacy, it seems that allowing the objective to check local consistency at arbitrary depths of the tree is the cause of both the indeterminacy and the undecidability. This intuition suggests the following conjecture. We say that L is a *path language* if L is a Boolean combination of languages of the form

$$\{t \mid \text{there exists a branch of } t \text{ belonging to a regular language of infinite words } K \subseteq \Gamma^\omega\}.$$

▶ **Conjecture 17.** *If* $\mathrm{G} = \langle \mathtt{B}, L\rangle$ *is a branching game and* L *is a path language then the game* G *is determined under mixed strategies.*

Acknowledgement. The authors would like to thank Damian Niwiński for insightful discussions on the subject.

References

1 Eugene Asarin, Julien Cervelle, Aldric Degorre, Catalin Dima, Florian Horn, and Victor Kozyakin. Entropy games. *CoRR*, abs/1506.04885, 2015. URL: `http://arxiv.org/abs/1506.04885`.

2 Robert J. Aumann. Mixed and behavior strategies in infinite extensive games. Research Memorandum 32, Economic Research Program, Princeton University, 1961.

3 Julius Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

4 Krishnendu Chatterjee and Laurent Doyen. The complexity of partial observation parity games. In *In Proc. LPAR'10, LNCS 6397*. Springer, 2010.

5 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. A survey of partial-observation stochastic parity games. *Formal Methods in System Design*, 43(2):268–284, 2012.

6 Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Algorithms for omega-regular games with imperfect information. *CoRR*, abs/0706.2619, 2007. URL: `http://arxiv.org/abs/0706.2619`.

7 Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

8 Taolue Chen, Klaus Dräger, and Stefan Kiefer. Model checking stochastic branching processes. In *MFCS*, pages 271–282, 2012.

9 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theor. Comput. Sci.*, 352(1–3):190–196, 2006.

10 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997.

11 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In *ICALP (2)*, pages 527–538, 2010.

12 Irving Leonard Glicksberg. *Minimax Theorem for Upper and Lower Semi-continuous Payoffs.* Memorandum (Rand Corporation). Rand Corporation, 1950.

13 Tomasz Gogacz, Henryk Michalewski, Matteo Mio, and Michał Skrzypczak. Measure properties of game tree languages. In *MFCS*, pages 303–314, 2014.

14 Alexander Kechris. *Classical descriptive set theory.* Springer-Verlag, New York, 1995.

15 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

16 Donald A. Martin. The determinacy of Blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

17 Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

18 P. Wolfe Melvin Dresher, A. W. Tucker. *Contributions to the Theory of Games.* Number 3 in Annals of mathematics studies. Princeton University Press, 1957.

19 Henryk Michalewski and Matteo Mio. On the problem of computing the probability of regular sets of trees. In *FSTTCS*, pages 489–502, 2015.

20 Matteo Mio. *Game Semantics for Probabilistic μ-Calculi.* PhD thesis, University of Edinburgh, 2012.

21 Matteo Mio. On the equivalence of game and denotational semantics for the probabilistic mu-calculus. *Logical Methods in Computer Science*, 8(2), 2012.

22 Marcin Przybyłko. Tree games with regular objectives. In *GandALF*, pages 231–244, 2014.

23 Michael Oser Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. of the American Math. Soc.*, 141:1–35, 1969.

24 Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, pages 389–455. Springer, 1996.

25 Wolfgang Thomas and Helmut Lescow. Logical specifications of infinite computations. In *REX School/Symposium*, pages 583–621, 1993.

Symbolic Lookaheads for Bottom-Up Parsing

Paola Quaglia

University of Trento, Italy
`paola.quaglia@unitn.it`

Abstract

We present algorithms for the construction of LALR(1) parsing tables, and of LR(1) parsing tables of reduced size. We first define specialized characteristic automata whose states are parametric w.r.t. variables symbolically representing lookahead-sets. The propagation flow of lookaheads is kept in the form of a system of recursive equations, which is resolved to obtain the concrete LALR(1) table. By inspection of the LALR(1) automaton and of its lookahead propagation flow, we decide whether the grammar is LR(1) or not. In the positive case, an LR(1) parsing table of reduced size is computed by refinement of the LALR(1) table.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, F.4.3 Formal Languages

Keywords and phrases LALR(1) grammars, LR(1) grammars, Bottom-up parsing

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.79

1 Introduction

Various classes of grammars can be parsed bottom-up by applying the same shift/reduce algorithm driven by different parsing tables (e.g., SLR(1) [4], LALR(1) [3], LR(1) [2]). Parsing tables are defined on top of deterministic finite state characteristic automata whose size is crucial to the applied parsing technique: the finer the information encoded by automata, the larger the class of parsed grammars, and the bigger the size of parsing tables.

LR(1)-automata are the richer structures in LR(1) parsing. The number of states of these automata has the striking upper bound $O(2^{n(t+1)})$ in the size of the grammar and in the number of terminal symbols (n and t, resp.) [12]. LALR(1) grammars have been defined as a technical compromise between the abundance of syntactic constructs of the generated languages and the size of the associated parsing tables. All the states of the LR(1)-automaton sharing the same LR(0) projection are collapsed into a single state of the LALR(1)-automaton. So, the size of LALR(1) parsing tables is much smaller than that of the corresponding LR(1) tables, and definitely tractable, as widespread parser generators clearly show [9, 2, 6]. At the same time, though, either debugging an LALR(1) grammar or choosing the appropriate directives for resolving conflicts is made harder by the fact that the user is compelled to reason about the propagation of LR(1) lookaheads modulo the – technical, and not necessarily intuitive - merging of LR(1)-states.

In this setting, and especially for large automata, it can be beneficial having some sort of explicit representation of the lookahead propagation flow among the states of the underlying automaton. We work towards this direction and propose a technique for the construction of LALR(1)-automata which provides a compact encoding of the propagation of lookaheads from one state to the other. The approach is based on the definition of symbolic characteristic automata that use items with two components: an LR(0)-item, and a symbolic lookahead-set. When a new state P is added to the automaton, each kernel LR(0)-item is associated with a variable that is propagated to the closure items of P. All the contributions to the lookahead-sets of the items in P are recorded by equations over the variables owned by the kernel items of the state. The propagation flow of lookaheads is embedded by defining equations for variables. For instance, a plausible equation for the variable x can look like $x \doteq \{b, x'\}$, meaning that x can take the value b and all the values that x' can take. To disclose the actual values of lookahead-sets, we resolve the system of equations. We reduce this problem to a reachability problem on the dependency graph of a propagation relation over equivalence classes of variables.

Once symbolic lookahead-sets are instantiated to ground elements, we construct the LALR(1) parsing table for the given grammar. We prove that the algorithm for the LALR(1) construction is correct. The proof of the assertion is based on two auxiliary results. One of them is the symbolic correspondence between the proposed automata and the merged LR(1)-automata used in the simplest algorithm for the construction of LALR(1) parsing tables. The other intermediate result is the correctness of the actualization of lookahead-sets.

Further, we describe an algorithm for the construction of LR(1) parsing tables. Such an algorithm is itself an example of use of the explicit representation of the lookahead propagation in LALR(1)-automata. The table for the larger class is obtained by refining the LALR(1) table. We look after reduce/reduce conflicts of the LALR(1) table and check whether they can be eliminated by unrolling the relevant LALR(1) mergings of states. This by-need strategy has the further benefit of providing opportunities for the early detection that the grammar at hand is not LR(1).

The rest of the paper is organized as follows. Basic definitions and conventions are introduced in Sec. 2. Sec. 3 presents symbolic automata, and their properties. The construction of LALR(1) parsing tables is the subject of Sec. 4, and the algorithm for the construction of LR(1) parsing tables is sketched in Sec. 5. Sec. 6 concludes this extended abstract.

2 Preliminaries

In this section we will introduce the basic definitions and the conventions which will be used in this manuscript. Some familiarity with the theory of LR(1)-parsing is assumed.

A context-free grammar is a tuple $\mathcal{G} = (V, T, S, \mathcal{P})$ whose elements represent, respectively, the vocabulary, the set of terminal symbols in the vocabulary, the start symbol, and the set of productions. Productions are written $A \to \beta$ where $A \in V \setminus T$, and $\beta \in V^*$. The reflexive and transitive closure of the one-step rightmost derivation relation is denoted by '$\Rightarrow^*$'.

We assume that grammars are reduced, and the following notational conventions are adopted. Members of V are denoted by $Y, Y_0, \ldots$; members of V^* by $\alpha, \beta, \ldots$; members of $(V \setminus T)$ by $A, B, \ldots$; members of T by $a, b, \ldots$; and members of T^* by $w, w_0, \ldots$. The empty string is denoted by ϵ. For every $\alpha \in V^*$, first(α) denotes the set of terminals that begin strings w such that $\alpha \Rightarrow^* w$. Moreover, if $\alpha \Rightarrow^* \epsilon$ then $\epsilon \in$ first(α).

Given any context-free grammar $\mathcal{G}$, parsing is applied to strings followed by the endmarker symbol $\$ \notin V$. Also, the parsing table refers to an augmented version of $\mathcal{G}$, denoted by $\mathcal{G}' = (V', T, S', \mathcal{P}')$ where, for a fresh symbol S', $V' = V \cup \{S'\}$, and $\mathcal{P}' = \mathcal{P} \cup \{S' \to S\}$. An LR(0)-item of $\mathcal{G}'$ is a production of $\mathcal{G}'$ with the marker "$\cdot$" at some position of its body. An LR(1)-item of $\mathcal{G}'$ is a pair consisting of an LR(0)-item of $\mathcal{G}'$ and of a symbol in the set $T \cup \{\$\}$. The LR(0)-item $A \to \alpha \cdot \beta$ is called *kernel item* if either $\alpha \neq \epsilon$ or $A = S'$, *closure item* if it is not kernel, *reducing item* if $\beta = \epsilon$, and *bypassing item* if it is not reducing. The same terminology is naturally extended to the LR(1)-items with first projection $A \to \alpha \cdot \beta$. For a set L of LR(1)-items, prj(L) is the set of LR(0)-items occurring as first components of the elements of L, and kernel(L) is the set of the kernel items of L.

We will call *LR(0)-automata*, and *LR(1)-automata* respectively, the characteristic automata constructed by collecting sets of LR(0)-items, and sets of LR(1)-items respectively. Also, we will call *LRm(1)-automata* the characteristic "merged" LR(1)-automata which are at the basis of the simplest, although inefficient, algorithm for the construction of LALR(1) parsing tables. We assume that LR(1)-automata and LRm(1)-automata are constructed after the methodology fully detailed in [1] and in its previous editions.

Below, we will denote the above mentioned characteristic automata by tuples of the form $(\mathcal{Q}, V, \tau, Q_0, \mathcal{F})$ where $\mathcal{Q}$ is the set of states, V the vocabulary, $\tau : (\mathcal{Q} \times V) \to \mathcal{Q}$ the transition function, $Q_0 \in \mathcal{Q}$ the initial state, and $\mathcal{F} \subseteq \mathcal{Q}$ the set of final states, i.e. of the states containing at least one reducing item. In particular, we will use the tuple $(St_l, V, \tau_l, L_0, F_l)$, called $\mathscr{A}_l$ for short, to stand for the LR(1)-automaton for $\mathcal{G}$. The tuple $(St_m, V, \tau_m, M_0, F_m)$, named $\mathscr{A}_m$, will denote the LRm(1)-automaton for $\mathcal{G}$.

Bottom-up parsing tables are filled in after an algorithm which is shared by various techniques (SLR(k), LALR(k), etc.). *Shift* and *goto* moves are determined after the transition function of the appropriate characteristic automaton. Further, the reducing items in the final states of the automaton, and the lookahead-sets associated with them, are used to set up *reduce* moves. By that, we will call *parsing table* the pair consisting of a characteristic automaton and of the collection of lookahead-sets for the reducing items of its final states.

3 Symbolic characteristic automata

In this section we will present the construction of the symbolic characteristic automaton that is central to further developments. In what follows, the prototypical LALR(1) grammar $\mathcal{G}_1$ with start symbol S_1 and production set $\mathcal{P}_1 = \{S_1 \to L = R \mid R,\ L \to *R \mid \text{id},\ R \to L\}$ [2] will be used as running example.

We let $\mathbb{V}$ be a set of symbols disjoint from $V' \cup \{\$\}$. Elements of $\mathbb{V}$ stand for *variables* and are ranged over by $x, x', \ldots$. We use $\Delta, \Delta', \ldots, \Gamma, \Gamma', \ldots$ to denote subsets of $\mathbb{V} \cup T \cup \{\$\}$. Also, we let $\text{ground}(\Delta) = \Delta \cap (T \cup \{\$\})$, and $\text{var}(\Delta) = \Delta \cap \mathbb{V}$. A *symbolic item* of $\mathcal{G}'$ is a pair of the shape $[A \to \alpha \cdot \beta, \Delta]$, whose second component is called *lookahed-set*. Below, symbolic items will be shortly called items when no confusion may arise. We assume the existence of a function newVar() which returns a fresh symbol of $\mathbb{V}$ at any invocation. This assumption on newVar() induces a strict total order over the generated variables, and we write $x \prec x'$ if the call to newVar() which returns x precedes the invocation of newVar() whose response is x'.

The definitions of kernel, closure, reducing, and bypassing items are extended to symbolic items in the natural way. Also, functions prj(_) and kernel(_) are overloaded to be applied to sets of symbolic items. Function first(_) is extended to arguments of the form $\beta\Delta$ as follows:

$$\text{first}(\beta\Delta) = \begin{cases} \text{first}(\beta) & \text{if } \epsilon \notin \text{first}(\beta) \\ (\text{first}(\beta) \setminus \{\epsilon\}) \cup \Delta & \text{otherwise.} \end{cases}$$

The closure of a set of symbolic items P, written closure(P), is defined as the smallest set of items, with smallest lookahead-sets, that satisfies the following equation:

$$\begin{aligned} \text{closure}(P) = P \cup \{[B \to \cdot\gamma, \Gamma] \text{ such that} \\ [A \to \alpha \cdot B\beta, \Delta] \in \text{closure}(P) \text{ and } B \to \gamma \in \mathcal{P} \text{ and first}(\beta\Delta) \subseteq \Gamma\}. \end{aligned}$$

The symbolic characteristic automaton for $\mathcal{G}$ is a tuple $(St_s, V, \tau_s, P_0, F_s, \mathit{Vars}, \mathit{Eqs})$, that we will shortly denote by $\mathscr{A}_s$. The first five elements of the tuple represent the set of states, the vocabulary, the transition function, the initial state, and the set of final states. *Vars* is a set of variables, and *Eqs* is a queue of defining equations of the form $x \doteq \Delta$ for the variables in *Vars*. When the actual ordering of the enqueued equations is irrelevant, we will interpret *Eqs* just as a set.

```
$x_0 \longleftarrow \mathrm{newVar}()$; $Vars \longleftarrow \{x_0\}$; $P_0 \longleftarrow \mathrm{closure}(\{[S' \to \cdot S, \{x_0\}]\})$;
initialize $Eqs$ to contain the equation $x_0 \doteq \{\$\}$; initialize $St_s$ to contain $P_0$;
set $P_0$ unmarked;
while there is some unmarked state in $St_s$ do
    foreach unmarked state $P$ in $St_s$ do
        foreach grammar symbol $Y$ do
            $tmp \longleftarrow \emptyset$;
            foreach $[A \to \alpha \cdot Y\beta, \Delta]$ in $P$ do
                add $[A \to \alpha Y \cdot \beta, \Delta]$ to $tmp$;
            if $tmp \neq \emptyset$ then
                if $\mathrm{prj}(tmp) = \mathrm{prj}(\mathrm{kernel}(Q))$ for some $Q$ in $St_s$ then /* Refine */
                    foreach pair
                    $([A \to \alpha Y \cdot \beta, \Gamma] \in \mathrm{kernel}(Q)$ , $[A \to \alpha Y \cdot \beta, \Delta] \in tmp)$ do
                        if $\beta = \epsilon$ then
                            update $[A \to \alpha Y \cdot, \Gamma]$ to $[A \to \alpha Y \cdot, \Gamma \cup \Delta]$ in $\mathrm{kernel}(Q)$;
                        else if $\Gamma = \{x\}$ and $(x \doteq \Delta_1) \in Eqs$ then
                            update $(x \doteq \Delta_1)$ to $(x \doteq \Delta_1 \cup \Delta)$ in $Eqs$;
                        $\tau_s(P, Y) \longleftarrow Q$;
                else /* Generate */
                    foreach $[A \to \alpha Y \cdot \beta, \Delta] \in tmp$ such that $\beta \neq \epsilon$ do
                        $x \longleftarrow \mathrm{newVar}()$;
                        $Vars \longleftarrow Vars \cup \{x\}$;
                        enqueue $(x \doteq \Delta)$ into $Eqs$;
                        change $[A \to \alpha Y \cdot \beta, \Delta]$ into $[A \to \alpha Y \cdot \beta, \{x\}]$ in $tmp$;
                    $\tau_s(P, Y) \longleftarrow \mathrm{closure}(tmp)$;
                    add $\tau_s$(P , Y ) to $St_s$ as an unmarked state;
        mark state $P$ ;
```

Algorithm 1: Construction of $\mathscr{A}_s$ for $\mathcal{G} = (V, T, S, \mathcal{P})$

The algorithm for the construction of the symbolic characteristic automaton for $\mathcal{G}$ is reported as Alg. 1. The collection of states is initialized to contain the initial state P_0, which is defined as the closure of $\{[S' \to \cdot S, \{x_0\}]\}$ where x_0 is a fresh variable. Correspondingly, *Eqs* is let to contain the equation $x_0 \doteq \{\$\}$. The generation of further states goes together with the incremental definition of the transition function. For every state P already found, and for all the grammar symbols Y such that some bypassing item $[A \to \alpha \cdot Y\beta, \Delta]$ is in P, a temporary set *tmp* is computed. Such a set represents, in the LR(0) sense, the kernel of $\tau_s(P, Y)$, and is used to check – irrespectively of lookahead-sets – whether the target state for the Y-transition from P has already been collected or not.

If the wanted target $\tau_s(P, Y)$ has not been collected yet, then a new state, say P', is created by closing up a set which is derived from *tmp* as follows. Reducing items of *tmp* are left untouched. On the other hand, each bypassing item is given a lookahead-set containing a fresh variable. Also, an equation for each such variable is installed in *Eqs* to record the lookahead-set carried by *tmp* from the corresponding item of P. The closure procedure is then applied to the modified instance of the kernel set *tmp*. This ensures the possible propagation of variables, and hence of symbolic lookaheads, to the closure items of P'. As an example, when we start running the algorithm for $\mathcal{G}_1$, we get $[S_1 \to \cdot L = R, \{x_0\}], [L \to \cdot * R, \{=, x_0\}] \in P_0$. By that, $[S_1 \to L \cdot = R, \{x_1\}] \in \tau_s(P_0, L)$, and $[L \to * \cdot R, \{x_2\}] \in \tau_s(P_0, *)$, with *Eqs* provisionally containing the equations $x_0 \doteq \{\$\}$, $x_1 \doteq \{x_0\}$, and $x_2 \doteq \{=, x_0\}$.

Either the states already generated or the equations which have been installed for them can still undergo refinements. This happens when the collected state P' is recognized, again in the LR(0) sense, as the target of the Y-transition from yet another state, say P''. If this is the case, then both the reducing kernel items of P' and the right-hand sides of the equations installed for the kernel bypassing items of P' are treated as accumulators, to record the contributions coming from the items in P''. No further modification is applied to P', nor a closure procedure invoked. Essentially, multiple incoming edges to the same state bring in multiple contributions, and all of them are encoded either by equations over variables or in the lookahead-sets of reducing kernel items. For instance, upon termination of the construction of the automaton for $\mathcal{G}_1$, *Eqs* is given by $E_1 = \langle x_0 \doteq \{\$\}, x_1 \doteq \{x_0\}, x_2 \doteq \{=, x_0, x_2, x_3\}, x_3 \doteq \{x_1\}\rangle$, where x_3 is the variable installed for the kernel item $[S \to L = \cdot R, \{x_3\}]$ of the state generated as $\tau_s(\tau_s(P_0, L), =)$.

A few basic properties of symbolic characteristic automata follow. An easy consequence of the technique used for the construction of symbolic automata is that, seen as graphs, $\mathscr{A}_s$ and the LR(0)-automaton for $\mathcal{G}$ are isomorphic. Moreover, the items contained in each state of the LR(0)-automaton are just the projections of the symbolic items in the corresponding state of $\mathscr{A}_s$. Lemma 1 below characterizes the rôle of variables in the symbolic construction. The lookahead-set of every bypassing kernel item of every state is a singleton set consisting of a distinct variable. Hence, each variable implicitly identifies a pair consisting of a bypassing kernel item and of the state the item belongs to.

▶ **Lemma 1.** *Let $\mathscr{A}_s$ be the symbolic characteristic automaton for $\mathcal{G}$, and let $P, P' \in St_s$. Also, assume that $[A \to \alpha \cdot Y\beta, \Delta] \in P$ and $[A' \to \alpha' \cdot Y'\beta', \Delta'] \in P'$ are such that $A \to \alpha \cdot Y\beta = A' \to \alpha' \cdot Y'\beta'$ implies $P \neq P'$. Then, for some x, x' with $x \neq x'$, $\Delta = \{x\}$ and $\Delta' = \{x'\}$.*

We observe that, by Lemma 1 and by definition of closure, if the lookahead-sets of the items not in the kernel contain any variable, then such a variable must be one of those installed for the bypassing items in the kernel of the same state.

The next lemma accounts for the properties of the equations for the variables identifying bypassing kernel items. The single equation installed for the initial state P_0 remains unchanged throughout the computation of the whole automaton. Hence, there is at least one equation with a ground right-hand side in *Eqs*. Moreover, every equation $x \doteq \Delta$ installed for the bypassing kernel item of a state $P \neq P_0$ is such that Δ collects contributions from all the predecessors of P. The set Δ can contain x itself, due, e.g., to a self-loop on P in the graph. By construction, however, the first provisional versions of every state P and of the relative equations are generated when processing the possible transitions of a state collected before P. So, even if Δ can contain x, it can never contain x alone.

▶ **Lemma 2.** *Let $\mathscr{A}_s$ be the symbolic characteristic automaton for $\mathcal{G}$.*

1. *If $[S' \to \cdot S, \{x\}] \in P_0$ then the equation for x in Eqs is $x \doteq \{\$\}$.*

2. *Assume that $P \in St_s \setminus \{P_0\}$, and that $[A \to \alpha Y \cdot \beta, \{x\}] \in \mathrm{kernel}(P)$ is a bypassing item. Also, let $x \doteq \Delta$ be the equation for x in Eqs. Then the following holds.*
   - *Let $\mathbb{D} = \{\Delta_i \mid [A \to \alpha \cdot Y\beta, \Delta_i] \in Q_i$ for $Q_i \in St_s$ such that $\tau_s(Q_i, Y) = P\}$. Then $\Delta = \bigcup_{\Delta_i \in \mathbb{D}} \Delta_i$.*
   - *$\Delta \setminus \{x\} \neq \emptyset$. Also, if $\Delta \setminus \{x\} = \{x'\}$ then $x' \prec x$.*

The following lemma, dual to Lemma 2, states the main properties of the lookahead-sets of kernel reducing items.

▶ **Lemma 3.** *Let $\mathscr{A}_s$ be the symbolic characteristic automaton for $\mathcal{G}$. Assume that $P \in St_s$, and that $[A \to \alpha Y \cdot, \Delta] \in \mathrm{kernel}(P)$. Also, let $\mathbb{D} = \{\Delta_i \mid [A \to \alpha \cdot Y, \Delta_i] \in Q_i$ for $Q_i \in St_s$ such that $\tau_s(Q_i, Y) = P\}$. Then $P \neq P_0$, $\Delta \neq \emptyset$, and $\Delta = \bigcup_{\Delta_i \in \mathbb{D}} \Delta_i$.*

4 LALR(1) tables

In this section we will first focus on the symbolic correspondence existing between $\mathscr{A}_s$ and the LRm(1)-automaton for the given grammar $\mathcal{G}$. Then, to make this correspondence concrete, we will show how *Eqs* can be resolved.

A key point for the proof of the symbolic correspondence between $\mathscr{A}_s$ and $\mathscr{A}_m$ is the relation between the states of the symbolic automaton and those of the LR(1)-automaton for $\mathcal{G}$. In order to capture such a relation, an appropriate handle on the propagation flow of ground lookaheads through the defining equations in *Eqs* is needed. To this end, we introduce a notion of reachability which relates ground lookaheads to lookahead-sets via equations. Intuitively, we say that the lookahead l reaches Δ if either $l \in \Delta$ or if l is in the right-hand side of the equation for some x, and x propagates, through a chain of uses and definitions, to a variable belonging to Δ. Consider for instance the system of equations $E_1 = \langle x_0 \doteq \{\$\}, x_1 \doteq \{x_0\}, x_2 \doteq \{=, x_0, x_2, x_3\}, x_3 \doteq \{x_1\}\rangle$ mentioned in Sec. 3. In E_1 the ground lookahead \$ reaches $\{\$\}$ simply because it is contained in the set. Moreover, there is a sequence of hops from right-hand sides to left-hand sides of equations which takes, e.g., \$ to x_3. In fact, \$ is used in the definition of x_0, which is used in the definition of x_1, which is used in the definition of x_3. By that, we conclude that \$ reaches, among the rest, every lookahead-set Δ containing x_3. Def. 4 below captures this intuition.

▶ **Definition 4.** Let $E = \{x_i \doteq \Delta_i\}_i$ be a set of defining equations for the variables in the finite subset $\{x_i\}_i$ of $\mathbb{V}$. Also, let $l \in T \cup \{\$\}$. Then:
- x_j gets x_k in E, written $x_j \text{ gets}_E\, x_k$, iff $x_j, x_k \in \{x_i\}_i$ and $x_k \in \Delta_j$;
- Δ takes l in E, written $\Delta \text{ takes}_E\, l$, iff $l \in \Delta$ or $x_j, x_k \in \{x_i\}_i$ exist such that $x_j \in \Delta$ and $x_j \text{ gets}^*_E\, x_k$ and $l \in \Delta_k$.

Next, we show in what respect the membership of the LR(1)-item $[A \to \alpha \cdot \beta, l]$ in a state of $\mathscr{A}_l$ is related to the membership of $[A \to \alpha \cdot \beta, \Delta]$, with l taken to Δ in *Eqs*, in a state of $\mathscr{A}_s$. The asymmetry of the statements of Lemma 5 and Lemma 6 below is due to the existence of a one-to-many correspondence between the states of $\mathscr{A}_s$ and those of $\mathscr{A}_l$. In fact, each state of $\mathscr{A}_l$ is simulated by an appropriate cut of *Eqs* and of the lookahead-sets of the reducing items of a state of $\mathscr{A}_s$. On the other hand, however, each state $P \in \mathscr{A}_s$, together with *Eqs*, stands for all the states of $\mathscr{A}_l$ whose projection is the same as the projection of P. So, if $[A \to \alpha \cdot \beta, \Delta] \in P$, then not all the states of the LR(1)-automaton with projection $\mathrm{prj}(P)$ necessarily contain the pairing of $A \to \alpha \cdot \beta$ with each of the lookahead l that are taken to Δ.

The proofs of the lemmata below crucially rely upon the following key issue. Both in $\mathscr{A}_l$ and in $\mathscr{A}_s$, the lookahead \$ is generated by the single kernel item of their initial state. All the other lookaheads show up by the application of the closure procedure, and this depends, in either automata and in corresponding ways, on the projection of the item undergoing closure. Once a lookahead has been generated, it propagates along the paths of the two automata in lock-step fashion.

▶ **Lemma 5.** *Let $\mathscr{A}_l$ and $\mathscr{A}_s$ be the LR(1)-automaton and the symbolic automaton for $\mathcal{G}$, respectively. Then for every $L \in St_l$ there exists $P \in St_s$ such that $\text{prj}(P) = \text{prj}(L)$, and, for every $[A \to \alpha \cdot \beta, l] \in L$, if $[A \to \alpha \cdot \beta, \Delta] \in P$ then Δ takes_{Eqs} l.*

Proof sketch. By construction of $\mathscr{A}_s$ and of $\mathscr{A}_l$, if we walk on both automata a path starting from the initial state and labelled by some γ, we reach states with equal projections. Also, given any $L \in St_l$, there is a unique state $P \in St_s$ such that $\text{prj}(P) = \text{prj}(L)$. Hence, all the lookaheads carried to the items of L are also carried, through the corresponding paths, to the relevant items of that state P. ◀

▶ **Lemma 6.** *Let $\mathscr{A}_s$ and $\mathscr{A}_l$ be the symbolic automaton and the LR(1)-automaton for $\mathcal{G}$, respectively. Also, let $[A \to \alpha \cdot \beta, \Delta] \in P \in St_s$, and let l be such that Δ takes_{Eqs} l. Then there exists $L \in St_l$ such that $\text{prj}(L) = \text{prj}(P)$ and $[A \to \alpha \cdot \beta, l] \in L$.*

Proof sketch. By the assumption that Δ takes_{Eqs} l, there is at least one state $P_g \in St_s$ which contains an item, precisely related to $[A \to \alpha \cdot \beta, \Delta]$, that generates l. If $l = \$$, then $P_g = P_0$ and the generating item is the kernel item of P_0. Otherwise, the generating item has the form $[B \to \cdot\delta, \Gamma]$ and $l \in \Gamma$. In either case, for some γ_1, there is a γ_1-path from P_g to P. The string γ_1 can be traced backwards from P to P_g, and depends both on the chain of hops among variables which takes l to Δ, and on the structure of the items found along the way. Now, let γ be a path from P_0 to P_g. By construction of $\mathscr{A}_l$, the state L reached from L_0 by the path $\gamma\gamma_1$ has the same projection as that of P, and contains the item $[A \to \alpha \cdot \beta, l]$. ◀

The relation between the states of $\mathscr{A}_s$ and of $\mathscr{A}_l$ is at the basis of the symbolic correspondence between $\mathscr{A}_s$ and $\mathscr{A}_m$ which is stated by the following theorem.

▶ **Theorem 7.** *Let $\mathscr{A}_s$ and $\mathscr{A}_m$ be the symbolic automaton and the LRm(1)-automaton for $\mathcal{G}$, respectively. Then the following holds.*

- *For every $P \in St_s$ there exists $M \in St_m$ such that $\text{prj}(M) = \text{prj}(P)$, and, for every Y, if $\tau_s(P, Y) = P'$ then $\tau_m(M, Y) = M'$ with M' such that $\text{prj}(M') = \text{prj}(P')$.*
- *For every $M \in St_m$ there exists $P \in St_s$ such that $\text{prj}(P) = \text{prj}(M)$, and, for every Y, if $\tau_m(M, Y) = M'$ then $\tau_s(P, Y) = P'$ with P' such that $\text{prj}(P') = \text{prj}(M')$.*
- *If $P \in St_s$ and $M \in St_m$ are such that $\text{prj}(P) = \text{prj}(M)$, then $[A \to \alpha \cdot \beta, l] \in M$ iff $[A \to \alpha \cdot \beta, \Delta] \in P$ and Δ takes_E l.*

Proof. The first two assertions are consequences of the construction procedures used to obtain $\mathscr{A}_s$ and $\mathscr{A}_m$, which can both be projected into the LR(0)-automaton for $\mathcal{G}$. The third assertion comes from Lemma 5 and Lemma 6, by construction of $\mathscr{A}_m$ from $\mathscr{A}_l$. ◀

To set up the the LALR(1) parsing table that we want to construct, we still need to compute the actual lookahead-set of reducing items. Suppose that $Vars = \{x_0, \ldots, x_n\}$ and $Eqs = \{x_i \doteq \Delta_i\}_{i=0,\ldots,n}$. Our goal is to compute the set of actual instantiations of $x_0, \ldots, x_n$, hereby called $val(x_0), \ldots, val(x_n)$. By definition of takes_{Eqs}, $val(x_i)$ is given by the union of ground(Δ_i) with $val(x_k)$, for all the variables x_k such that x_i gets$_{Eqs}$ x_k. Hence, we actually look for the solution of a system of recursive equations of the form

$$val(x_i) = \text{ground}(\Delta_i) \cup \bigcup_{x_k \,:\, x_i \text{gets}_{Eqs} x_k} val(x_k)\,.$$

We observe that $D = (2^{T \cup \{\$\}})^{n+1}$ is a cpo with least element, and that $val : D \to D$ is a monotone function. Relying on standard approximation techniques, we can prove that the least solution of the system of recursive equations for $val(x_i)$ is given by

$$val(x_i) = \bigcup_{x_k \,:\, x_i \text{gets}^*_{Eqs} x_k} \text{ground}(\Delta_k)\,.$$

To gain in efficiency, instead of computing the above solution for all the variables in *Vars*, we first partition variables into equivalence classes. This allows us to define a reduced system of equations *REqs* which induces a reachability relation of smaller size over a relevant subset *RVars* of *Vars*. The intuition behind this reduction is that characteristic automata are typically quite sparse, and lookahead propagation is usually preponderant over lookahead generation. So, in general *Eqs* is expected to contain many equations of the shape $x_i \doteq \{x_j\}$. An obvious optimization is computing only one of $val(x_i)$ and $val(x_j)$ and then, by need, copying it into the other.

```
inizialize $RVars$ and $REqs$ to $\emptyset$ ;
while  $Eqs$ not empty do
    $x \doteq \Delta \longleftarrow$ dequeue($Eqs$) ;
    if  $\Delta \setminus \{x\} = \{x'\}$ then
        $class(x) \longleftarrow class(x')$ ;
    else
        $class(x) \longleftarrow x$ ;
        add $x$ to $RVars$ ;
foreach  $x \in RVars$ such that $x \doteq \Delta \in Eqs$ do
    update each $x'$ in $\Delta$ to $class(x')$ ;
    add $x \doteq \Delta \setminus \{x\}$ to $REqs$ ;
```
Algorithm 2: Reduced system of equations *REqs* for the variables in $RVars \subseteq Vars$

The algorithm for the computation of *REqs* is reported as Alg. 2. For every $x \in Vars$ we record the membership of the variable into an equivalence class. The equations in *Eqs* are processed one at a time exploiting the generation order of the variables in *Vars*. We first check whether $\Delta \setminus \{x\} = \{x'\}$. If so, then the equation at hand has either the shape $x \doteq \{x'\}$ or the shape $x \doteq \{x', x\}$. Hence the variable x is reached, in *Eqs*, exactly by the same ground values that reach x', and we let both variables belong to same equivalence class. Moreover, by Lemma 2, if $\Delta \setminus \{x\} = \{x'\}$ then $x' \prec x$. Hence, by the ordering of equation processing, $class(x')$ has already been set when handling the equation for x. Once the set of representative variables *RVars* has been identified, we start populating *REqs*. Suppose $x \in RVars$, and assume that $x \doteq \Delta \in Eqs$. Then the equation installed into *REqs* for x is obtained by updating Δ as follows. First, non-representive variables in var(Δ) are replaced by the corresponding class representative. Second, the possible occurrence of x is removed from the resulting set. This is a further optimization that, as done above, amounts to disregard a possible self-recurrence in the computation of $val(x)$. As an example, the system of equations $E_1 = \langle x_0 \doteq \{\$\}, x_1 \doteq \{x_0\}, x_2 \doteq \{=, x_0, x_2, x_3\}, x_3 \doteq \{x_1\}\rangle$ for grammar $\mathcal{G}_1$ is reduced to $\langle x_0 \doteq \{\$\}, x_2 \doteq \{=, x_0\}\rangle$.

The following theorem is a consequence of the resolution strategy for Eqs that we have illustrated so far.

▶ **Theorem 8.** *Let $\mathscr{A}_s$ be the symbolic automaton for $\mathcal{G}$, $x \in Vars = \{x_i\}_i$, and $Eqs = \{x_i \doteq \Delta_i\}_i$. Also, let $RVars$, $REqs$, and $class(x)$ be as computed by Alg. 2. Then, for every $x_i \in RVars$, $val(x_i) = \bigcup_{x_k \,:\, x_i \mathrm{gets}^*_{REqs} x_k} \mathrm{ground}(\Delta_k)$, and, for every $x_i \in Vars \setminus RVars$, $val(x_i) = val(class(x_i))$.*

Commenting on the complexity of the resolution of Eqs, we notice that Alg. 2 is linear in the size of $Vars$, and that $val(x_i)$ can be efficiently computed by a depth-first search algorithm run on the dependency graph of the relation gets_{REqs}. Briefly, each node x_i of the graph can be initially associated with the value $\mathrm{ground}(\Delta_i)$. Then the graph is visited and the values associated with the farthest nodes are accumulated with the values of the nodes found along the way back to the origin of the path. The visit can be organized in such a way that strongly connected components, if any, are recognized on-the-fly and traversed only once (see, e.g., [13, 8, 5]). By that, the search algorithm is linear in the size of the dependency graph of the relation gets_{REqs}. Referring again to $\mathcal{G}_1$, the computation of the actual lookahead-sets goes through the depth-first visit of the dependency graph

$$\begin{tikzcd} \llbracket x_2 \rrbracket \arrow[r] & \llbracket x_0 \rrbracket \end{tikzcd}$$

where $\llbracket x_0 \rrbracket$ and $\llbracket x_2 \rrbracket$ stand for the equivalence classes of x_0 and of x_2, respectively.

The following result, which is a consequence of Theorem 7 and of Theorem 8, concludes the section by concretizing the symbolic correspondence between $\mathscr{A}_s$ and $\mathscr{A}_m$.

▶ **Theorem 9.** *Let $\mathscr{A}_s$ be the symbolic automaton for $\mathcal{G}$. Also, for $P \in F_s$ and $[A \to \beta\cdot, \Delta] \in P$, let*

$$\mathcal{LA}(P, [A \to \beta\cdot, \Delta]) = \mathrm{ground}(\Delta) \cup \bigcup_{x \,:\, x' \in \Delta \text{ and } x = class(x')} val(x)\,.$$

Then the pair consisting of $\mathscr{A}_s$ and of $\{\mathcal{LA}(P, [A \to \beta\cdot, \Delta])\}_{P \in F_s, [A \to \beta\cdot, \Delta] \in P}$ is an LALR(1) parsing table for $\mathcal{G}$.

5 LR(1) tables

Below we will briefly overview the strategy we propose for deciding whether the grammar at hand is LR(1), and for constructing a compact LR(1) parsing table in the positive case.

We adopt an optimistic approach, and build an LR(1) parsing table by appropriately expanding the LALR(1) table. The cases when the LALR(1) table does not contain any conflict, or only contains shift/reduce (s/r) conflicts are equally not relevant for our argument. Indeed, in the first case the grammar is LALR(1), and in the second case it is surely not LR(1). So, we focus on the reduce/reduce (r/r) conflicts of the LALR(1) table. If the grammar at hand is not LR(1), then at least one of these conflicts is a genuine LR(1) conflict. This is the case, e.g., for the r/r conflict of the table for the ambiguous grammar $\mathcal{G}_2$ with start symbol S_2 and productions in the set $\{S_2 \to Ab \mid Bb, A \to a, B \to a\}$. If instead the grammar is LR(1), then the r/r conflicts in the LALR(1) table depend on the fact that the procedure for the construction of $\mathscr{A}_s$ caused the merging of states which would have remained separated in the construction of the LR(1)-automaton. Our goal

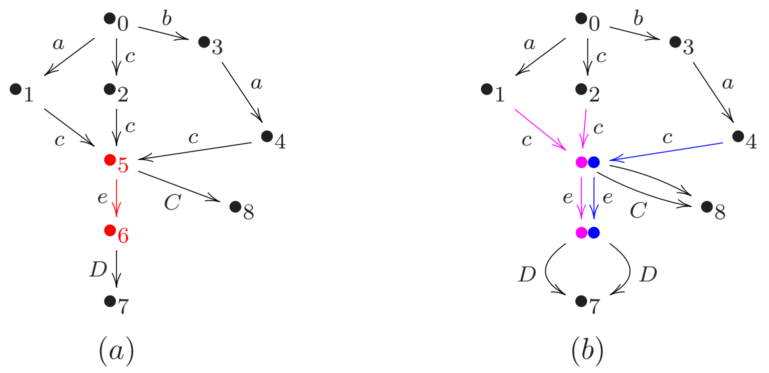


Figure 1 Partial layout of the symbolic automaton for $\mathcal{G}_3$ before (a) and after (b) the splitting due to the r/r conflict at state 6.

is eliminating that sort of r/r conflicts by appropriately splitting the *critical states* that cause those spurious r/r conflicts. Suppose that the symbolic state P is one of such merged states of $\mathscr{A}_s$, and that P stands for the union of the LR(1)-states $L_1, \ldots, L_k$. Also, assume that $P = \{[A_i \to \alpha_i \cdot \beta_i, \Delta_i]\}_i$, and that the instantiation of variables in the lookahead-sets transforms P to $P_{val} = \{[A_i \to \alpha_i \cdot \beta_i, \Sigma_i]\}_i$. Then, there exist k cuts of P_{val} of the form $P_{val_j} = \{[A_i \to \alpha_i \cdot \beta_i, \Sigma_{ij}]\}_i$ where $\bigcup_{j=1,\ldots,k} \Sigma_{ij} = \Sigma_i$, and each P_{val_j} plays the LR(1)-state L_j. Modulo an appropriate tuning of the automaton transition function, a way out for the elimination of the original r/r conflict would be exploding P_{val} in $P_{val_1}, \ldots, P_{val_k}$, and accordingly replicating the subgraph rooted at P_{val}. Two observations are in place here, both related to the space complexity of the resulting structure. Splitting P_{val} in k states might be an overkill, because, e.g., the conflict would be eliminated as well by letting P_{val_1} be still merged with P_{val_2}. For analogous reasons, replicating the whole subgraph rooted at P_{val} can be a waste, too.

We aim at applying the procedure hinted above while limiting the replication of states as much at possible. In the overall, we take the following steps. We start analyzing the r/r conflicts of the LALR(1) table, and check whether the conditions to eliminate them are met or not. If we find any r/r conflict that cannot be eliminated, we infer that the grammar is not LR(1) and conclude. Otherwise, we apply an optimized state splitting procedure to the states involved in the r/r conflicts. Operationally, this is performed by replicating some of the rows of the LALR(1) table, up to minor modifications to their contents. At worst, the resulting table has the same size as the LR(1) table built from the LR(1)-automaton.

A crucial issue in the application of the procedure is the ability to identify critical states. Also, when a state P is recognized as critical, a key point is how we actually split it under the guarantee that the intended LR(1) behaviour is preserved. The lookahead propagation flow embedded by gets_{Eqs} provides the needed support. We describe below the technique for the identification of critical states. The very first step towards this end is locating the states where the lookaheads leading to conflicts, called *critical lookaheads*, are actually generated. Assume we are considering an r/r conflict at state Q for the critical lookahead d. Also, suppose that the symbolic lookahead-sets associated with the conflicting reducing items get instantiated to Σ and to Σ'. There are various combinations here, depending on whether d is a ground element of one or both of the symbolic lookahead-sets, or it is the by-product of variable instantiation. Here we consider this last case which is the most intricate one. An example of this scenario is given by the LR(1) grammar $\mathcal{G}_3$ with start symbol S_3 and production set $\mathcal{P}_3 = \{S_3 \to aAd \mid aBc \mid baAe \mid baBd \mid cAd \mid cBc,\ A \to ce,\ B \to cC,\ C \to eD, D \to \epsilon\}$. The relevant portion of the layout of the symbolic automaton for $\mathcal{G}_3$ is drawn in Fig. 1(a), where 0 is the initial state. The r/r conflict for d is at state 6 and is induced by the reducing items $[A \to ce\cdot, \{x_7\}]$ and $[D \to \cdot, \{x_{11}\}]$ where $x_7 \doteq \{d, e\}$, $x_{11} \doteq \{x_8\}$, and $x_8 \doteq \{c, d\}$ are the relevant equations in Eqs. So, for this specific instance of state Q, $\Sigma = \{d, e\}$ and $\Sigma' = \{c, d\}$. By an analysis of gets^*_{Eqs}, we infer that the lookaheads d and e for $A \to ce\cdot$ come from the kernel item which owns x_7, say i_7, and the lookaheads c and d for $D \to \cdot$ come from the kernel item which owns x_8, say i_8 (both i_7 and i_8 are located at state 5). The items that actually generate the critical lookaheads are located in the predecessors of the state containing i_7 and i_8 (states 1, 2, and 4), and their identity can be inferred by inspection of the projections of i_7 and i_8.

Once the states generating the critical lookaheads have been identified, we check whether the r/r conflict in Q is either genuine or spurious. In the second case, we also decide which is the best possible split of Q. First, we set up two sets of pairs, say $loc(\Sigma)$ and $loc(\Sigma')$, to associate each lookahead in Σ and in Σ' with the state where the lookahead is actually generated. Then, for all the pairs in $loc(\Sigma)$ and $loc(\Sigma')$ that share the same lookahead, we deduce that the associated source states Q_1 and Q_2 are in conflict, written $Q_1 \# Q_2$. E.g., for the running example, $loc(\Sigma) = \{(d, 1), (d, 2), (e, 4)\}$, $loc(\Sigma') = \{(c, 1), (c, 2), (d, 4)\}$, and, by $(d, 1), (d, 2) \in loc(\Sigma)$ and $(d, 4) \in loc(\Sigma')$, we infer $1\#4$, and $2\#4$. The conflict relation $\#$ is the basic tool for deciding whether the r/r conflict at hand is a genuine LR(1) conflict or not. The intuition here is that if the r/r conflict at state Q is spurious, then, as discussed above, it must be possible finding cuts of Q that match the lookaheads contributed by the various merged states. Operationally, we check whether the source states occurring in $loc(\Sigma)$ and in $loc(\Sigma')$ can be partitioned in at least two groups of maximal size so that each group G of the partition meets the following requirements: (i) G contains non-conflicting states; (ii) the restriction of $loc(\Sigma)$ to the pairs whose second component is in G, written $loc(\Sigma)\restriction G$, is non empty; (iii) $loc(\Sigma')\restriction G$ is non empty either. For the example at hand, we end up partitioning $\{1, 2, 4\}$ in the two groups $\{1, 2\}$ and $\{4\}$.

If the above partition cannot be found, then the analyzed r/r conflict in P is a genuine LR(1) conflict, and we conclude that the grammar is not LR(1). This is always the case, e.g., if the conflict relation contains a pair $R\#R$. This conflict reveals that a critical lookahead is generated by distinct items of the state R. Hence this particular lookahead directly depends on the projection of R, which is the same either in $\mathscr{A}_s$ or in $\mathscr{A}_l$. (An instance of this scenario is found in the analysis of the ambiguous grammar $\mathcal{G}_2$ for the critical lookahead b that is generated in the initial state for either $A \to \cdot a$ or $B \to \cdot a$.)

If the source states of the actual lookahead-sets of Q can be partioned into the groups $G_1, \ldots, G_j$, then Q is split in j replica. For each actualized lookahead-set Σ_i, the jth replica of Q is assigned the lookahead-set containing the first elements of the pairs in $\Sigma_i\restriction G_j$. At the same time, relying upon gets^*_{Eqs} we identify the paths $\gamma_1, \ldots, \gamma_m$ that start at the states where the critical lookaheads are sourced and that lead to Q. Such paths share at least one state that is Q at latest. Among the states traversed by $\gamma_1, \ldots, \gamma_m$, those states which are shared by paths from conflicting source states are all critical (states in red in Fig. 1(a)). We split them to grant distinct and parallel routes to the contributions from the states of each group G_j to the corresponding replica of Q. As for the elimination of the spurious r/r conflict at hand, no other state needs to be replicated.

Looking at the splitting procedure from the perspective of the parsing table, the modifications performed are as follows. The row for state Q is copied j times, and each copy retains only the reduction steps for the represented group. The row for each replica of the other critical states is copied from the row of the replicated state. The shift moves of the non-conflicting predecessors of critical states (edges in magenta and in blue in Fig. 1(b)) are redirected to the replica for the appropriate group. Here we notice that each new row of the table has shift moves exactly for the same symbols as the old copy of the row, and fewer reduce moves. So, each pass of the splitting procedure cannot generate neither new s/r conflict nor new r/r conflicts.

The described approach for the construction of LR(1) tables guarantees the early detection that the grammar is not LR(1). In the opposite case, when all the r/r conflicts are eliminated, we get a table for LR(1) parsing where all the states whose merging is not critical are still merged as in an LALR(1) table. The generated table is then expected to be generally smaller than the table constructed on top of LR(1)-automata. An easy example of this is the size of the LR(1) table that we obtain by applying the above algorithm to the grammar $\mathcal{G}_4$ with start symbol S_4 and with production set $\{S_4 \to S_1 \mid S_3\} \cup \mathcal{P}_1 \cup \mathcal{P}_3$. The symbolic automaton for $\mathcal{G}_4$ has two separated sub-graphs corresponding to the symbolic automata for $\mathcal{G}_1$ and for $\mathcal{G}_3$, respectively. The subgraph representing the symbolic automaton for $\mathcal{G}_3$ is refined as described above. The sub-graph for $\mathcal{G}_1$, though, remains untouched, and hence definitely smaller than the LR(1)-automaton for $\mathcal{G}_1$.

6 Concluding remarks

We defined symbolic characteristic automata, and used them as the basis for the construction of LALR(1) parsing tables, for the construction of LR(1) parsing tables, and for early detecting that grammars are non LR(1).

Among the algorithms for the construction of LALR(1) parsing tables, the most popular ones are the `Yacc` algorithm [9, 2], and the algorithm by DeRemer and Pennello [5]. The algorithm we proposed is more similar to the `Yacc` algorithm than to the algorithm by DeRemer and Pennello. In fact, our technique retains, although making it symbolic, the `Yacc` strategy of generating/propagating LR(1) lookaheads. The approach taken by DeRemer and Pennello is instead a refinement of the SLR(1) technique. In a nutshell, the algorithm by DeRemer and Pennello elaborates on the state of the LR(0)-automaton where $A \to \beta\cdot$ is located, and infers which precise subset of the productions of the grammar should be considered when computing the follow-set of that specific occurrence of A.

In [11], Pager presented an algorithm that is used in the implementation of Menhir [7], the parsing engine of OCaml [10]. The algorithm by Pager generates on-the-fly a compact LR(1)-automaton by checking whether already generated states can be the target of the processed transition. We adopt a quite different strategy. Driven by local reasoning on r/r conflicts, we construct LR(1) parsing tables as refinements of the corresponding LALR(1) tables. This delays as much as possible any sort of check on the content of states.

In the algorithms for the construction of parsing tables, the number of set-union operations on lookahead-sets is typically taken as performance measure. Distinct algorithms execute those operations on different kinds of auxiliary structures, and the size of these structures is often grammar-dependent. So, even statistical reasoning about performance, which in many cases is likely as much as we can do, has to be very carefully tuned. Precise comparisons between our algorithms and those mentioned above are untimely at this stage, as they should be based on large test-sets.

The symbolic techniques we presented can be extended to produce parsing tables for grammars in classes bigger than LALR(1). E.g., we used it to define LALR(k) parsing tables for $k > 1$. The major benefit of the symbolic structures we proposed is, however, that equations over variables, together with the fact that each variable uniquely identifies an item, provide a compact and synthetic feedback on the origin of lookaheads, on their flow, and on their inter-dependency. The explicit representation of lookahead propagation can be most useful in the design phase of grammars, i.e. especially when the grammar under investigation is not yet in the wanted class. Orthogonally, the algorithm we presented for upgrading LALR(1) tables to LR(1) tables shows another sort of application of the explicit encoding of lookahead propagation in LALR(1)-automata.

References

1 Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Prentice Hall, 2006.

2 Alfred V. Aho and Jeffrey D. Ullman. *Principles of Compiler Design*. Addison-Wesley, 1977.

3 Frank DeRemer. *Practical Translators for LR(k) Languages*. PhD thesis, MIT, Cambridge, Mass., 1969.

4 Frank DeRemer. Simple LR(k) Grammars. *Commun. ACM*, 14(7):453–460, 1971. `doi:10.1145/362619.362625`.

5 Frank DeRemer and Thomas J. Pennello. Efficient Computation of LALR(1) Look-Ahead Sets. *ACM Trans. Program. Lang. Syst.*, 4(4):615–649, 1982. `doi:10.1145/69622.357187`.

6 Charles Donnelly and Richard Stallman. Bison: The Yacc-compatible Parser Generator (Ver. 3.0.4). 2015. URL: `http://www.gnu.org/software/bison/manual/bison.pdf`.

7 François Pottier et Yann Régis-Gianas. Menhir. URL: `http://pauillac.inria.fr/~fpottier/menhir/menhir.html.fr`.

8 J. Eve and Reino Kurki-Suonio. On Computing the Transitive Closure of a Relation. *Acta Inf.*, 8:303–314, 1977. `doi:10.1007/BF00271339`.

9 Stephen C. Johnson. Yacc: Yet Another Compiler-Compiler. Tech. Rep. CSTR 32, Bell Laboratories, Murray Hill, N.J., 1974. URL: `http://dinosaur.compilertools.net/`.

10 Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The OCaml system release 4.02. 2014. URL: `http://caml.inria.fr/pub/docs/manual-ocaml/`.

11 David Pager. A Practical General Method for Constructing LR(k) Parsers. *Acta Informatica*, 7:249–268, 1977.

12 Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory – Volume II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1990. `doi:10.1007/978-3-662-08424-3`.

13 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. `doi:10.1137/0201010`.

Structural Control in Weighted Voting Games*†

Anja Rey[1] **and Jörg Rothe**[2]

1 **Technische Universität Dortmund, 44221 Dortmund, Germany**
`anja.rey@tu-dortmund.de`

2 **Heinrich-Heine-Universität Düsseldorf, 40225 Düsseldorf, Germany**
`rothe@cs.uni-duesseldorf.de`

Abstract

Inspired by the study of control scenarios in elections and complementing manipulation and bribery settings in cooperative games with transferable utility, we introduce the notion of structural control in weighted voting games. We model two types of influence, adding players to and deleting players from a game, with goals such as increasing a given player's Shapley–Shubik or probabilistic Penrose–Banzhaf index in relation to the original game. We study the computational complexity of the problems of whether such structural changes can achieve the desired effect.

1998 ACM Subject Classification I.2.11 [Distributed Artificial Intelligence] Multiagent Systems, J.4 [Social and Behavioral Sciences] Economics

Keywords and phrases Algorithmic games theory, weighted voting games, structural control, power indices, computational complexity

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.80

1 Introduction

A major task in computational social choice [42, 12, 13] is the complexity analysis of the question of whether a certain form of influence is possible in an election under some voting rule (see, e.g., [42, 13]). Bartholdi et al. [5] introduced and analyzed the notion of manipulation in elections, where one or more voters strategically change their true preference in order to make a distinguished candidate a winner. In a bribery scenario, on the other hand, an external agent tries to pay voters for them to change their votes such that a certain candidate becomes a winner, and the question is whether the briber can be successful within a given budget. This idea has been introduced and analyzed by Faliszewski et al. [24, 25]. In a third model, control, the chair of an election changes the structure of an election by adding, deleting, or partioning either voters or candidates, with the aim of making a distinguished candidate a winner [6]. In addition to these constructive types of control, destructive control – the problem of whether a given candidate can be prevented from being a winner – has also been introduced and studied by Hemaspaandra et al. [32]. Manipulation, bribery, and control have been studied for many voting systems, and we refer the reader to the book chapters by Baumeister and Rothe [10], Conitzer and Walsh [15], and Faliszewski and Rothe [27] for an overview of numerous related results. In a nutshell, whenever successful manipulative actions are possible, a high computational complexity may provide some protection against them, or at least against detecting whether such actions are possible or not for an election.

* A two-page extended abstract of this paper appeared in the proceedings of AAMAS 2016 and has been presented at the workshops CoopMAS 2016 and LOFT 2016, both with informal proceedings.

† This work was supported in part by DFG grant RO-1202/14-2.

Similar ideas have been adapted to other fields, such as manipulation in preference aggregation [21] and manipulation, bribery, and control in judgement aggregation [23, 8, 7] (see the book chapters by Endriss [22] and Baumeister et al. [9] for an overview). In algorithmic game theory, the question of influencing the outcome of a game has also been studied extensively. In particular, for weighted voting games, manipulation by merging a coalition of players to a single player, or by splitting a player into several players in order to increase a player's power, have been introduced by Elkind et al. [2]. Here, "power" refers to the notion of power indices, such as the Penrose–Banzhaf [37, 4, 17] and the Shapley–Shubik index [43], measuring the significance of a player in a game (formal definitions will be given in Section 2). The complexity of beneficial merging, splitting, and annexation[1], e.g., for the Shapley–Shubik index, have been studied by Aziz et al. [1] who show NP-hardness. Faliszewski and Hemaspaandra [26] show that the beneficial merging problem is in PP. Rey and Rothe [40] prove PP-hardness for beneficial merging and splitting. Other forms of manipulation have been studied in weighted voting games. For example, Zuckerman et al. [48] study manipulation of the quota. From an algorithmic point of view, this is different from our model: In their model, the number of players and thus the denominator in a power index (see Equations (1) and (2) in Section 2) remains the same but the same coalitions can have different success due to different quotas, whereas with structural control the number of players varies but all coalitions that remain in the game are equally successful before and after the change. Relatedly, Zick et al. [46, 47] study algorithmic properties of the quota. In dynamic weighted voting games, as presented by Elkind et al. [18], the quota is changed as well, but dynamically over time. The notion of bribery has been adapted from voting theory to a model in cooperative game theory, so-called path-disruption games [3], where an external player tries to bribe a coalition of players so as to reach a target node in a graph [41]. Another perspective of persuasion for weighted voting games has been studied by Freixas and Pons [30].

Inspired by the notion of control in elections, we consider control scenarios in weighted voting games. We define the problems of whether it is possible to change the structure of a game by either *adding* or *deleting* players in order to achieve certain goals. One could, for instance, think of a committee that needs a certain quota of votes so as to decide upon an issue. In order to increase the significance of some participant, an organizer might invite further participants or might choose a certain meeting schedule to make sure that originally existing participants are excluded. These structural changes could also be viewed as a change of the players' participation over time without malicious intentions. Goals include *increasing* and *decreasing* the power of a distinguished player, in relation to the player's power in the original game. Increasing and decreasing power in a game by adding or deleting players can be seen as analogues of, respectively, constructive and destructive control in elections by adding or deleting either candidates or voters. Moreover, if an *exact* number of players is to be added, it might be desirable to *maintain* an original player's power index (or to keep it upper-bounded or lower-bounded by the original value – we will say the index is *nonincreasing* or *nondecreasing*).

Note that power indices in weighted voting games can be used to model decision processes in legislative bodies such as the EU Commission, national parliaments, or the United Nations Security Council. For a real-world example, suppose there is a discussion on whether a new member will join the EU, or an old member will leave the EU. What impact does this change

[1] While merging is an action of a manipulative coalition, annexation describes one manipulative player who takes over other players. This goes back to the bloc paradox [28].

Table 1 Overview of complexity results of control problems in weighted voting games with respect to the Shapley–Shubik and the probabilistic Penrose–Banzhaf index. Key: k is the number of players to be added or deleted, respectively; *PI* stands for *power index* (either *SS* or *PB*); *SS* (respectively, *PB*) indicates that these results are only known to hold for the Shapley–Shubik index (respectively, for the probabilistic Penrose–Banzhaf index); the other results each hold for both indices.

Goal	Control type		
	Adding players		Deleting players
	k fixed	k given	$k = 1$
Increase *PI*	PP-complete (Thm. 9)	PP-hard (Thm. 8)	NP-hard *(SS)* (Thm. 10)
Nondecrease *PI*	PP-complete (Thm. 9)	PP-hard (Thm. 8)	?
Decrease *PI*	PP-complete (Thm. 9)	PP-hard (Thm. 8)	coNP-hard *(PB)* (Thm. 11)
Nonincrease *PI*	PP-complete (Thm. 9)	PP-hard (Thm. 8)	coNP-hard *(PB)* (Thm. 13)
Maintain *PI*	coNP-hard, in PP (Thm. 12)	PP-hard (Thm. 8)	coNP-hard (Thm. 13)

have on the power of the existing, or remaining, EU members? Will they benefit from adding or deleting other members? Or, when it has been decided already that some new members will join the EU, an old member may be interested in maintaining the same power as before the new members have joined the game. We will show that all control types to be defined are possible in weighted voting games, and we will therefore analyze the computational complexity of whether control by structural changes can be exerted successfully in a given game. The complexity depends on the control type, the goal, and on whether the number of players that can be added or deleted is fixed or is given in the problem instance. Table 1 gives an overview.

2 Preliminaries

A *cooperative game with transferable utility* $\mathcal{G} = (N, v)$ consists of a set of players N and a coalitional function $v : 2^N \to \mathbb{R}$ assigning a value to each subset of players, called a *coalition*. $\mathcal{G}$ is called *simple* if v it is *monotonic* (i.e., $v(C) \leq v(D)$ whenever $C \subseteq D \subseteq N$) and if a coalition C is either *winning* ($v(C) = 1$) or *losing* ($v(C) = 0$).

Power indices are a common concept to measure a player's significance in a simple game $\mathcal{G} = (N, v)$. Two popular indices are the Penrose–Banzhaf index [37, 4] and the Shapley–Shubik index [43]. Let n be the number of players in $\mathcal{G}$ and $i \in N$. Define i's *raw Penrose–Banzhaf power index in* $\mathcal{G}$ by $\text{PenroseBanzhaf}^*(\mathcal{G}, i) = \sum_{C \subseteq N \smallsetminus \{i\}} (v(C \cup \{i\}) - v(C))$ and i's *probabilistic Penrose–Banzhaf power index in* $\mathcal{G}$ (proposed by Dubey and Shapley [17]) by

$$\text{PenroseBanzhaf}(\mathcal{G}, i) = \frac{\text{PenroseBanzhaf}^*(\mathcal{G}, i)}{2^{n-1}}. \tag{1}$$

We say that i is *critical for a coalition* C if the marginal contribution $v(C \cup \{i\}) - v(C)$ of player i to coalition C in the definition above is 1, i.e., if C is losing but, after i has joined, $C \cup \{i\}$ is winning. On the other hand, $v(C \cup \{i\}) - v(C) = 0$ means that i is not critical for C. Player i's *raw Shapley–Shubik power index in* $\mathcal{G}$ is $\text{ShapleyShubik}^*(\mathcal{G}, i) =$

$\sum_{C \subseteq N \smallsetminus \{i\}} \|C\|!(n-1-\|C\|)!(v(C \cup \{i\}) - v(C))$, which is then normalized by

$$\text{ShapleyShubik}(\mathcal{G}, i) = \frac{\text{ShapleyShubik}^*(\mathcal{G}, i)}{n!} \tag{2}$$

to obtain i's *Shapley–Shubik power index in* $\mathcal{G}$.

Some simple games $\mathcal{G} = (N, v)$ can be compactly represented as *weighted voting games* $(w_1, \ldots, w_n;\, q)$, where w_i, $1 \leq i \leq n$, is player i's *weight* and q is a *quota*, and a coalition $C \subseteq N$ *wins* if $\sum_{i \in C} w_i \geq q$ and otherwise it *loses*. Note that this representation is not fully expressive, i.e., there are simple games that cannot be represented by weighted voting games. For further background on cooperative game theory, see, e.g., the textbooks by Shoam and Leyton-Brown [44] and Peleg and Sudhölter [36] and, for computational aspects, the book by Chalkiadakis et al. [14] and the book chapters by Elkind et al. [19, 20].

For some background on computational complexity, see, e.g. the textbook by Papadimitriou [35]. We use the standard notions of *hardness* and *completeness* for a complexity class with respect to *many-one polynomial-time reducibility*. NP is the class of decision problems that can be solved in nondeterministic polynomial time, and coNP is the class of problems whose complements are in NP. Partition is the following well-known NP-complete problem:

	Partition
Given:	A set $A = \{1, \ldots, n\}$ and a function $a : A \to \mathbb{N} \smallsetminus \{0\}$, $i \mapsto a_i$, such that $\sum_{i=1}^{n} a_i$ is even.
Question:	Does there exist a partition into two subsets of equal weight, that is, does there exist a subset $A' \subseteq A$ such that $\sum_{i \in A'} a_i = \sum_{i \in A \smallsetminus A'} a_i$?

SubsetSum is also a well-known NP-complete problem:

	SubsetSum
Given:	A set $A = \{1, \ldots, n\}$, a function $a : A \to \mathbb{N} \smallsetminus \{0\}$, $i \mapsto a_i$, and a positive integer q.
Question:	Is there a subset $A' \subseteq A$ such that $\sum_{i \in A'} a_i = q$?

Let $(a_1, \ldots, a_n;\, q)$ and $(a_1, \ldots, a_n)$ denote SubsetSum and Partition instances, respectively. A third well-known NP-complete problem that we will need is

	Exact Cover By 3-Sets (X3C)
Given:	A set $B = \{1, \ldots, 3k\}$, $k > 0$, and a collection $\mathcal{S} = \{S_1, \ldots, S_n\}$ of subsets $S_i \subseteq B$ with $\|S_i\| = 3$ for $1 \leq i \leq n$.
Question:	Is there an exact cover of B in $\mathcal{S}$, that is, is there a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that $\bigcup_{S \in \mathcal{S}'} S = B$ and $S_i \cap S_j = \emptyset$, for each $S_i, S_j \in \mathcal{S}'$, $i \neq j$?

We furthermore consider the function class #P, the class of functions that give the number of solutions of NP problems. A function is #P-*many-one-hard* if there exists a polynomial-time reduction from each function in #P; it is #P-*parsimonious-hard* if there exists such a reduction from each function in #P that preserves the number of solutions. If a #P function is #P-many-one-hard (#P-parsimonious-hard) it is said to be #P-*many-one-complete* (#P-*parsimonious-complete*). For instance, #SubsetSum and #X3C are known to be #P-parsimonious-complete functions. #P is closed under addition, i.e., if $f, g \in$ #P then $f + g \in$ #P. From the literature [38, 16, 26] we obtain the following lemma.

▶ **Lemma 1** ([38, 16, 26]). *Computing the raw Penrose–Banzhaf index is* #P*-parsimonious-complete. Computing the raw Shapley–Shubik index is* #P*-many-one-complete.*

The complexity class PP (*probabilistic polynomial time*) was introduced by Gill [31] via probabilistic Turing machines; equivalently, it can be defined as the class of all decision problems X for which there exist a function $f \in \#\mathrm{P}$ and a polynomial p such that for all instances x: $x \in X \iff f(x) \geq 2^{p(|x|)-1}$. PP is considered to be a rather large complexity class, since it contains both NP and coNP (and even $\mathrm{P}_{\parallel}^{\mathrm{NP}}$ as shown by Beigel et al. [11]) and since it is known to be at least as hard (in terms of polynomial-time Turing reductions) as the polynomial hierarchy (i.e., $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$) by Toda's theorem [45]. PP is closed under complement (which is easy to see) and, far from being trivial, it is also closed under union and intersection [29]. We make use of the following lemma by Faliszewski and Hemaspaandra [26, Lemma 2.3] in the context of comparing a player's power in weighted voting games with respect to the probabilistic Penrose–Banzhaf and the Shapley–Shubik index.

▶ **Lemma 2** ([26]). *Let F be a* #P*-parsimonious-complete function. Then, the problem* Compare-$F = \{(x, y) \mid F(x) > F(y)\}$ *is* PP*-complete.*

Since #X3C and #SubsetSum are #P-parsimonious-complete, Compare-#SubsetSum and Compare-#X3C are PP-complete. Moreover, we will use the following lemma due to Faliszewski and Hemaspaandra [26] that has been slightly adapted by Rey and Rothe [40].

▶ **Lemma 3.** *Every* X3C instance $(B', \mathcal{S}')$ can be transformed into an X3C instance $(B, \mathcal{S})$ where $\|B\| = 3k$ and $\|\mathcal{S}\| = n$, such that $k/n = 2/3$ without changing the number of solutions (i.e., $\#\mathrm{X3C}(\mathrm{B}, \mathcal{S}) = \#\mathrm{X3C}(\mathrm{B}', \mathcal{S}')$). Consequently, we can assume that the size of each solution in a SubsetSum instance is $2n/3$, that is, each subsequence summing up to the given quota contains the same number of elements.

We consider a restricted variant of the Compare-#SubsetSum problem, namely Compare-#SubsetSum-RR as defined in [40]: Given a set $A = \{1, \ldots, n\}$ and a function $a : A \to \mathbb{N} \smallsetminus \{0\}$, $i \mapsto a_i$, is the number of subsets of A with values summing up to $(\alpha/2) - 2$, where $\alpha = \sum_{i=1}^{n} a_i$, greater than the number of subsets of A with values summing up to $(\alpha/2) - 1$, i.e., is it true that

$$\#\textsc{SubsetSum}((a_1, \ldots, a_n;\ (\alpha/2) - 2)) \ > \ \#\textsc{SubsetSum}((a_1, \ldots, a_n;\ (\alpha/2) - 1))? \qquad (3)$$

Let $(a_1, \ldots, a_n)$ denote an instance of Compare-#SubsetSum-RR. From [40, Lemma 4.5] we obtain the following lemma.

▶ **Lemma 4** ([40]). Compare-#SubsetSum-RR *is* PP*-hard.*

Likewise, the analogous problem of whether $<$ holds in (3), denoted by Compare-#SubsetSum-ЯЯ, is PP-hard [40]. The following lemma differentiates between players that are not part of a weighted voting game and those who are but do not have any weight.

▶ **Lemma 5.** *For both the probabilistic Penrose–Banzhaf index and the Shapley–Shubik index, given a weighted voting game, adding a player with weight zero does not change the original players' power indices, and the new player's power index is zero.*

3 Control Types and Goals

We define control by adding and by deleting players in weighted voting games. For each control type, we consider goals, such as increasing or decreasing a distinguished player's power, in relation to the original game. We first define how adding and deleting a player affects the coalitional function for weighted voting games: For control by adding players, from a given weighted voting game $\mathcal{G} = (w_1, \dots, w_n;\, q)$ with $N = \{1, \dots, n\}$ and a set $M = \{n+1, \dots, n+m\}$ of m unregistered players with weights $w_{n+1}, \dots, w_{n+m}$, we obtain a new game $\mathcal{G}_{\cup M} = (w_1, \dots, w_{n+m};\, q)$.

For example, we consider the following decision problem for a power index PI:

Control by Adding Players to Increase PI

Given: A weighted voting game $\mathcal{G}$ with players $N = \{1, \dots, n\}$, a set M of unregistered players with weights $w_{n+1}, \dots, w_{n+m}$, a distinguished player $p \in N$, and a positive integer k.

Question: Can at most k players $M' \subseteq M$ be added to $\mathcal{G}$ such that for the new game $\mathcal{G}_{\cup M'}$ it holds that $\mathrm{PI}(\mathcal{G}_{\cup M'}, p) > \mathrm{PI}(\mathcal{G}, p)$?

Analogously, we can ask whether the game can be controlled so as to gain the opposite effect, and decrease a certain player's index. In these cases, hardness in terms of complexity can be seen as a shield to prevent a game from being controlled to improve a player's significance or to worsen a player's significance. On the other hand, we also consider the following control question: Is it possible to add players to a game without changing the distribution of power among the original players?

We can ask analogous questions with the same aims for removing players from the game. Deleting a subset $M \subseteq N$ of m players from a weighted voting game $\mathcal{G} = (w_1, \dots, w_n;\, q)$ yields a weighted voting game $\mathcal{G}_{\smallsetminus M} = (w_{j_1}, \dots, w_{j_{n-m}};\, q)$ with $\{j_1, \dots, j_{n-m}\} = N \smallsetminus M$.[2]

For instance, we define the following decision problem for a power index PI:

Control by Deleting Players to Increase PI

Given: A weighted voting game $\mathcal{G}$ with players $N = \{1, \dots, n\}$, a distinguished player $p \in N$, and a positive integer $k < \|N\|$.

Question: Can at most k players $M' \subseteq N \smallsetminus \{p\}$ be deleted from $\mathcal{G}$ such that in the new game $\mathcal{G}_{\smallsetminus M'}$ it holds that $\mathrm{PI}(\mathcal{G}_{\smallsetminus M'}, p) > \mathrm{PI}(\mathcal{G}, p)$?

Again, we can analogously define the variations of this problem where the goal is not to increase some player's power index but to decrease or to maintain it.

▶ **Example 6.** Let $\mathcal{G} = (N, v)$ be a weighted voting game with six players in $N = \{1, 2, 3, 4, 5, 6\}$ represented by $(1, 2, 2, 3, 4, 5;\, 10)$. Let $k = 1$, that is, one player can be removed from the game. Table 2 lists the players' probabilistic Penrose–Banzhaf and Shapley–Shubik power indices for the resulting games. Note that fractions are sometimes expanded or reduced to a comparable denominator.

Consider the Penrose–Banzhaf index. Player 1, 4, 5 and 6 with indices of $1/8$, $5/16$, $3/8$, and $9/16$, respectively, cannot improve from any other player being deleted. However, e.g., player 1's index can be decreased to $1/16$ when removing player 5 and is maintained in the other cases. Players 2 and 3 can benefit from the other one being removed, as the index increases from $3/16$ to $1/4$.

[2] One might also think of different ways to reasonably model the new game, and we will eloborate on that in Section 6. Here, we focus on the notion just presented.

Table 2 Power distribution in the games of Example 6.

Player i		1	2	3	4	5	6
$\mathrm{PenroseBanzhaf}(\mathcal{G}, i)$	$\cdot 32$	4	6	6	10	12	18
$\mathrm{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{1\}}, i)$	$\cdot 32$		6	6	10	10	18
$\mathrm{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{2\}}, i)$	$\cdot 32$	4		8	8	12	16
$\mathrm{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{3\}}, i)$	$\cdot 32$	4	8		8	12	16
$\mathrm{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{4\}}, i)$	$\cdot 32$	4	4	4		12	16
$\mathrm{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{5\}}, i)$	$\cdot 32$	2	6	6	10		14
$\mathrm{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{6\}}, i)$	$\cdot 32$	4	4	4	8	8	
$\mathrm{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{1,2\}}, i)$	$\cdot 32$			8	8	8	16
$\mathrm{ShapleyShubik}(\mathcal{G}, i)$	$\cdot 60$	4	6	6	11	13	20
$\mathrm{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{1\}}, i)$	$\cdot 60$		7	7	12	12	22
$\mathrm{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{2\}}, i)$	$\cdot 60$	5		10	10	15	20
$\mathrm{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{3\}}, i)$	$\cdot 60$	5	10		10	15	20
$\mathrm{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{4\}}, i)$	$\cdot 60$	5	5	5		15	30
$\mathrm{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{5\}}, i)$	$\cdot 60$	3	8	8	13		28
$\mathrm{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{6\}}, i)$	$\cdot 60$	6	6	6	21	21	
$\mathrm{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{1,2\}}, i)$	$\cdot 60$			10	10	10	30

For the Shapley–Shubik index, due to normalization over the permutations of participating players, an increase of power is expected when deleting a player. As an example, player 5 has an index of $13/60$ in $\mathcal{G}$ which increases to $1/4$ if either one of the players 2, 3, or 4 is deleted, and even to $7/20$ if 6 is deleted. However, players can also have a disadvantage, if a player leaves the game. For instance, player 1 loses power if 5 is deleted, 2 and 3 lose power if 4 is deleted, 4 loses power if 2 or 3 are deleted, and 5 loses power if 1 is deleted. This suggests a symmetric dependence of the players. In the same way, the power of players 2 and 3 remains the same if 6 is removed, and the other way around.

From the opposite view, consider the weighted voting game represented by $(2, 3, 4, 5; 10)$, two unregistered players with weights 1 and 2, and $k = 2$ (see the bottom rows for the two indices). Note that adding them both ends up in $\mathcal{G}$. Here, the four players have probabilistic Penrose–Banzhaf indices of $1/4$, $1/4$, $1/4$, and $1/2$. The first player (with weight 2) can only be worse off when adding any of the two players. The player with weight 3 as well as the player with weight 5 can benefit from adding both players or only the one with weight 2. The former keeps the same index, while the latter loses power if the player with weight 1 is added. Finally, the player with weight 4 improves in every situation when adding one or two players. The first and the fourth player (with weight 2 and 5, respectively) cannot benefit from adding players with respect to the Shapley–Shubik index. The other two can take advantage in the same cases as for the probabilistic Penrose–Banzhaf index.

In particular, the example shows that these types of control are each possible. We therefore turn to the question of how hard it is to find out whether they can be exerted successfully in a given game. Next to goals in relation to the old game, we can also compare an index either in relation to the other players' power, or in relation to a constant number. See Section 6 for initial results for this idea. If a player i is deleted from a weighted voting game, any other player j gains the same amount of power that i would gain if j were deleted [33].

The changes of power indices by deletion of players are bounded as follows.

▶ **Theorem 7.** *After deleting the players of a subset $M \subseteq N \setminus \{i\}$ of size $m \geq 1$ from a weighted voting game $\mathcal{G} = (N, v)$, the difference between player i's old and new*
1. *Penrose–Banzhaf index is at most $1 - 2^{-m}$ and is at least $-1 + 2^{-m}$;*
2. *Shapley–Shubik index is at most $1 - {}^{(n-m+1)!}/{}_{2n!}$ and is at least $-1 + {}^{(n-m-1)!}/{}_{2(n-2)!}$.*

In particular, if $m = 1$ player is deleted, both the Penrose–Banzhaf index and the Shapley–Shubik index of any other player can increase by at most $1/2$ and will decrease by at most $-1/2$. These bounds are tight.

4 Increasing or Decreasing an Index

Similarly to control by adding or deleting voters or candidates in elections, adding and deleting players are not merely inverse operations. This is due to the fact that when adding players all original players are guaranteed to be part of the game before and after the structural change, whereas when deleting players each player except the distinguished one can be removed from the game. Hardness in terms of complexity can be seen as a shield to prevent a game from being controlled to improve or worsen a player's significance.

4.1 Control by Adding Players

From a computational complexity point of view, we distinguish the cases where an upper bound of new players is given as defined above and where the number of new players is fixed.

▶ **Theorem 8.** *Control by adding a given number of players in order to increase (decrease) a distinguished player's probabilistic Penrose–Banzhaf or Shapley–Shubik index in a weighted voting game is* PP*-hard.*

Proof. We show PP-hardness via the techniques used by Rey and Rothe [40], Faliszewski and Hemaspaandra [26], and Zuckerman et al. [48]. By Lemma 4, Compare-#SubsetSum-RR is PP-hard. Reducing from this problem, we map an instance $(a_1, \ldots, a_n)$ with $\alpha = \sum_{i=1}^{n} a_i$ to a weighted voting game $\mathcal{G}$ represented by $(1, a_1, \ldots, a_n;\ \alpha/2)$, an unregistered player with weight $w_{n+2} = 1$, $k = 1$, and distinguished player $p = 1$. There is only one possible new game obtained by adding the unregistered player to the game $\mathcal{G}_{\cup\{n+1\}}$. We show that

$$\begin{aligned}&\text{PenroseBanzhaf}(\mathcal{G}_{\cup\{n+2\}}, 1) - \text{PenroseBanzhaf}(\mathcal{G}, 1) > 0\\ &\quad\iff\quad \#\textsc{SubsetSum}((a_1, \ldots, a_n;\ \alpha/2 - 2)) > \#\textsc{SubsetSum}((a_1, \ldots, a_n;\ \alpha/2 - 1)).\end{aligned}$$

It holds that

$$\begin{aligned}&\text{PenroseBanzhaf}(\mathcal{G}_{\cup\{n+1\}}, 1) - \text{PenroseBanzhaf}(\mathcal{G}, 1) && (4)\\ &\quad = 1/2^n (\|\{C \subseteq \{2, \ldots, n+1\} \mid 2 + \textstyle\sum_{i \in C} a_{i-1} \geq \alpha/2,\ 1 + \sum_{i \in C} a_{i-1} < \alpha/2\}\| && (5)\\ &\qquad\quad - \|\{C \subseteq \{2, \ldots, n+1\} \mid 1 + \textstyle\sum_{i \in C} a_{i-1} \geq \alpha/2,\ \sum_{i \in C} a_{i-1} < \alpha/2\}\|). && (6)\end{aligned}$$

If for some $C \subseteq \{2, \ldots, n+1\}$ the conditions of the set in (6) are satisfied (i.e., $\sum_{i \in C} a_{i-1} < \alpha/2$ but $1 + \sum_{i \in C} a_{i-1} \geq \alpha/2$), then $\alpha/2 - 1 = \sum_{i \in C} a_{i-1}$, since the weights and the quota are integers. If for some $C \subseteq \{2, \ldots, n+1\}$ the conditions of the set in (5) are satisfied, then $\alpha/2 - 2 = \sum_{i \in C} a_{i-1}$. Therefore, the term in (4) is positive if and only if the number of solutions that sum up to $\alpha/2 - 2$ is greater than $\alpha/2 - 1$. Thus it is PP-hard to verify whether the Penrose–Banzhaf index of a player can be increased by adding players.

Analogously, for the goal of decreasing an index we can reduce from the PP-hard problem Compare-#SubsetSum-ЯЯ, which is defined in [40] by switching $\alpha/2 - 2$ and $\alpha/2 - 1$ in the definition of Compare-#SubsetSum-RR.

Likewise, with Lemma 3 these results can be adapted to the Shapley–Shubik index. ◀

▶ Remark. An upper bound of $\mathrm{NP}^{\mathrm{PP}}$ can be established whenever the number of players to be added is given. We can guess the subset of new players to be added nondeterministically. Verifying whether the different goals are satisfied is encoded in the PP-oracle. We conjecture that this problem is complete for this class.

▶ **Theorem 9.** *Control by adding a fixed number of players in order to increase (decrease) a distinguished player's probabilistic Penrose–Banzhaf or Shapley–Shubik index in a weighted voting game is* PP*-complete.*

Proof. Since the number of players to be added is fixed, there are polynomially many combinations to be added. Therefore, we have polynomially many comparisons of power indices. No matter which goal we consider, the comparison can be done in PP by Lemmas 1 and 2 and by the facts that #P is closed under addition and PP is closed under complement. The problem belongs to PP, since PP is closed under union.

Hardness is implied by the case of $k = 1$ player to be added in the proof of Theorem 8. By Lemma 5, this also holds for any other fixed number of players to be added. ◀

4.2 Control by Deleting Players

Recall that although deleting a previously added player results in the same game, the possibility to fulfill a certain goal by adding a player is not the complement of the possiblity to fulfill the complement goal by deleting a player. Initially, we obtain the following.

▶ **Theorem 10.** *Control by deleting players to increase a distinguished player's Shapley–Shubik index in a weighted voting game is* NP*-hard (even if only one player is deleted).*

Proof. We show NP-hardness by means of a reduction from SubsetSum. By Lemma 3 we can assume that the satisfying solutions all have the same size ℓ. Let $(a_1, \ldots, a_n; q)$ be a SubsetSum instance, consider the weighted voting game $\mathcal{G}$ represented by $(1, a_1, \ldots, a_n, q+1; q+1)$, and consider player 1 as our distiguished player. Let $k = 1$ and let $\xi = \#\text{SubsetSum}((a_1, \ldots, a_n; q))$ denote the number of solutions for the SubsetSum instance. Then, for the raw Shapley–Shubik index it holds that $\xi \geq 1$ if and only if deleting some player but 1 can lead to an increase of 1's index.

If: If $\xi = 0$, $\text{ShapleyShubik}^*(\mathcal{G}, 1)$ is and remains 0 no matter which player is deleted.

Only if: Assume that $\xi \geq 1$. Then $\text{ShapleyShubik}^*(\mathcal{G}, 1) = \xi/2 \cdot \ell!(n+1-\ell)! + \xi/2 \cdot (n-\ell)!(\ell+1)!$. If player $n+2$ is deleted, player 1's new raw index is $\text{ShapleyShubik}^*(\mathcal{G}_{\smallsetminus\{n+2\}}, 1) = \xi \cdot \ell!(n-\ell)!$. This leads to

$$\begin{aligned}
&\text{ShapleyShubik}(\mathcal{G}_{\smallsetminus\{n+2\}}, 1) - \text{ShapleyShubik}(\mathcal{G}, 1)\\
&= \frac{1}{(n+1)!} \cdot \xi \cdot \ell!(n-\ell)! \cdot \frac{2}{2} - \frac{1}{(n+2)!} \cdot \frac{\xi}{2} \cdot \ell!(n-\ell)!(n+1-\ell+\ell+1)\\
&= \frac{1}{(n+1)!} \cdot \frac{\xi}{2}(2-1)\ell!(n-\ell)!,
\end{aligned}$$

which is greater than 0 because $\ell!$ and $(m-\ell)!$ are positve. ◀

▶ **Theorem 11.** *Control by deleting players to decrease a distinguished player's Penrose–Banzhaf index in a weighted voting game is* coNP*-hard (even if only one player can be deleted).*

Proof. We show coNP-hardness by means of a reduction from the complement of PARTITION, denoted by $\overline{\text{PARTITION}}$. Letting $(a_1, \dots, a_n)$ be a PARTITION instance with $\alpha = \sum_{i=1}^{n} a_i$ and $\xi = \#\text{PARTITION}((a_1, \dots, a_n))$, we construct the control instance consisting of $\mathcal{G} = (1, a_1, \dots, a_n, \alpha/2; \alpha/2 + 1)$, $p = 1$, and $k = 1$. We show that $\xi = 0$ if and only if there exists a player whose removal from the game causes player 1's Penrose–Banzhaf power to decrease.

Only if: Assume that $\xi = 0$. Then $\text{PenroseBanzhaf}^*(\mathcal{G}, 1) = 1$. However, if player $n+2$ with weight $\alpha/2$ is removed, there is no coalition left player 1 is critical for. Therefore, control in order to decrease player 1's Penrose–Banzhaf index is possible.

If: Assume that $\xi \geq 0$. Then $\text{PenroseBanzhaf}^*(\mathcal{G}, 1) = \xi + 1$. If player $n+2$ is deleted, $\text{PenroseBanzhaf}^*(\mathcal{G}_{\smallsetminus\{n+2\}}, 1) = \xi$ and

$$\text{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{n+2\}}, 1) - \text{PenroseBanzhaf}(\mathcal{G}, 1) = \frac{\xi}{2^n} - \frac{\xi+1}{2^{n+1}} = \frac{\xi-1}{2^{n+1}} \geq 0.$$

Note that this difference is even greater than 0, since ξ is even. If a player j, $2 \leq j \leq n+1$, is deleted, $\text{PenroseBanzhaf}^*(\mathcal{G}_{\smallsetminus\{j\}}, 1) = 1 + \xi/2$ and

$$\text{PenroseBanzhaf}(\mathcal{G}_{\smallsetminus\{n+2\}}, 1) - \text{PenroseBanzhaf}(\mathcal{G}, 1) = \frac{1+\frac{\xi}{2}}{2^n} - \frac{\xi+1}{2^{n+1}} = \frac{1}{2^{n+1}} > 0.$$

Consequently, a decrease of player 1's Penrose–Banzhaf index is not possible by deleting any other player than 1. ◀

5 Maintaining an Index

In addition to constructive or destructive goals, we now consider situations in which an exact number of players is to be added and the goal is to either maintain a distinguished player's power index in this new game, or at least to ensure that this player's power does not increase or decrease, compared with this player's power in the old game.

For instance, control by adding players with the goal to maintain a given player's power index PI is defined as follows. The other goals of nonincreasing or nondecreasing a given player's power by adding or deleting players can be defined analogously.

CONTROL BY ADDING PLAYERS TO MAINTAIN PI	
Given:	A weighted voting game $\mathcal{G}$ with players $N = \{1, \dots, n\}$, a set M of unregistered players with weights $w_{n+1}, \dots, w_{n+m}$, a distinguished player $p \in N$, and a positive integer k.
Question:	Can exactly k players $M' \subseteq M$ be added to $\mathcal{G}$ such that for the new game $\mathcal{G}_{\cup M'}$ it holds that $\text{PI}(\mathcal{G}_{\cup M'}, p) = \text{PI}(\mathcal{G}, p)$?

5.1 Control by Adding Players

Analogously to Theorem 8, since PP is closed under complement and by an alternative reduction from the complement of COMPARE-#SUBSETSUM-ЯЯ, control by adding a given number of players in order to maintain a distinguished player's probabilistic Penrose–Banzhaf or Shapley–Shubik index in a weighted voting game is PP-hard. Similarly, whenever the number of players to be added is given in unary, these problems are in NP^{PP}.

▶ **Theorem 12.** *Control by adding a fixed number of players to maintain a distinguished player's probabilistic Penrose–Banzhaf or Shapley–Shubik index in a weighted voting game is* coNP*-hard and in* PP*.*

Proof. The upper bound holds by the same argument as in Theorem 9. We can show coNP-hardness by reducing from $\overline{\text{PARTITION}}$. By Lemma 5, the arguments also hold for any other fixed number of players to be added. ◀

5.2 Control by Deleting Players

Note again that deleting players in order to increase a power index is not the inverse of adding players in order to nonincrease the same index.

▶ **Theorem 13.** *Control by deleting a player in order to maintain a distinguished player's probabilistic Penrose–Banzhaf index in a weighted voting game is* coNP*-hard (even if only one player can be deleted).*

Proof. Again, we show coNP-hardness by means of a reduction from $\overline{\text{PARTITION}}$. Letting $(a_1, \dots, a_n)$ be a PARTITION instance with $\alpha = \sum_{i=1}^{n} a_i$, we construct the game $\mathcal{G}$ represented by $(1, a_1, \dots, a_n, \alpha/2, \alpha/2; \alpha/2 + 1)$ and consider player 1 as our distigiuished player. Let $k = 1$ and let $\xi = \#\text{PARTITION}((a_1, \dots, a_n))$ denote the number of solutions to the PARTITION instance. Then, for the raw Penrose–Banzhaf, it holds that $\xi \geq 1$ if and only if deleting any player but 1 does not maintain the index of player 1.

If: Assume that $\xi = 0$. Then PenroseBanzhaf$^*(\mathcal{G}, 1) = 2$. If player $n + 2$ with weight $\alpha/2$ is deleted, the raw index of player 1 is PenroseBanzhaf$^*(\mathcal{G}_{\smallsetminus\{n+2\}}, 1) = 1$, which results in the same probabilistic Penrose–Banzhaf index. The factor of 2 is due to the fact that the raw index is twice as significant in the new game with one player less than in the old game.

Only if: Assume that $\xi \geq 1$. Then PenroseBanzhaf$^*(\mathcal{G}, 1) = \xi + 2$. If player $n + 2$ or $n + 3$ is deleted, player 1's new raw index is $\xi + 1$. This leads to a higher index since $\xi + 2 < 2(\xi + 1)$. Deleting player j, $2 \leq j \leq n + 1$, leads to a raw index of $\xi/2 + 2$, which means that in comparison to the old game, player 1's index is increased: $\xi + 2 < 2(\xi/2 + 2) = \xi + 4$.

Hence, the problem of whether it is possible to maintain a player's probabilistic Penrose–Banzhaf index is coNP-hard. ◀

In particular, if $\xi \geq 1$ in the above proof, then deleting any player cannot lead to a nonincrease. Therefore, it also holds that $\xi \geq 1$ if and only if deleting any player but 1 does not nonincrease the probabilistic Penrose–Banzhaf index of player 1. Therefore, we get coNP-hardness for the problem where the goal is to nonincrease the probabilistic Penrose–Banzhaf by essentially the same proof. Observe that from these constructions we cannot draw further conclusion about the complexity of structural control by deleting players for neither the Shapley–Shubik nor the probabilistic Penrose–Banzhaf index.

6 Conclusions and Future Work

For weighted voting games, we have studied two types of control, combined with the following variants of goals: Strictly increasing or strictly decreasing a player's power index by adding or deleting at most a given number of players as well as maintaining, nondecreasing, and nonincreasing a player's power index by adding or deleting an exact number of players. As a measure of a player's power we have analyzed the well-known Shapley–Shubik power index and the probabilistic Penrose–Banzhaf power index. If the number of players to be added is given, the problems of adding players in order to obtain a change in a player's index (or at least allow a change in one direction) is PP-hard. And if the number of players to be added is fixed, a corresponding PP upper bound is valid, so we have PP-completeness. In the case of deleting players, we have established NP- and coNP-hardness lower bounds, even for the case of deleting exactly one player. The complexity results are summed up in Table 1.

The complexity of some control problems is left open; for instance, interesting gaps remain, e.g., between NP-hardness and PP membership as well as PP-hardness and $\mathrm{NP}^{\mathrm{PP}}$ membership, and we do not know the complexity of control by deleting players in order to nondecrease a player's index. Also, considering other measures of voter power may provide further insights into the problem of structurally controlling a game. Next to classic worst-case complexity results, it would be interesting to study approximations and average cases to understand the occurrence of computationally hard instances. Especially, it is interesting to find out how frequent the occurrence of a player gaining power by adding other players (or, likewise decreasing and deleting) is. This case would not to be expected intuitively but, as we have seen in Example 6, it is possible.

So far we have only obtained results for goals in *relation to the original game*. Alternatively, one might think of a situation where the goal is to increase a player's significance in *comparison to the other players*, which can also be achieved if players are added or deleted; the distinguished player's power index remains the same, but all remaining players' indices are distributed so that they are below this value. Besides this, we can also model a scenario where a player is required to exceed a certain *constant power index*, and we ask whether it is possible to control a game by adding or deleting players in order to reach this index. So far, we can tell that if the number of players to be added or deleted is $k = 0$, our value is $1/2$, and the considered power index is the Penrose–Banzhaf index, the problem is PP-complete. This might change if $k > 0$ is required. We might also study the variant of obtaining an exact value. Further, there seems to be a close connection to the notion of *synergies* in cooperative games (see, e.g., [39]), and it will be interesting to have a closer look at related results here.

In addition to weighted voting games, other classes of cooperative games with transferable utility might of course be affected by control scenarios as well. In each case, adding and deleting players has to be well-defined. As an example, consider general (weighted) majority games. Let $\mathcal{G} = (w_1, w_2, \ldots, w_n;\ \alpha(n))$ be a majority game, that is, $v(C) = 1$ if $\sum_{i \in C} w_i \geq \lfloor \alpha(n) \rfloor + 1$, and $v(C) = 0$ otherwise, for each $C \subseteq N$. Now, if a player is deleted, the number of players n is decremented, which changes the threshold $\alpha(n)$. The new coalitional function is computed as above. Adding a player requires a set of unregistered players given by their weights, and n is increased. For (weighted) threshold games, the new coalitional function is determined similarly, with the difference that the threshold does not change. One could alternatively think of weights as a percentage, and change the weights of the remaining players proportionally. Thus the new game $\mathcal{G}_{\cup M}$ is defined differently, by normalizing the sum of weights to the original value. Similarly to majority games, players now do not make an absolute but a relative contribution to the game.

Adding and deleting players can be viewed as a change over time and analyzing to what extent this influences power indices is an interesting task for future work (previously, only changing the quota over time has been studied [18]). Other games that will be interesting to study in this context include games in which the Shapley–Shubik index is easy to compute, such as weighted graph games [16]. In such games, two indices in two games can be compared in polynomial time and, therefore, if the coalition that is added to or removed from a game is known, the possibility of control is easy to detect, rendering the problems trivial. If, on the other hand, there are several possible coalitions to be added, this problem might become interesting again. Eventually, if players correspond to an edge in a game, deleting an edge may be interesting in the context of Braess's paradox for noncooperative congestion games (see, e.g., [34, pp. 464–465]) where, informally, an extra fast lane might lead to congestion, whereas without this lane traffic may split up to equally slower paths. Can we find a connection to control by deleting a player in a cooperative game with transferable utility?

Acknowledgments. We thank the anonymous MFCS 2916, AAMAS 2016, CoopMAS 2016, and LOFT 2016 reviewers for many helpful comments on earlier drafts of this paper.

References

1 H. Aziz, Y. Bachrach, E. Elkind, and M. Paterson. False-name manipulations in weighted voting games. *Journal of Artificial Intelligence Research*, 40:57–93, 2011.

2 Y. Bachrach and E. Elkind. Divide and conquer: False-name manipulations in weighted voting games. In *Proc. AAMAS'08*, pages 975–982. IFAAMAS, 2008.

3 Y. Bachrach and E. Porat. Path disruption games. In *Proc. AAMAS'10*, pages 1123–1130. IFAAMAS, 2010.

4 J. Banzhaf III. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19:317–343, 1965.

5 J. Bartholdi III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

6 J. Bartholdi III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical Computer Modelling*, 16(8/9):27–40, 1992.

7 D. Baumeister, G. Erdélyi, O. Erdélyi, and J. Rothe. Control in judgment aggregation. In *Proc. STAIRS'12*, pages 23–34. IOS Press, 2012.

8 D. Baumeister, G. Erdélyi, O. Erdélyi, and J. Rothe. Complexity of manipulation and bribery in judgment aggregation for uniform premise-based quota rules. *Mathematical Social Sciences*, 76:19–30, 2015.

9 D. Baumeister, G. Erdélyi, and J. Rothe. Judgment aggregation. In J. Rothe, editor, *Economics and Computation. An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, chapter 6, pages 361–391. Springer-Verlag, 2015.

10 D. Baumeister and J. Rothe. Preference aggregation by voting. In J. Rothe, editor, *Economics and Computation. An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, chapter 4, pages 197–325. Springer-Verlag, 2015.

11 R. Beigel, L. Hemachandra, and G. Wechsung. On the power of probabilistic polynomial time: $\mathrm{P}^{\mathrm{NP[log]}} \subseteq \mathrm{PP}$. In *Proc. Structures'89*, pages 225–227. IEEE Computer Society Press, 1989.

12 F. Brandt, V. Conitzer, and U. Endriss. Computational social choice. In G. Weiß, editor, *Multiagent Systems*, pages 213–283. MIT Press, second edition, 2013.

13 F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. Procaccia, editors. *Handbook of Computational Social Choice.* Cambridge University Press, 2016.

14 G. Chalkiadakis, E. Elkind, and M. Wooldridge. *Computational Aspects of Cooperative Game Theory.* Morgan & Claypool, 2011.

15 V. Conitzer and T. Walsh. Barriers to manipulation in voting. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 6, pages 127–145. Cambridge University Press, 2016.

16 X. Deng and C. Papadimitriou. On the complexity of comparative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

17 P. Dubey and L. Shapley. Mathematical properties of the Banzhaf power index. *Mathematics of Operations Research*, 4(2):99–131, 1979.

18 E. Elkind, D. Pasechnik, and Y. Zick. Dynamic weighted voting games. In *Proc. AAMAS'13*, pages 515–522. IFAAMAS, 2013.

19 E. Elkind, T. Rahwan, and N. Jennings. Computational coalition formation. In G. Weiß, editor, *Multiagent Systems*, pages 329–380. MIT Press, second edition, 2013.

20 E. Elkind and J. Rothe. Cooperative game theory. In J. Rothe, editor, *Economics and Computation. An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division*, chapter 3, pages 135–193. Springer-Verlag, 2015.

21 U. Endriss. Sincerity and manipulation under approval voting. *Theory and Decision*, 74(3):335–355, 2013.

22 U. Endriss. Judgment aggregation. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 17, pages 399–426. Cambridge University Press, 2016.

23 U. Endriss, U. Grandi, and D. Porello. Complexity of judgment aggregation. *Journal of Artificial Intelligence Research*, 45:481–514, 2012.

24 P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *Journal of Artificial Intelligence Research*, 35:485–532, 2009.

25 P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *Journal of Artificial Intelligence Research*, 35:275–341, 2009.

26 P. Faliszewski and L. Hemaspaandra. The complexity of power-index comparison. *Theoretical Computer Science*, 410(1):101–107, 2009.

27 P. Faliszewski and J. Rothe. Control and bribery in voting. In F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 7, pages 146–168. Cambridge University Press, 2016.

28 D. Felsenthal and M. Machover. Postulates and paradoxes of relative voting power – A critical re-appraisal. *Theory and Decision*, 38(2):195–229, 1995.

29 L. Fortnow and N. Reingold. PP is closed under truth-table reductions. *Information and Computation*, 124(1):1–6, 1996.

30 J. Freixas and M. Pons. Circumstantial power: Optimal persuadable voters. *European Journal of Operational Research*, 186:1114–1126, 2008.

31 J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

32 E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5-6):255–285, 2007.

33 R. Myerson. Conference structures and fair allocation rules. *International Journal of Game Theory*, 9(3):169–182, 1980.

34 N. Nisan, T. Roughgarden, É. Tardos, and V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, 2007.

35 C. Papadimitriou. *Computational Complexity.* Addison-Wesley, second edition, 1995.

36 B. Peleg and P. Sudhölter. *Introduction to the Theory of Cooperative Games.* Springer-Verlag, second edition, 2007.

37 L. Penrose. The elementary statistics of majority voting. *Journal of the Royal Statistical Society*, 109(1):53–57, 1946.

38 K. Prasad and J. Kelly. NP-completeness of some problems concerning voting games. *International Journal of Game Theory*, 19(1):1–9, 1990.

39 T. Rahwan, T. Michalak, and M. Wooldridge. A measure of synergy in coalitions. Technical Report arXiv:1404.2954.v1 [cs.GT], CoRR, Apr. 2014.

40 A. Rey and J. Rothe. False-name manipulation in weighted voting games is hard for probabilistic polynomial time. *Journal of Artificial Intelligence Research*, 50:573–601, 2014.

41 A. Rey, J. Rothe, and A. Marple. Path-disruption games: Bribery and a probabilistic model. *Theory of Computing Systems*, 2016. `doi:10.1007/s00224-016-9669-1`.

42 J. Rothe, editor. *Economics and Computation. An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division.* Springer-Verlag, 2015.

43 L. Shapley and M. Shubik. A method of evaluating the distribution of power in a committee system. *American Political Science Review*, 48(3):787–792, 1954.

44 Y. Shoham and K. Leyton-Brown. *Multiagent Systems. Algorithmic, Game-Theoretic, and Logical Foundations.* Cambridge University Press, 2009.

45 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

46 Y. Zick. On random quotas and proportional representation in weighted voting games. In *Proc. IJCAI'13*, pages 432–438. AAAI Press, 2013.

47 Y. Zick, A. Skopalik, and E. Elkind. The Shapley value as a function of the quota in weighted voting games. In *Proc. IJCAI'11*, pages 490–496. AAAI Press, 2011.

48 M. Zuckerman, P. Faliszewski, Y. Bachrach, and E. Elkind. Manipulating the quota in weighted voting games. *Artificial Intelligence*, 180–181:1–19, 2012.

Every Binary Pattern of Length Greater Than 14 Is Abelian-2-Avoidable

Matthieu Rosenfeld

Lip, ENS de lyon, 46 Allée d'Italie Lyon 69364 France; and CNRS, UCBL, Université de Lyon
matthieu.rosenfeld@ens-lyon.fr

Abstract

We show that every binary pattern of length greater than 14 is abelian-2-avoidable. The best known upper bound on the length of abelian-2-unavoidable binary pattern was 118, and the best known lower bound is 7.

We designed an algorithm to decide, under some reasonable assumptions, if a morphic word avoids a pattern in the abelian sense. This algorithm is then used to show that some binary patterns are abelian-2-avoidable. We finally use this list of abelian-2-avoidable pattern to show our result. We also discuss the avoidability of binary patterns on 3 and 4 letters.

1998 ACM Subject Classification G.2.1 Combinatorial Algorithms

Keywords and phrases Combinatorics on words; Pattern avoidance; Abelian repetitions

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.81

1 Introduction

The avoidability of patterns in words has been widely studied since the work of Thue on avoidability of repetitions [22, 23]. Thue wanted to know whether, for any word u and long enough word w, there is always a non-erasing morphism h such that $h(u)$ is a factor of w. He answered negatively to the question by constructing an infinite word over three letters that does not contain any image of AA, and an infinite word over two letters that does not contain any image of AAA.

The formal notion of pattern was introduced in [2]. For two words P and w, we say that w avoids the pattern P if there is no non-erasing morphism h such that $h(P)$ is a factor of w, or equivalently if there is no factor $w_1w_2\dots w_{|P|}$ in w such that $\forall i,j,\ P_i = P_j \implies w_i = w_j$. The avoidability of patterns was studied by Zimin [24] and many other authors worked on the classification of avoidable patterns [4, 14, 15, 20, 21]. In particular Roth proved in [20] that binary patterns of length greater than 6 are avoidable over the binary alphabet and in [1] authors showed the existence of a pattern avoidable over 4 letters, but not avoidable over 3 letters. More recently it has been showed that patterns with m different letters of length at least $3(2^{m-1})$ are 2-avoidable [3, 16].

Erdős proposed a commutative version of the results of Thue [8, 9]. An *abelian square* is any non-empty word uv where u and v are permutations of each other. Erdős asked whether there is an infinite abelian-square-free word over an alphabet of size 4 [8, 9]. After some intermediary results (alphabet of size 25 by Evdokimov [10] and size 5 by Pleasant [17]), Keränen answered positively to Erdős's question by giving a 85-uniform morphism (found with the assistance of a computer) whose fixed point is abelian-square-free [11]. Moreover, Dekking showed that it is possible to avoid abelian cubes on a ternary alphabet and abelian-4-powers over a binary alphabet [7].

Following the question of Erdős we say that two words u and v are *abelian equivalent*, denoted $u \approx_a v$, if they are permutations of each other, for example: *listen* $\approx_a$ *silent*. Let $P = P_1P_2 \ldots P_n$ be a pattern, where the P_i are letters. Then we say that a word $w \in \Sigma^*$ *realizes* P *in the abelian sense* if there are $w_1, \ldots, w_n \in \Sigma^+$ such that $w = w_1w_2 \ldots w_n$ and $\forall i, j,\ P_i = P_j \implies w_i \approx_a w_j$. If a word w has no factor that realizes a pattern P in the abelian sense, then w *avoids* P in the abelian sense, w is *abelian-P-free*. In Section 2 we show that one can decide if the fixed point of a morphism avoids a given pattern. This generalizes a result from [5] that tells that under some conditions one can decide if the fixed point of a morphism avoids abelian-k-powers.

We say that a pattern is *abelian-k-avoidable* if there is a word from an alphabet of size k that avoids this pattern. For any pattern $P \in \Delta^*$, the *abelian-avoidability index* of P (denoted by $\lambda_a(P)$) is the smallest integer k such that P is abelian-k-avoidable or ∞ if there is no such k. It is an abelian analog of the usual avoidability index of a pattern P. For example $\lambda_a(ABA) = \lambda_a(ABACABA) = \infty$, $\lambda_a(AA) = 4$ [11], $\lambda_a(AAA) = 3$ and $\lambda_a(AAAA) = 2$ [7]. In [6] authors showed that binary pattern of length greater than 118 are abelian-2-avoidable and asked for a more precise characterization. We can use the algorithm of Section 2 to show that every binary pattern of length greater than 14 are abelian-2-avoidable.

In Section 2 we explain how to decide, under some conditions, whether a morphic word avoids a given pattern in the abelian sense. In Section 3 we show that binary patterns of length greater than 14 are abelian-2-avoidable. We also discuss the avoidability of binary patterns over any finite alphabet and we raise some open questions.

2 Proving the decidability

In this part we explain how to decide, under some conditions, if the fixed point of a morphism avoids some given pattern in the abelian sense.

We use terminology and notations of Lothaire [13]. For any morphism $h : \Sigma^* \mapsto \Sigma^*$, if there is $a \in \Sigma$ and $w \in \Sigma^*$ such that $h(a) = aw$, then the sequence $(h^n(a))_{n\geq 0}$ converges for the usual topology on $\Sigma^* \cup \Sigma^\omega$ and we denote by $h^\omega(a) = \lim_{n\to 0} h^n(a)$. Note that $h^\omega(a)$ is a fixed point of h. To any pattern P we associate the function $\varphi_P : [1, |P|] \mapsto [1, |P|]$ such that $\varphi_P(i) = \min\{j : P_j = P_i\}$ is the position of the first occurrence of the letter P_i in P.

For any word w, we denote by $[w]_i$ the letter at position i in w (or w_i if it is clear in the context). For any word w, we denote by $|w|$ the length of w and for any letter $a \in \Sigma$, $|w|_a$ is the number of occurrences of a in w. The *Parikh vector* of a word $w \in \Sigma^*$, denoted by $\Psi(w)$, is the vector indexed by Σ such that for every $a \in \Sigma$, $\Psi(w)[a] = |w|_a$. Note that by definition for any two words u and v, $u \approx_a v$ iff $\Psi(u) = \Psi(v)$.

We associate to any morphism $h : \Sigma^* \mapsto \Sigma^*$ a matrix M_h on $\Sigma \times \Sigma$ such that $(M_h)_{a,b} = |h(b)|_a$. Note that from the definition we can deduce for any morphism h and any word u the formula:

$$\Psi(h(u)) = M_h\Psi(u).$$

The induced norm of a matrix $M \in \mathbb{R}_{m\times m}$ is given by $\|M\|_2 = sup_{x\in\mathbb{R}^m} \frac{\|Mx\|_2}{\|x\|_2}$ where $\|v\|_2$ is the Euclidean norm of v.

We generalize the notion of k-templates introduced in [5] in order to show Theorem 1. Let Δ and Σ be two alphabets, and let P be a pattern over Δ, then a *P-template* over Σ is a $2(|P|+1)$-tuple of the form: $[w_0, v_1, w_1, v_2 \ldots, v_{|P|}, w_{|P|}]$ where for all i, $w_i \in \Sigma^*$, $v_i \in \mathbb{Z}^{|\Sigma|}$. A word $w \in \Sigma^*$ *realizes* (or *is a realization* of) a P-template $t = [w_0, v_1, \ldots, v_{|P|}, w_{|P|}]$ if there are $u_1, \dots, u_{|P|} \in \Sigma^+$ such that $w = w_0u_1w_1u_2w_2 \dots u_{|P|}w_{|P|}$ and $\forall i,j\ P_i = P_j \implies \Psi(u_i) - \Psi(u_j) = v_i - v_j$.

Each template can be associated to its set of realizations, but many templates are associated to the same set. We say that a P-template is *normalized* if for all $i \in [1, |P|]$, $\varphi_P(i) = i \implies v_i = \overrightarrow{0}$. For any P-template, we can compute a normalized template that is realized by the same set of words. Since one doesn't change the set of realizations by adding the same vector to all vectors corresponding to the same letter, one get the *normalization* of a template by taking for all i $v'_i = v_i - v_{\varphi_P(i)}$ and $w'_i = w_i$. Note that there is a natural bijection between the set of k-templates from [5] and the set of normalized A^k-templates. In the following we only use normalized templates.

We say that a morphism $h : \Sigma^* \to \Sigma^*$ is *convenient* if its associated matrix M_h is invertible, $\|M_h^{-1}\|_2 < 1$ and $\forall a \in \Sigma$, $|h(a)| > 1$. We can now state the main theorem:

▶ **Theorem 1.** *For any alphabets Δ and Σ, pattern $P \in \Delta^*$, P-template t, convenient morphism $h : \Sigma^* \mapsto \Sigma^*$ and any letter $a \in \Sigma$ such that $h(a) = as$ for some s, it is possible to decide if $h^\omega(a)$ avoids t.*

By definition, w avoids P if and only if w avoids the P-template $[\varepsilon, \overrightarrow{0}, \varepsilon, \dots, \overrightarrow{0}, \varepsilon, \overrightarrow{0}, \varepsilon]$. From that we can deduce the following corollary:

▶ **Corollary 2.** *For any alphabets Δ and Σ, pattern $P \in \Delta^*$, any convenient morphism $h : \Sigma^* \mapsto \Sigma^*$ and any letter $a \in \Sigma$ such that $h(a) = as$ for some s, it is possible to decide if $h^\omega(a)$ is abelian-P-free.*

The rest of this section is devoted to the proof of Theorem 1. The main idea of the proof is that we can compute S, a set of templates, such that $t \in S$ and $h^{n+1}(a)$ avoids any template of S if and only if $h^n(a)$ avoids any template of S. Thus $h^\omega(a)$ avoids t if and only if a avoids any template of S which is easy to check. The set S corresponds to what we call *the set of special ancestors*. In the following Δ will always be the alphabet of patterns and Σ the alphabet of words and templates.

2.1 Parents and ancestors of a template

Let $t = [w_0, v_1, \dots, v_{|P|}, w_{|P|}]$ and $t' = [w'_0, v'_1, \dots, v'_{|P|}, w'_{|P|}]$ be two normalized P-templates. We say that t' *is a parent of* t *by* h if there are $p_0, s_0, \dots, p_{|P|}, s_{|P|} \in \Sigma^*$ such that:

- $\forall i \in [0, |P|]$, p_i is a prefix of the image of the first letter of w'_i (a prefix of $h([w'_i]_1)$), s_i is a suffix of the image of the last letter of w'_i and $h(w'_i) = p_iw_is_i$,
- $\forall i,j \in [1, |P|]$, $P_i = P_j \implies v_i - v_j = (\Psi(s_{i-1}) + M_hv'_i + \Psi(p_i)) - (\Psi(s_{j-1}) + M_hv'_j + \Psi(p_j))$.

For any normalized template t we denote $\mathrm{Par}_h(t)$ the set of parents of t by h. The *ancestors of* t *by* h is the set $\mathrm{Ancestors}_h(t) = \cup_{i=0}^{\infty} \mathrm{Par}_h^i(t)$. The relation "is an ancestor" is the transitive and reflexive closure of the relation "is a parent".

Lemmas 3, 4 and 5 tell us that the set of ancestors of a template is computable.

▶ **Lemma 3.** *For any convenient morphism $h : \Sigma^* \mapsto \Sigma^*$ and normalized P-template t, the set $\mathrm{Par}_h(t)$ is finite and computable.*

Proof. Since the template t is normalized we know that:

$$M_hv'_i = \begin{cases} 0 & \text{if } \varphi_P(i) = i \\ v_i - \Psi(s_{i-1}) - \Psi(p_i) + \Psi(s_{\varphi_P(i)-1}) + \Psi(p_{\varphi_P(i)}) & \text{if } \varphi_P(i) \neq i. \end{cases}$$

Since M_h is invertible, there is at most one parent for a given valuation of $(w'_i)_{0\leq i\leq|P|}$, $(s_i)_{0\leq i\leq|P|}$ and $(p_i)_{0\leq i\leq|P|}$. Moreover the possibilities for the s_i, p_i and hence for the w'_i are finite (if h is injective there is at most one possibility for each w'_i). So we can try all the valuations for $(w'_i)_{0\leq i\leq|P|}$, $(s_i)_{0\leq i\leq|P|}$ and $(p_i)_{0\leq i\leq|P|}$ and we get all the parents. ◀

▶ **Lemma 4.** *For any convenient morphism $h : \Sigma^* \mapsto \Sigma^*$ and normalized P-template t there are $(r_1, \dots, r_{|P|}) \in \mathbb{R}^+$ such that if $t' = [w'_0, v'_1, w'_1, v'_2 \dots, v_{|P|}, w_{|P|}]$ is an ancestor of t by h then for all i $\|v'_i\|_2 < r_i$.*

We omit the details of the proof of Lemma 4 which is similar to the proof of Lemma 4 in [5]. Let v_i be the i-th vector of t, then $v'_i = M^{-n}v_i + \sum_{j=0}^{n-1} M^{-j}(\Psi(s_j) + \Psi(p_j) - \Psi(s'_j) - \Psi(p'_j))$ for some s_j, s'_j and p_j, p'_j being respectively suffixes and prefixes of images of letters by h. Moreover $\|M_h^{-1}\|_2 < 1$, so $\|v'_i\|_2$ is bounded.

▶ **Lemma 5.** *For any normalized P-template t the set of ancestors of t by h is finite and computable.*

Proof. Let $t' = [w_0, v_1, w_1, v_2 \dots, v_{|P|}, w_{|P|}]$ be an ancestor of t by h. From Lemma 4, each of the v_i is bounded and since $v_i \in \mathbb{Z}^{|\Sigma|}$ there are finitely many choices for each of the v_i. Moreover, since for all $a \in \Sigma$, $|h(a)| > 1$, the length of the w_i is bounded and there are finitely many different values for the w_i. It implies that there are only finitely many possible ancestors.

In order to compute the set of ancestors, one starts with the singleton $S = \{t\}$ and repeats the operation $S = S \cup \mathrm{Par}_h(S)$ (computable thanks to Lemma 3) until S reaches a fixed point, which will eventually happen since the set of ancestors is finite. ◀

▶ **Lemma 6.** *For any word w and any P-templates t and $t' \in \mathrm{Par}_h(t)$, if w is a realization of t' then $h(w)$ contains a realization of t.*

Proof. Let $t = [w_0, v_1, \dots, v_{|P|}, w_{|P|}]$ and $t' = [w'_0, v'_1, \dots, v'_{|P|}, w'_{|P|}] \in \mathrm{Par}_h(t)$. Then by definition there are $p_0, s_0, \dots, p_{|P|}, s_{|P|} \in \Sigma^*$ such that:

- $\forall i \in [1, |P|]$, $h(w'_i) = p_i w_i s_i$,
- $\forall i, j \in [1, |P|]$, $P_i = P_j \implies v_i - v_j = (\Psi(s_{i-1}) + M_h v'_i + \Psi(p_i)) - (\Psi(s_{j-1}) + M_h v'_j + \Psi(p_j))$.

Assume there is a word w realizing t'. Then there are $u'_1, \dots, u'_{|P|} \in \Sigma^+$ such that $w = w'_0 u'_1 w'_1 u'_2 w'_2 \dots u'_{|P|} w'_{|P|}$ and $\forall i, j$ $P_i = P_j \implies \Psi(u'_i) - \Psi(u'_j) = v'_i - v'_j$. Then $h(w) = p_0 w_0 s_0 h(u'_1) p_1 w_1 \dots s_{|P|-1} h(u'_{|P|}) p_{|P|} w_{|P|} s_{|P|}$. For all i let $u_i = s_{i-1} h(u'_i) p_i$, then $W = w_0 u_1 w_1 u_2 w_2 \dots u_{|P|} w_{|P|}$ is a factor of $h(w)$.

Moreover for all $i, j \in [1, |P|]$, if $P_i = P_j$ then:

$$\begin{aligned}
\Psi(u_i) - \Psi(u_j) &= \Psi(s_{i-1} h(u'_i) p_i) - \Psi(s_{j-1} h(u'_j) p_j) \\
&= (\Psi(s_{i-1}) + \Psi(h(u'_i)) + \Psi(p_i)) - (\Psi(s_{j-1}) + \Psi(h(u'_j)) + \Psi(p_j)) \\
&= (\Psi(s_{i-1}) + M_h v'_i + \Psi(p_i)) - (\Psi(s_{j-1}) + M_h v'_j + \Psi(p_j)) \\
&= v_i - v_j
\end{aligned}$$

So W realizes t and is a factor of $h(w)$. ◀

It tells us that if one of the ancestors of t is not avoided by $h^n(a)$ for some $n \in \mathbb{N}$, then there is $m > n$ such that t is not avoided by $h^m(a)$.

2.2 Specializations of a template

Let $P \in \Delta^*$ and $L \subseteq \Delta$ then we denote by $P_{|L}$ the pattern which is obtained by deleting from P every letter from $\Delta - L$. For example $ABCBBCCA_{|\{A,C\}} = ACCCA$.

Let $\mathrm{Pos}_{(P,L)} : [1, |P_{|L}|] \mapsto [1, |P|]$ be such that $\mathrm{Pos}_{(P,L)}(i) = \min\{j : |(P_1 \dots P_j)_{|L}| = i\}$, where P_i is the i-th letter of P. $\mathrm{Pos}_{(P,L)}(i)$ is the position of the letter in P that is sent to position i in $P_{|L}$.

Let $t = [w_0, v_1, w_1, \dots, v_{|P|}, w_{|P|}]$ be a P-template and $t_{|L} = [w'_0, v'_1, w'_1, \dots, v'_{|P_{|L}|}, w'_{|P_{|L}|}]$ be a $P_{|L}$-template. We say that $t_{|L}$ is a *L-specialization* of t if there are $(u_i)_{i:P_i \notin L} \in \Sigma^+$ such that:

- $\forall i \; v'_i = v_{\mathrm{Pos}_{(P,L)}(i)}$,
- $\forall i, j, \; P_i = P_j \notin L \implies \Psi(u_i) - \Psi(u_j) = v_i - v_j$,
- $\forall i \; w'_i = w_{i_b} u_{i_b+1} w_{i_b+1} \dots w_{i_e-1} u_{i_e} w_{i_e}$, where $i_b = \mathrm{Pos}_{(P,L)}(i)$ and $i_e = \mathrm{Pos}_{(P,L)}(i{+}1){-}1$.

▶ **Lemma 7.** *Let $P \in \Delta^*$ be a pattern and $L \subseteq \Delta$. For any P-template t and any L-specialization $t_{|L}$ of t if there is a word w realizing $t_{|L}$ then w realizes t.*

We omit the proof of Lemma 7 which is technical but straightforward.

With this definition a P-template t has infinitely many L-specializations, but in most cases the parents of a given L-specialization are included in the L-specializations of the parents. Thus we need to introduce the set of *small L-specializations* in order to keep a finit subset of them. A L-specialization of a P-template t is a *small L-specialization* if, with the notations from the definition of L-specialization, for any $A \in \Delta - L$ there is $i \in [1, |P|]$ such that $P_i = A$ and $|u_i| \leq 2 \cdot \max_{a \in \Sigma} |h(a)|$.

▶ **Lemma 8.** *For any pattern $P \in \Delta^*$, P-template t and $L \subseteq \Delta$ the set of small L-specializations of t is finite and computable.*

Proof. Let $P \in \Delta^*$, $t = [w_0, v_1, w_1, \dots, v_{|P|}, w_{|P|}]$ be a P-template and $L \subseteq \Delta$. Let $t_{|L} = [w'_0, v'_1, w'_1, \dots, v_{|P_{|L}|}, w_{|P_{|L}|}]$ be a small L-specialization of t. Since $t_{|L}$ is a small L-specialization of t, for any letter $A \notin L$ there is an index i_A such that $P_{i_A} = A$ and $|u_{i_A}| \leq 2 \cdot max_{a \in \Sigma} |h(a)|$, and there are only finitely many possible values for the u_{i_A}. Then from the definition for all j, $(P_j = A \text{ and } j \neq i_A) \implies \Psi(u_j) = \Psi(u_{i_A}) + v_{P_j} - v_{P_{i_A}}$. So there are only finitely many possible values for each u_j.

Once we have chosen the $(u_i)_{i:P_i \notin L}$ the w'_i and v'_i are fixed. Hence by trying all the possible values for $(u_i)_{i:P_i \notin L}$ we can compute the set of all small L-specializations of t. ◀

We denote by $\mathrm{SmallSpec}_L(t)$ the set of small L-specializations of a P-template t.

2.3 Special ancestors of a template

The set of *special ancestors* of a P-template t by h is the smallest set of templates containing t and any ancestor or small-L-specialization of any of its element. Let us first show that we can compute this set:

▶ **Theorem 9.** *For any alphabets Δ and Σ, pattern $P \in \Delta^*$, normalized P-template t and convenient morphism $h : \Sigma^* \mapsto \Sigma^*$, one can compute the set of special ancestors of t by h.*

Proof. The following algorithm computes this set for any P, t and h.

```
Input  : $P$, $t$, $h$.
Output: The set $S$ of special ancestors of $t$.
$S = \text{Ancestors}_h(t)$;
for $i = |\Delta| - 1 \dots 0$ do
    for $L \subseteq \Delta$, $|L| = i$ do
        $S = S \cup \text{SmallSpec}_L(S)$;
    end
    $S = S \cup \text{Ancestors}_h(S)$;
end
```

Algorithm 1: How to compute special ancestors.

This algorithm halts because if S is finite at some point then by Lemmas 5 and 8 one can execute $S = S \cup \text{Ancestors}(S)$ and $S = S \cup \text{SmallSpec}_L(S)$ and keep S finite.

For any $D \subseteq L \subseteq \Delta$ and any pattern $P \in \Delta^*$, $(P_{|D})_{|L} = P_{|D}$. So for any L-specialization t_{DL} of a D-specialization t_D of a P-template t, $t_{DL} = t_D$. It implies that at the end for any $L \subseteq \Delta$, every element of S has all of its small L-specializations in S. Since the last operation of the algorithm adds the ancestors, every ancestor of any element of S is in S. ◀

In some reasonable implementation of the algorithm, it is important to use for S a data-structure that allows to check if a template is already in S in logarithmic time. Moreover, we are careful with specialization so that we do not obtain twice the same template by two different paths of specialization (dropping the letter A and then the letter B is the same than dropping B and then A).

2.4 Using special ancestors to decide

Under the conditions of Theorem 1, one can compute the set of special ancestors of a template, thanks to Theorem 9. Now we show that this set allows us to decide if the morphism's fixed point avoids the template.

▶ **Theorem 10.** *For any pattern $P \in \Delta^+$, any normalized P-template t, any convenient morphism h and any word $w \in \Sigma^+$, if there is a factor f of $h(w)$ that realizes t, then there is a factor f' of w that realizes a special ancestor of t.*

In fact we show that f' realizes the parent of an L-specialization of t for some well chosen set L. The only thing we do is to unfold the definitions with this set L.

Proof. Let $t = [w_0, v_1, w_1, \dots, v_{|P|}, w_{|P|}]$ be a normalized P-template and assume there is a factor f of $h(w)$ that realizes t. Then by definition there are $u_1, \dots, u_{|P|} \in \Sigma^+$ such that $f = w_0 u_1 w_1 u_2 w_2 \dots u_{|P|} w_{|P|}$ and

$$\forall i, j \; P_i = P_j \implies \Psi(u_i) - \Psi(u_j) = v_i - v_j. \tag{1}$$

Let us introduce the set L:

$$L = \left\{ A \in \Delta : \forall i, P_i = A \implies |u_i| > 2 \cdot \max_{a \in \Sigma} |h(a)| \right\}. \tag{2}$$

Take the $P_{|L}$-template $t_{|L} = [w'_0, v'_1, w'_1, \dots, v_{|P_{|L}|}, w_{|P_{|L}|}]$ such that:

- $\forall i,\ v'_i = v_{\text{Pos}_{(P,L)}(i)}$,
- $\forall i,\ w'_i = w_{i_b} u_{i_b+1} w_{i_b+1} \dots w_{i_e-1} u_{i_e} w_{i_e}$, where $i_b = \text{Pos}_{(P,L)}(i)$ and $i_e = \text{Pos}_{(P,L)}(i{+}1){-}1$.

From the equality (1) and the definition of L, $t_{|L}$ is a small L-specialization of t. Let $(u'_i)_{1\leq i\leq |P_{|L}|}$ be such that for all i, $u'_i = u_{\mathrm{Pos}_{(P,L)}(i)}$. Then $f = w'_0u'_1w'_1\dots u'_{|P_{|L}|}w'_{|P_{|L}|}$. Then from the equality (1) we can deduce:

$$\forall i,j\ [P_{|L}]_i = [P_{|L}]_j \implies \Psi(u'_i) - \Psi(u'_j) = v'_i - v'_j. \tag{3}$$

So f is a realization of the $P_{|L}$-template $t_{|L}$.

Since f is a factor of $h(w)$ there is a factor f' of w such that $h(f') = p_0fs_{|P_{|L}|}$, where $p_0 \in \text{prefixes}(h(f'_1))$ and $s_{|P_{|L}|} \in \text{suffixes}(h(f'_{|f'|}))$. By construction, for all i, $|u'_i| > 2\cdot \max_{a\in\Sigma}|h(a)|$ so we know that each of the u'_i contains at least the full image of one letter. So there are $u''_1,\dots,u''_{|P_{|L}|}\in\Sigma^+$, $w''_0,\dots,w''_{|P_{|L}|}\in\Sigma^*$ and $s_0,p_1,s_1,\dots,s_{|P_{|L}|-1},p_{|P_{|L}|}\in\Sigma^*$ such that $f' = w''_0u''_1w''_1u''_2\dots u''_{|P_{|L}|}w''_{|P_{|L}|}$ and for all $i\in[0,|P|]$:

- p_i is a prefix of the image of the first letter of w''_i,
- s_i is a suffix of the image of the last letter of w''_i,
- $h(w''_i) = p_iw'_is_i$,
- $u'_i = s_{i-1}h(u''_i)p_i$.

For all $i\in[1,|P_{|L}|]$, let $v''_i = \Psi(u''_i) - \Psi(u''_{\varphi_{P_{|L}}(i)})$ and let t'' be the $P_{|L}$-template $t'' = [w''_0, v''_1, w''_1, v''_2,\dots,v''_{|P_{|L}|}, w''_{|P_{|L}|}]$. Then t'' is the normalization of the $P_{|L}$-template $[w''_0,\Psi(u''_1),w''_1,\Psi(u''_2),\dots,\Psi(u''_{|P_{|L}|}),w''_{|P_{|L}|}]$ which is realized by f', thus f' is a realization of t''.

From $u'_i = s_{i-1}h(u''_i)p_i$ we get:

$$\Psi(u'_i) = \Psi(s_{i-1}) + M_h\Psi(u''_i) + \Psi(p_i). \tag{4}$$

Let $i,j\in[1,|P_{|L}|]$ such that $[P_{|L}]_i = [P_{|L}]_j$ then $\varphi(i)=\varphi(j)$ and hence

$$\Psi(u''_{\varphi(i)}) = \Psi(u''_{\varphi(j)}). \tag{5}$$

Now if we put all of that together we get:

$$\begin{aligned}v'_i - v'_j =&\Psi(u'_i) - \Psi(u'_j)\text{ (from (3))}\\ =&(\Psi(s_{i-1}) + M_h\Psi(u''_i) + \Psi(p_i)) - (\Psi(s_{j-1}) + M_h\Psi(u''_j) + \Psi(p_j))\text{ (from (4))}\\ =&(\Psi(s_{i-1}) + M_hv''_i + \Psi(p_i)) - (\Psi(s_{j-1}) + M_hv''_i + \Psi(p_j))\text{ (from (5))}\end{aligned}$$

Thus t'' is a parent of $t_{|L}$. So t'' is a parent of a specialization of t and is realized by a factor f' of w. ◀

Theorem 10 together with the fact that, by definition, a special ancestor of a special ancestor of t is itself a special ancestor of t gives:

▶ **Theorem 11.** *For any pattern $P\in\Delta^*$, any normalized P-template t, any convenient morphism h and any letter $a\in\Sigma$, if there is a positive integer n and a factor of $h^n(a)$ that realizes t, then a realizes a special ancestor of t.*

We also need the converse, that is:

▶ **Theorem 12.** *For any pattern $P\in\Delta^*$, any normalized P-template t, any convenient morphism h and any letter $a\in\Sigma$, if a realizes a special ancestor t' of t, then there is a positive integer n and a factor of $h^n(a)$ that realizes t'.*

Proof. We first take the sequence of parent and L-specialization that reaches t' from t. Then we use Lemmas 6 and 7 to reverse operations on a and we reach the factor of $h^n(a)$ that realizes t. ◀

From Theorems 11 and 12 we deduce the following one:

▶ **Theorem 13.** *For any pattern $P \in \Delta^*$, any normalized P-template t, any convenient morphism h and any letter $a \in \Sigma$, $h^\omega(a)$ avoids t if and only if a does not realize any special ancestor of t.*

Since we can compute the set of special ancestors and compare it to the letter a, we can decide if $h^\omega(a)$ avoids t. We implemented this algorithm in c++ and thus we can check if a pattern is avoided by the fixed point of a morphism.

3 Abelian avoidability of patterns

Patterns are words so we can say that a pattern avoids another pattern in the abelian sense. Moreover, for two patterns P, P' and word w, if P' is not abelian-P-free and the word w is abelian-P-free, then w is abelian-P'-free. It means that if P' is not abelian-P-free, then $\lambda_a(P') \leq \lambda_a(P)$. For instance, since $\lambda_a(AA) = 4$ for any $P \in \{A, B\}^*$, if $P \notin \{A, B, AB, BA, ABA, BAB\}$, then $\lambda_a(P) \leq 4$ because all the other binary patterns are not abelian-AA-free. So every binary pattern is either abelian-4-avoidable or abelian unavoidable. It is interesting to know which of them have avoidability index 2 or 3.

▶ **Theorem 14.** *Binary patterns of length greater than 8 are abelian-3-avoidable. More precisely every pattern that does not appear up to symmetry on the following list is abelian-3-avoidable:*
A, AA, AB, AAB, ABA, $AABA$, $AABB$, $ABAB$, $ABBA$, $AABAA$, $AABAB$, $AABBA$, $ABAAB$, $ABABA$, $AABAAB$, $AABABA$, $AABABB$, $AABBAA$, $ABAABA$, $AABAABA$, $AABABAA$, $ABBABBA$, $AABAABAA$, $ABAABAAB$.

Proof. It is well known that AAA is abelian-3-avoidable [7] and it is already enough to show the upper bound. Moreover, we can use the algorithm from Theorem 1 to show that any fixed point of $a \mapsto aabaac$, $b \mapsto cbbbab$, $c \mapsto cbccac$ is abelian-$AABBAB$-free. So we only need to find exhaustively all the words that avoid AAA, $AABBAB$ and $ABAABB$. This gives the list of Theorem 14. ◀

Conversely, if there is a word that avoids $AABAA$, there is also a recurrent word w that avoids $AABAA$ and then w avoids AA, thus $\lambda_a(AABAA) = 4$. So the patterns $A, AA, AB, AAB, ABA, AABA, AABAA$ are not abelian-3-avoidable. But, for the rest of the list, we do not know which of them are abelian-3-avoidable

▶ **Problem 1.** *Which of the following patterns are abelian-3-avoidable?*
$ABAB$, $ABBA$, $AABAB$, $AABBA$, $ABAAB$, $ABABA$,$AABAAB$, $AABABA$, $AABABB$, $AABBAA$, $ABAABA$, $AABAABA$, $AABABAA$, $ABBABBA$, $AABAABAA$, $ABAABAAB$.

There is a direct link with the following question:

▶ **Problem 2** (Mäkelä (see [12])). *Can you avoid abelian squares of the form uv where $|u| \geq 2$ over three letters ? - Computer experiments show that you can avoid these patterns at least in words of length 450.*

If the answer to the question from Mäkelä is positive then all the patterns from Problem 1 are abelian-3-avoidable. In [18] we showed that abelian squares of the form uv where $|u| \geq 6$ are avoidable over three letters.

Abelian-2-avoidability. For the binary case it was showed in [6] that:

▶ **Theorem 15** (J. D. Currie, T. I. Visentin). *Binary patterns of length greater than 118 are abelian-2-avoidable.*

They also asked:

▶ **Problem 3** (J. D. Currie, T. I. Visentin). *Characterize which binary patterns are abelian-2-avoidable.*

Using the algorithm from Theorem 1 we can improve this result and lower the 118 to 14. First we use the algorithm to check that:

▶ **Lemma 16.** *The fixed point of the morphisms on the left avoid in the abelian sense the corresponding patterns in the right:*

morphisms	*avoided patterns*
$a \mapsto aabaa$ $b \mapsto bbabb$	$AABBBAAAB$, $ABAAABBBA$, $AAABABABBB$, $AAABABBABB$,$AAABABBBAB$, $AABBBABAAB$, $AABBBABABA$, $ABAABABBBA$, $ABAABBBABA$, $ABABAABBBA$, $ABBBABAAAB$, $AABAABBBAB$, $AABBBAABAB$, $AABBBAABAAB$, $AAABABBAAAB$, $AABBBABBBAA$, $ABABABBBABA$, $ABABBABBABA$, $AAABAAABBAB$, $AAABBABAAAB$, $AAABAABAABAB$, $AAABABAAABAB$, $AABAAABABAAB$, $AAABAAABABBA$, $AAABAABABAAB$, $AAABABAABAAB$, $ABBABAAABAAB$, $ABABBBABBBABA$.
$a \mapsto aaaab$ $b \mapsto abbab$	$ABAABBBAAB$, $AAABBABABB$, $AAABBABBAB$, $AABAABBABB$, $AABABABBBA$, $AABABBABBA$, $AABABBBAAB$, $AABABBBABA$, $AABBAABBBA$, $AABBABABBA$, $AABBABBAAB$, $AABBABBABA$, $AABBBAABBA$, $ABAABBABBA$, $AABBABABBBA$, $AABABBBABBBA$,
$a \mapsto abb$ $b \mapsto aaab$	$AAAA$, $AAABAABBB$, $AAABBBABB$, $AABBABBBA$, $AABBBABBA$, $AAABBAAABB$, $AABABAAABB$, $ABBBAABBBA$, $AAABAABBAB$, $AAABAABAABB$, $AAABBAABAAB$, $AABAABAABBA$, $AABAABBAAAB$, $AABABABAAAB$, $AAABBAAABAB$, $AABAAABABAB$, $AABAAABBAAB$, $AAABAABAAABAB$,
$a \mapsto aaab$ $b \mapsto bbba$	$AAABABBBAA$, $AAABBAABBB$, $AAABBABBAA$, $ABABAAABBB$, $ABABBBAABBA$, $AABABBAAABA$, $AABBABAAABA$,
$a \mapsto abaa$ $b \mapsto babb$	$AABBABBABBA$, $AABABBABBBA$, $AABBBABBBABA$, $ABABBABBABBA$, $ABABBABBBABA$, $ABABBBABABBA$, $ABBABABBABBA$,
$a \mapsto aaaba$ $b \mapsto babbb$	$ABAABBBAAA$, $AABABBBAAA$,
$a \mapsto aababbaaaba$ $b \mapsto babbbaababb$	$AABAAABAAABAB$, $ABBBABBBABBBA$, $AAABAAABAAABAAA$,

It implies that, if a pattern contains any of the pattern from Lemma 16, then it can be avoided by a binary word. One can easily check that any binary pattern of length greater than 14 contains at least one of the patterns from the Lemma 16. It implies:

▶ **Theorem 17.** *Binary patterns of length greater than 14 are abelian-2-avoidable.*

In fact, up to symmetry, there are only 284 patterns that avoid all patterns of Lemma 16.

▶ **Theorem 18.** *The patterns from the following list are abelian-2-unavoidable: A, AA, AB, AAA, AAB, ABA, $AAAB$, $AABA$, $AABB$, $ABAB$, $ABBA$, $AAABA$, $AAABB$, $AABAA$, $AABAB$, $AABBA$, $ABAAB$, $ABABA$, $ABBBA$, $AAABAA$, $AAABAB$, $AABAAB$, $ABAAAB$, $AAABAAA$.*

Proof. Let assume that $AAABAAA$ is abelian-2-avoidable, then we can find a recurrent word that avoids $AAABAAA$ in the abelian sense and this words necessarily avoids AAA which is not possible. Thus $AAABAAA$ is abelian-2-unavoidable.

For all the other patterns one can do an exhaustive search and check that they are abelian-2-unavoidable. ◀

For the 260 other patterns we don't know which are abelian-2-avoidable and which are not. For most of them there is probably no fixed point of a binary morphism avoiding them, but they could be avoided by the image of a fixed point by a second morphism.

We are left with some interesting questions:

▶ **Problem 4.** *What is the length of the longest abelian-2-unavoidable binary pattern?*

We know that the answer is between 7 and 14.

▶ **Problem 5.** *What is the exact list of the abelian-2-unavoidable binary patterns?*

It is probably related somehow to the question 6 which seems really hard.

▶ **Problem 6** (Mäkelä (see [12])). *Can you avoid abelian-cubes of the form uvw where $|u| \geq 2$, over two letters ? - You can do this at least for words of length 250.*

It was showed in [19] that the answer to Problem 6 is negative. But we can replace the 2 in the question by any integer. In particular a proof that abelian cubes of the form uvw where $|u| \geq 3$ are avoidable over two letters would imply that many of the 284 patterns are also abelian-2-avoidable.

Finally we have some more general questions:

▶ **Problem 7.** *For any finite alphabet Δ is it true that:*
- $\exists n \in \mathbb{N}$ *such that any pattern over Δ of length greater than n is abelian-avoidable?*
- $\exists n \in \mathbb{N}$ *such that any pattern over Δ of length greater than n is abelian-$|\Delta|$-avoidable?*
- $\exists n \in \mathbb{N}$ *such that any pattern over Δ of length greater than n is abelian-2-avoidable?*

Acknowledgements. The author would like to thank Pascal Ochem and Michaël Rao for valuable discussions.

References

1 K. A. Baker, G. F. McNulty, and W. Taylor. Growth problems for avoidable words. *Theoretical Computer Science*, 69(3):319–345, 1989. doi:10.1016/0304-3975(89)90071-6.

2 D. R. Bean, A. Ehrenfeucht, and G. F. McNulty. Avoidable patterns in strings of symbols. *Pacific J. Math.*, 85(2):261–294, 1979.

3 F. Blanchet-Sadri and B. Woodhouse. Strict bounds for pattern avoidance. *Theoretical Computer Science*, 506:17–27, 2013.

4 J. Cassaigne. Unavoidable binary patterns. *Acta Informatica*, 30(4):385–395. doi:10.1007/BF01209712.

5 J. D. Currie and N. Rampersad. Fixed points avoiding abelian k-powers. *Journal of Combinatorial Theory, Series A*, 119(5):942–948, July 2012. doi:10.1016/j.jcta.2012.01.006.

6 J. D. Currie and T. I. Visentin. Long binary patterns are abelian 2-avoidable. *Theoretical Computer Science*, 409(3):432–437, 2008. doi:10.1016/j.tcs.2008.08.039.

7 F. M. Dekking. Strongly non-repetitive sequences and progression-free sets. *Journal of Combinatorial Theory, Series A*, 27(2):181–185, 1979. `doi:10.1016/0097-3165(79)90044-X`.

8 P. Erdős. Some unsolved problems. *The Michigan Mathematical Journal*, 4(3):291–300, 1957. `doi:10.1307/mmj/1028997963`.

9 P. Erdős. Some unsolved problems. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 6:221–254, 1961.

10 A. A. Evdokimov. Strongly asymmetric sequences generated by a finite number of symbols. *Dokl. Akad. Nauk SSSR*, 179:1268–1271, 1968.

11 V. Keränen. Abelian squares are avoidable on 4 letters. In *ICALP*, pages 41–52, 1992.

12 V. Keränen. New abelian square-free DT0L-languages over 4 letters. *Manuscript*, 2003.

13 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.

14 R. Mercas, P. Ochem, A. V. Samsonov, and A. M. Shur. Binary patterns in binary cube-free words: Avoidability and growth. *RAIRO – Theor. Inf. and Applic*, 48(4):369–389, 2014.

15 P. Ochem. Doubled patterns are 3-avoidable. *Electron. J. Combinatorics.*, 23(1), 2016.

16 P. Ochem and A. Pinlou. Application of entropy compression in pattern avoidance. *Electron. J. Combinatorics.*, 21(2), 2014.

17 P. A. B. Pleasants. Non-repetitive sequences. *Mathematical Proceedings of the Cambridge Philosophical Society*, 68:267–274, 9 1970. `doi:10.1017/S0305004100046077`.

18 M. Rao and M. Rosenfeld. On Mäkelä's Conjectures: deciding if a morphic word avoids long abelian-powers. *ArXiv e-prints*, November 2015. `arXiv:1511.05875`.

19 M. Rao and M. Rosenfeld. Avoidability of long k-abelian repetitions. *Mathematics of Computation*, in press 2015.

20 P. Roth. Every binary pattern of length six is avoidable on the two-letter alphabet. *Acta Informatica*, 29(1):95–107. `doi:10.1007/BF01178567`.

21 U. Schmidt. Avoidable patterns on two letters. *Theoretical Computer Science*, 63(1):1–17, 1989. `doi:10.1016/0304-3975(89)90064-9`.

22 A. Thue. Über unendliche Zeichenreihen. *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiania,*, 1906.

23 A. Thue. Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen. *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiania,*, 10:1–67, 1912.

24 A. I. Zimin. Blocking sets of terms. *Sbornik: Mathematics*, 47(2):353–364, 1984.

Bounded Depth Circuits with Weighted Symmetric Gates: Satisfiability, Lower Bounds and Compression[*][†]

Takayuki Sakai[1], Kazuhisa Seto[2], Suguru Tamaki[3], and Junichi Teruyama[4]

1 **Oki Electric Industry Co., Ltd.**
2 **Seikei University, 3-3-1 Kichijoji-Kitamachi, Musashino-shi, Tokyo 180-8633, Japan**
`seto@st.seikei.ac.jp`
3 **Kyoto University, Yoshida Honmachi, Sakyo-ku, Kyoto 606-8501, Japan**
`tamak@kuis.kyoto-u.ac.jp`
4 **National Institute of Informatics, and JST, ERATO, Kawarabayashi Large Graph Project, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan**
`teruyama@nii.ac.jp`

Abstract

A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *weighted symmetric* if there exist a function $g : \mathbb{Z} \to \{0,1\}$ and integers $w_0, w_1, \ldots, w_n$ such that $f(x_1, \ldots, x_n) = g(w_0 + \sum_{i=1}^{n} w_i x_i)$ holds.

In this paper, we present algorithms for the circuit satisfiability problem of bounded depth circuits with AND, OR, NOT gates and a limited number of weighted symmetric gates. Our algorithms run in time super-polynomially faster than 2^n even when the number of gates is super-polynomial and the maximum weight of symmetric gates is nearly exponential. With an additional trick, we give an algorithm for the maximum satisfiability problem that runs in time $\mathrm{poly}(n^t) \cdot 2^{n-n^{1/O(t)}}$ for instances with n variables, $O(n^t)$ clauses and *arbitrary* weights. To the best of our knowledge, this is the first moderately exponential time algorithm even for Max 2SAT instances with arbitrary weights.

Through the analysis of our algorithms, we obtain average-case lower bounds and compression algorithms for such circuits and worst-case lower bounds for majority votes of such circuits, where all the lower bounds are against the generalized Andreev function. Our average-case lower bounds might be of independent interest in the sense that previous ones for similar circuits with arbitrary symmetric gates rely on communication complexity lower bounds while ours are based on the restriction method.

1998 ACM Subject Classification F.2.0 [Analysis of Algortihms and Problem Complexity] General

Keywords and phrases exponential time algorithm, circuit complexity, circuit minimization, maximum satisfiability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.82

[*] Subsumes the technical report [50]. Due to the page limit, we omit many proofs, which can be found in the full version of the paper.

[†] Supported in part by MEXT KAKENHI (24106003); JSPS KAKENHI (26330011, 26730007, 16H02782); JST, ERATO, Kawarabayashi Large Graph Project; the John Mung Advanced Program of Kyoto University. Part of the work performed while ST was at Department of Computer Science and Engineering, University of California, San Diego and the Simons Institute for the Theory of Computing, Berkeley.

1 Introduction

We are concerned with bounded depth circuits with AND, OR, NOT and (weighted) symmetric gates. Let $\mathbb{Z}$ be the set of integers and $x_1, x_2 \ldots, x_n$ be Boolean variables. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *weighted symmetric* if there exist a function $g : \mathbb{Z} \to \{0,1\}$ and integers $w_0, w_1, \ldots, w_n$ such that $f(x_1, \ldots, x_n) = g(w_0 + \sum_{i=1}^{n} w_i x_i)$ holds. If $w_1 = w_2 = \cdots = w_n = 1$ holds, then f is *symmetric*.

For example, if we set $g(z) = \mathrm{sgn}(z)$, where $\mathrm{sgn}(z) = 1$ if and only if $z \geq 0$, we obtain *majority* functions as symmetric functions and *linear threshold* functions as weighted symmetric functions. If we define $g(z) = 1$ if and only if $z \equiv 0 \bmod m$ for an integer $m \geq 2$, then we obtain *modulo* m functions as symmetric functions.

A (weighted) symmetric gate is a logic gate that computes a (weighted) symmetric function. We denote by $\mathbf{SYM}_w$ the set of weighted symmetric gates such that $\max_i |w_i| \leq w$ holds. When we consider satisfiability and compression algorithms, we assume that $g(z)$ can be evaluated in time polynomial in $\log_2 |z|$, where $|z|$ denotes the absolute value of z. When we consider circuit lower bounds, we assume that g is computable, i.e., there exists a Turing machine that computes g.

1.1 Our contribution

Satisfiability Algorithms: In the *circuit satisfiability problem* (Circuit SAT), our task is, given a Boolean circuit C, to decide whether there exists a 0/1 assignment to the input variables such that C evaluates 1. If input instances are restricted to a class of Boolean circuits $\mathcal{C}$, the problem is called $\mathcal{C}$-SAT. A naïve algorithm can solve Circuit SAT in time $O(\mathrm{poly}(|C|) \cdot 2^n)$, where we denote by $|C|$ the size of C and by n the number of input variables of C respectively. We say an algorithm for $\mathcal{C}$-SAT is *moderately exponential time* if it checks the satisfiability of every $C \in \mathcal{C}$ in time $\mathrm{poly}(|C|) \cdot 2^{n-\omega(\log n)}$, i.e., super-polynomially faster than 2^n. We are interested in for which class $\mathcal{C}$ moderately exponential time satisfiability algorithms exist.

Let $\mathbf{SYM}_w \circ \mathbf{AND}(n,m)$ be the set of n-variate depth 2 circuits with a weighted symmetric gate in $\mathbf{SYM}_w$ at the top and at most m AND gates at the bottom. Let $\mathbf{SYM}_w \circ \mathbf{AC}^0_d(n,m)$ be the set of n-variate unbounded fan-in depth $d+1$ layered circuits with AND, OR, NOT gates and a weighted symmetric gate in $\mathbf{SYM}_w$ such that the top gate is the weighted symmetric gate and each layer contains at most m gates. Let $\mathbf{AC}^0_d[\mathbf{SYM}_w](n,m,t)$ be the set of n-variate unbounded fan-in depth d layered circuits with AND, OR, NOT gates and at most t weighted symmetric gates in $\mathbf{SYM}_w$ such that each layer contains at most m gates.

In this paper, we show moderately exponential time algorithms for the counting version of $\mathcal{C}$-SAT, where $\mathcal{C} \in \{\mathbf{SYM}_w \circ \mathbf{AND}(n,m), \mathbf{SYM}_w \circ \mathbf{AC}^0_d(n,m), \mathbf{AC}^0_d[\mathbf{SYM}_w](n,m,t)\}$, as follows.

▶ **Theorem 1** (depth 2, weighted symmetric gate at the top, AND gates at the bottom)**.** *We can count the number of satisfying assignments for $C \in \mathbf{SYM}_w \circ \mathbf{AND}(n,m)$ deterministically in time*

$$\mathrm{poly}(n,m,\log w) \cdot 2^{n-\Omega\left((n/\log(mw))^{\log n/4\log(nm)}\right)}$$

and exponential space.

The running time is super-polynomially faster than 2^n when, e.g., $m = n^{o(\log n/\log\log n)}$ and $w = 2^{n^{0.99}}$. Note that $\mathbf{SYM}_{2^n}$ contains all Boolean functions (if we ignore the assumption that $g(z)$ can be evaluated in time polynomial in $\log_2 |z|$). The heart of our algorithms is a (seemingly new) *bottom fan-in reduction* technique inspired by recent developments on the analysis of "greedy restriction" by "concentrated shrinkage" [51, 54, 17, 49]. With an additional trick, we give an algorithm for the maximum satisfiability problem that runs in time $\mathrm{poly}(n^t) \cdot 2^{n-n^{1/O(t)}}$ for instances with n variables, $O(n^t)$ clauses and *arbitrary* weights. To the best of our knowledge, this is the first moderately exponential time algorithm even for Max 2SAT instances with arbitrary weights.

We extend the above algorithm with the help of the depth reduction algorithm due to Beame, Impagliazzo and Srinivasan [7].

▶ **Theorem 2** (depth d, weighted symmetric gate only at the top)**.** *We can count the number of satisfying assignments for $C \in \mathbf{SYM}_w \circ \mathbf{AC}^0_d(n,m)$ deterministically in time*

$$\mathrm{poly}(n,m,\log w) \cdot 2^{n-\Omega\left((n/2^{2d(\log m)^{4/5}}\log(mw))^{\log n/9\log m}\right)}$$

and exponential space.

The running time is super-polynomially faster than 2^n when, e.g., $m = 2^{(\log n/4d)^{5/4}}$ and $w = 2^{n^{0.49}}$.

We further extend the above algorithm relying on the circuit transformation techniques due to Beigel, Reingold and Spielman [9] and Beigel [8].

▶ **Theorem 3** (depth d, $t(n)$ weighted symmetric gates)**.** *We can count the number of satisfying assignments for $C \in \mathbf{AC}^0_d[\mathbf{SYM}_w](n,m,t)$ deterministically in time*

$$\mathrm{poly}(n,m,d,t,\log w) \cdot 2^{n+O(t\log mw)-\Omega\left((n/2^{4d(\log m)^{4/5}}t\log(mw))^{\frac{\log n}{18\log m}}\right)}$$

and exponential space.

The running time is super-polynomially faster than 2^n when, e.g., $m = n^c$, $w = 2^{n^{\frac{1}{40c}}}$ and $t = n^{\frac{1}{40c}}$, where $c \leq \frac{\log^{1/4} n}{2(4d)^{5/4}}$.

Although our algorithms run in time super-polynomially faster than 2^n instead of exponentially faster than 2^n ($2^{(1-\varepsilon)n}$ for a universal constant $\varepsilon > 0$), this seems unavoidable due to the Strong Exponential Time Hypothesis (SETH) [12, 32, 34]: The hypothesis states that for all k, there exists $\varepsilon_k > 0$ such that the satisfiability problem of k-CNF formulas cannot be solved in time $2^{(1-\varepsilon_k)n}$. SETH has been used in proving conditional time lower bounds for several exponential time and polynomial time algorithms, see, e.g., [21, 37, 40].

Circuit Lower Bounds: Through the analysis of our satisfiability algorithms, we obtain the following average-case lower bounds.

▶ **Theorem 4** (depth 2, weighted symmetric gate at the top, AND gates at the bottom)**.** *There exists a constant $\alpha > 0$ such that for every m, w and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w}$ such that for every $C \in \mathbf{SYM}_w \circ \mathbf{AND}(n,m)$, it holds that*

$$\Pr_{x\in\{0,1\}^n}[f(x) = C(x)] \leq \frac{1}{2} + 2^{-\Omega\left((n/\log(mw))^{\alpha\log n/\log(nm)}\right)}.$$

We also obtain similar average-case lower bounds for $\mathbf{SYM}_w \circ \mathbf{AC}^0_d(n,m)$ and $\mathbf{AC}^0_d[\mathbf{SYM}_w](n,m,t)$, see Theorems 12 and 13 in Section 5.

Our average-case lower bounds might be interesting in the sense that (1) previous ones for similar circuits with arbitrary symmetric gates rely on communication complexity lower bounds while ours are based on the restriction method and (2) we are not aware of (even worst-case) lower bounds for $\mathbf{SYM}_w \circ \mathbf{AND}$ with $w = n^{\omega(\log n)}$.

Let $\mathcal{C}$ be a set of Boolean circuits and $\mathbf{MAJ} \circ \mathcal{C}$ be the set of Boolean circuits, where $C \in \mathbf{MAJ} \circ \mathcal{C}$ is a majority vote of $\mathcal{C}$ circuits, i.e., $C(x) = \mathrm{sgn}(C_1(x) + \cdots + C_s(x) + w_0)$ holds for some $C_1, \ldots, C_s \in \mathcal{C}$ and an integer w_0.

Combining the above average-case lower bounds and the discriminator lemma due to Hajnal, Maass, Pudlák, Szegedy and Turán [27], we obtain the following worst-case lower bounds.

▶ **Theorem 5** (majority vote of depth 2, weighted symmetric gate at the top, AND gates at the bottom). *There exists a constant $\alpha > 0$ such that for every m, w and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w}$ such that any $C \in \mathbf{MAJ} \circ \mathbf{SYM}_w \circ \mathbf{AND}(n, m)$ cannot compute $f_{n,m,w}$ if the majority gate at the top of C has fan-in at most $2^{o\left((n/\log(mw))^{\alpha \log n/\log(nm)}\right)}$.*

We also obtain similar worst-case lower bounds for $\mathbf{MAJ} \circ \mathbf{SYM}_w \circ \mathbf{AC}^0_d(n, m)$, $\mathbf{MAJ} \circ \mathbf{AC}^0_d[\mathbf{SYM}_w](n, m, t)$ (and $\mathbf{AC}^0_d[\mathbf{SYM}_w](n, m, t)$ with different parameters), see Theorems 24, 25 and 26 in Section 6.

Compression Algorithms: In the *circuit compression problem* (Circuit CMP), our task is, given the truth table of an s-sized Boolean circuit C and an integer $s' \geq s$, to construct a Boolean circuit C' that is at most s'-sized and computes the same function as C. If input instances are restricted to a class of Boolean circuits $\mathcal{C}$, the problem is called $\mathcal{C}$-CMP. In $\mathcal{C}$-CMP, we do not have to construct C' as a circuit in $\mathcal{C}$. Since every n-variate Boolean function can be represented as a $\frac{(1+o(1))2^n}{n}$-sized circuit [39][1], the problem is interesting if $s' \ll 2^n/n$ and in particular we consider the case $s' = 2^{n-\omega(\log n)}$.

A compression algorithm is *efficient* if it runs in time $2^{O(n)}$ given the truth table of an n-variate Boolean function. Note that input length is 2^n and an efficient algorithm runs in polynomial time. The running time analyses of our satisfiability algorithms imply efficient compression algorithms. Let $\mathcal{C} \in \{\mathbf{SYM}_w \circ \mathbf{AND}(n, m), \mathbf{SYM}_w \circ \mathbf{AC}^0_d(n, m), \mathbf{AC}^0_d[\mathbf{SYM}_w](n, m, t)\}$. We obtain deterministic efficient algorithms for $\mathcal{C}$-CMP if parameters n, m, w, d, t are such that the corresponding algorithms for $\mathcal{C}$-SAT run in time $2^{n-\omega(\log n)}$.

1.2 Background

Bounded Depth Circuits with (Weighted) Symmetric Gates: Let $\mathbf{AC}^0$ be the set of bounded depth circuits with AND, OR and NOT gates, $\mathbf{AC}^0[m]$ be the set of $\mathbf{AC}^0$ circuits with modulo m gates, $\mathbf{AC}^0[\mathbf{MAJ}]$ be the set of $\mathbf{AC}^0$ circuits with majority gates (also known as $\mathbf{TC}^0$), $\mathbf{AC}^0[\mathbf{THR}]$ be the set of $\mathbf{AC}^0$ circuits with linear threshold gates and $\mathbf{AC}^0[\mathbf{SYM}_w]$ be the set of $\mathbf{AC}^0$ circuits with gates in $\mathbf{SYM}_w$. Note that for every linear threshold gate, there exists a polynomial size depth 2 majority circuit that computes it [24].

In their seminal work, Razborov [46] and Smolensky [55] showed exponential lower bounds on the size of $\mathbf{AC}^0[m]$ circuits computing majority or mod q functions when m, q are prime powers and relatively prime. Since then, people have been trying to obtain super-polynomial size lower bounds against stronger circuit classes such as $\mathbf{AC}^0[m]$ with arbitrary m or $\mathbf{AC}^0[\mathbf{MAJ}]$. Despite much effort of researchers, super-polynomial size lower bounds have been only shown for such circuit classes with some restriction, see,

[1] Such a representation can be obtained in time $2^{O(n)}$.

e.g., [4, 9, 14, 22, 23, 26, 27, 28] (here we consider circuits computing "explicit" Boolean functions, i.e., functions in NP).

One of the best studied restriction is limiting the number of (weighted) symmetric gates. The following lower bounds are known:

- (Worst-case lower bounds) Exponential lower bounds for $\mathbf{AC}^0[\mathbf{MAJ}]$ circuits with $n^{o(1)}$ majority gates [6, 8] and $\mathbf{AC}^0[\mathbf{THR}]$ circuits with $o(\log n)$ linear threshold gates [44].
- (Average-case lower bounds) super-polynomial lower bounds for $\mathbf{AC}^0[\mathbf{SYM}_1]$ circuits with $o(\log^2 n)$ symmetric gates [58]; arbitrary large polynomial lower bounds for $\mathbf{AC}^0[\mathbf{SYM}_1]$ circuits with $n^{1-o(1)}$ symmetric gates and $\mathbf{AC}^0[\mathbf{THR}]$ circuits with $n^{1/2-o(1)}$ linear threshold gates [38].

The above average-case lower bounds are based on the results of Håstad and Goldmann [29] and Razborov and Wigderson [48] that show average-case lower bounds for $\mathbf{SYM}_1 \circ \mathbf{AND}$ circuits from the communication complexity lower bounds due to Babai, Nisan and Szegedy [5] and also show worst-case lower bounds for $\mathbf{MAJ} \circ \mathbf{SYM}_1 \circ \mathbf{AND}$ circuits using the discriminator lemma.

Circuit Satisfiability: Studying moderately exponential time algorithms for Circuit SAT is motivated by not only the importance in practice, e.g., logic circuit design and constraint satisfaction but also the viewpoint of Boolean circuit complexity. As pointed out by several papers such as [60, 65], there are strong connections between proving circuit lower bounds for $\mathcal{C}$ and designing moderately exponential time algorithms for $\mathcal{C}$-SAT; see also excellent surveys [52, 43, 62]. Typical such connections are:

(1) Some proof techniques such as deterministic/random restriction (shrinkage analysis/switching lemma) simultaneously prove circuit lower bounds for $\mathcal{C}$ and provides $\mathcal{C}$-SAT algorithms [51, 31, 7, 54, 17, 16, 15, 20, 25].

(2) Williams [60, 64] showed that if we obtain a moderately exponential time algorithm for $\mathcal{C}$-SAT and $\mathcal{C}$ satisfies some closure property, then we also have a separation of complexity classes such as $\mathrm{E}^{\mathrm{NP}} \not\subseteq \mathcal{C}$ or $\mathrm{NE} \not\subseteq \mathcal{C}$, where E^{NP} is the set of languages decidable by exponential time Turing machines with NP oracles and NE is the set of languages decidable by non-deterministic exponential time Turing machines; see also [59, 61, 63, 10, 35] for the improvement of such connections. Since then, people have developed moderately exponential time satisfiability algorithms for various circuit classes [33, 18, 30, 1, 3, 2, 42, 19, 57]. In particular, one of the current best lower bounds, $\mathrm{NE} \not\subseteq \mathbf{ACC}^0 \circ \mathbf{THR}$ (also $\mathrm{NE} \not\subseteq \mathbf{ACC}^0 \circ \mathbf{SYM}_1$), was obtained through satisfiability algorithms [63], where $\mathbf{ACC}^0 := \bigcup_m \mathbf{AC}^0[m]$.

Circuit Compression: Circuit CMP is a relaxed version of the circuit minimization problem. Chen, Kabanets, Kolokolova, Shaltiel and Zuckerman [17] established a connection between compression algorithms and circuit lower bounds as follows: If there exists a deterministic efficient algorithm for $\mathcal{C}$-CMP, then $\mathrm{NEXP} \not\subseteq \mathcal{C}$. They also gave efficient compression algorithms for $\mathbf{AC}^0$ circuits, Boolean formulas and branching programs of certain size range. Srinivasan [56] showed an efficient compression algorithm for $\mathbf{AC}^0[m]$ with a prime power m. Carmosino, Impagliazzo, Kabanets and Kolokolova [13] established interesting connections between the tasks of compression/learning and "natural properties" in the sense of Razborov and Rudich [47].

2 Preliminaries

We use random access machines as our computation model. For a set S, we denote by $|S|$ the cardinality of S.

A *literal* is either a Boolean variable or its negation. A *term* is a conjunction of literals. A *Boolean circuit* is a directed acyclic graph whose source nodes are labeled by literals or constants and internal and sink nodes are labeled by logic gates such as AND, OR, NOT, or weighted symmetric gates. A Boolean circuit with a single sink node computes a Boolean function in a natural way. We call source nodes and a sink node *input nodes* and *output node* respectively. The *depth* of a node is defined as the length of the longest path from it to the output node. The *depth* of a Boolean circuit is the maximum value of the depth over all nodes. A Boolean circuit is *layered* if for every edge (u, v), u and v have depth d and $d + 1$ for some d.

A Boolean circuit $C : \{0,1\}^n \to \{0,1\}$ is *satisfiable* if there exists a *satisfying assignment* for C, i.e., an assignment $a \in \{0,1\}^n$ such that $C(a) = 1$ holds. For two Boolean functions (or circuits) f, g in the same variables, we write $f \equiv g$ if $f(a) = g(a)$ holds for all $a \in \{0,1\}^n$. A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *k-junta* if it depends on at most k variables, i.e., there exist $g : \{0,1\}^k \to \{0,1\}$ and $1 \le i_1 < \cdots < i_k \le n$ such that $f(x_1, \ldots, x_n) = g(x_{i_1}, \ldots, x_{i_k})$ holds.

Let $V = \{x_1, \ldots, x_n\}$. A *restriction* is a mapping $\rho : V \to \{0, 1, *\}$. The meaning of ρ is that if $\rho(x_i) \in \{0,1\}$, then we assign the value $\rho(x_i)$ to x_i, and if $\rho(x_i) = *$, then we leave x_i as it is. Thus, when we *apply* a restriction ρ to a Boolean function f, we obtain the Boolean function $f|_\rho$ defined over the variables $\rho^{-1}(*)$. We also apply a restriction ρ to a Boolean circuit C and obtain a Boolean circuit $C|_\rho$. When we apply a restriction ρ to a Boolean circuit C, we *simplify* a Boolean circuit C using the identities $0 \wedge f \equiv 0$, $1 \wedge f \equiv f$ repeatedly (each appearance of L.H.S. is replaced by R.H.S.).

A *restriction decision tree* T over $x_1, \ldots, x_n$ is an ordinary decision tree except that leaves are not necessarily labeled by 0 or 1. The *height* of T is defined as the number of nodes on the longest path from the root to a leaf and the *size* of T is defined as the number of nodes in T. We identify a path from the root to a leaf with a restriction. A *random root-to-leaf path* is sampled by repeatedly selecting a child of the current node uniformly at random from the root. Note that a path of length ℓ is chosen with probability $2^{-\ell}$.

3 A Dynamic Programming Algorithm for $\mathrm{SYM}_w \circ \mathrm{AND}_k$

We denote by $g \circ \mathbf{AND}_k(n, m, w)$ the set of n-variate Boolean circuits of the form $g(w_0 + \sum_{i=1}^{s} w_i t_i)$, where $g : \mathbb{Z} \to \{0,1\}$, $s \le m$, $w_0, w_1, \ldots, w_s \in \mathbb{Z}, \max_{0 \le i \le s} |w_i| \le w$, and $t_1, \ldots, t_s$ are terms that contain at most k-literals such that $t_i \ne t_j$ holds for $i \ne j$. We define

$$\mathbf{SYM}_w \circ \mathbf{AND}_k(n, m) := \bigcup_{g:\mathbb{Z}\to\{0,1\}} g \circ \mathbf{AND}_k(n, m, w).$$

We specify an element C in $\mathbf{SYM}_w \circ \mathbf{AND}_k(n, m)$ as $C = \{g, w_0, (t_1, w_1), \ldots, (t_s, w_s)\}$ and call s and $\max_{0 \le i \le s} |w_i|$ the *size* and the *maximum weight* of C respectively.

For a restriction ρ, we simplify $C|_\rho = \{g, w_0, (t_1|_\rho, w_1), \ldots, (t_s|_\rho, w_s)\}$ repeatedly if there exists a pair (i, j), $1 \le i < j \le s$ such that $t_i|_\rho \equiv t_j|_\rho$ holds. That is, we delete $(t_j|_\rho, w_j)$ and replace $(t_i|_\rho, w_i)$ by $(t_i|_\rho, w_i + w_j)$. If there are multiple such pairs, we may handle them in arbitrary order.

Our first satisfiability algorithm for $\mathbf{SYM}_w \circ \mathbf{AND}_k(n, m)$ is described in Fig. 1. The algorithm involves two parameters n', m' that are specified in the proof of Theorem 6.

The basic idea is as follows:

Step 1: We construct a table T that contains pairs of the form $(C, \#\mathrm{sat}(C))$ for every circuit C in $g \circ \mathbf{AND}_k(n', m', w')$, where $\#\mathrm{sat}(C)$ denotes the number of satisfying assignments

```
Algorithm1($C = \{g, w_0, (t_1, w_1), \ldots, (t_s, w_s)\}$: instance, $n, m, k, w$: integer)
01: if $C \notin \mathbf{SYM}_w \circ \mathbf{AND}_k(n, m)$, return $\perp$.
02: $T \leftarrow \emptyset$. /* table for dynamic programming */
03: for each $C \in g \circ \mathbf{AND}_k(n', m', (s+1) \cdot w)$, /* lexicographical order */
04:     $T \leftarrow T \cup \{(C, \#\mathrm{sat}(C))\}$. /* brute force search */
05: $N \leftarrow 0$.
06: for each $\rho : V \to \{0, 1, *\}$ such that $\rho^{-1}(*) = \{x_1, \ldots, x_{n'}\}$,
07:     $N \leftarrow N + \#\mathrm{sat}(C|_\rho)$. /* binary search in $T$ */
08: return $N$.
```

Figure 1 A Dynamic Programming Algorithm for $\mathbf{SYM}_w \circ \mathbf{AND}_k$.

for C and n', m', w' are appropriately chosen parameters. Furthermore, pairs are sorted in the lexicographical order with respect to the first coordinate C so that we can use binary search. To do so, we check the number of satisfying assignments for every circuit in $g \circ \mathbf{AND}_k(n', m', w')$ one by one in the lexicographical order using brute force search.

Step 2: Let C be an input instance in $g \circ \mathbf{AND}_k(n, m, w)$. For each restriction ρ that assigns $*$ to the first n' variables of C, we check the number of satisfying assignments for $C|_\rho$ using binary search in T and output the sum of them.

We will show the following theorem.

▶ **Theorem 6.** *We can count the number of satisfying assignments for $C \in \mathbf{SYM}_w \circ \mathbf{AND}_k(n, m)$ deterministically in time*

$$\mathrm{poly}(n, m, \log w) \cdot 2^{n - \Omega((n/\log(mw))^{1/k})}$$

and exponential space.

Proof. We denote by $|g \circ \mathbf{AND}_k(n, m, w)|$ the cardinality of $g \circ \mathbf{AND}_k(n, m, w)$. To evaluate the running time of (Step 1), we upper bound the size of the table T using the following fact.

▶ **Fact 7.** *For all m, we have*

$$|g \circ \mathbf{AND}_k(n, m, w)| \leq (2w+1)^{\sum_{i=0}^{k} 2^i \binom{n}{i}} \leq 2^{(k+1)(2n)^k \log(2w+1)}.$$

Proof. Note that $\sum_{i=0}^{k} 2^i \binom{n}{i}$ is the number of different terms that consist of at most k-literals (including a constant function 1). Each term has a weight in $\{-w, -w+1, \ldots, w-1, w\}$. Thus, we have the first inequality. The second inequality follows from an elementary calculation. ◀

Thus, we can bound the running time of Lines 03-04 from above by

$$2^{(k+1)(2n')^k \log(2(m+1)w+1)} \times \mathrm{poly}(m', \log(mw)) \cdot 2^{n'},$$

where we set $m' = \sum_{i=0}^{k} 2^i \binom{n'}{i} \leq (k+1)(2n')^k$.

Next we evaluate the running time of (Step 2). Note that the following guarantees that every $C|_\rho$ in Line 06 belongs to $g \circ \mathbf{AND}_k(n', m', (m+1) \cdot w)$.

▶ **Fact 8.** *Let $C = \{g, w_0, (t_1, w_1), \ldots, (t_m, w_m)\}$. If $C \in g \circ \mathbf{AND}_k(n, m, w)$ holds, then for all restriction ρ with $|\rho^{-1}(*)| = n'$, we have $C|_\rho \in g \circ \mathbf{AND}_k(n', m', (m+1) \cdot w)$.*

Proof. By the definition of $\mathbf{SYM}_w \circ \mathbf{AND}_k(n,m)$, we have $\sum_{i=0}^{s} |w_i| \le (m+1)w$. This implies the maximum weight of $C|_\rho$ is at most $(m+1)w$. ◀

For each $C|_\rho$, binary search in Line 07 takes time at most

$$\log_2 |g \circ \mathbf{AND}_k(n', m', (m+1)\cdot w)| \times \mathrm{poly}(m', \log(mw)) = \mathrm{poly}(m', \log(mw)).$$

Thus, we can bound the running time of Lines 06-07 above by

$$\mathrm{poly}(m, m', \log(mw)) \cdot 2^{n-n'}.$$

If we set $n' = \left(\frac{n}{(k+1)2^{k+1}\log(2(m+1)w+1)}\right)^{1/k} = \Theta((n/\log(mw))^{1/k})$, the total running time of **Algorithm1** is bounded from above by $\mathrm{poly}(n, m, \log w) \cdot 2^{n-\Omega((n/\log(mw))^{1/k})}$. This completes the proof. ◀

▶ Remark. In the case when $g(z) = \mathrm{sgn}(z)$, we can reduce the weight of the top gate of $C|_\rho$ from $(m+1)w$ to $2^{n'^{O(k)}}$ efficiently by Theorem 16 in [41]. With this trick, we can handle Max SAT instances with arbitrary weights.

4 A Greedy Restriction Algorithm for $\mathrm{SYM}_w \circ \mathrm{AND}_k$

For a term t, we denote by $|t|$ the width of t, i.e., the number of literals in t and by $\mathrm{var}(t)$ the set of variables that appear in t (possibly negated). Let $C \in \mathbf{SYM}_w \circ \mathbf{AND}_k(n,m)$ be a circuit $\{g, w_0, (t_1, w_1), \ldots, (t_s, w_s)\}$. We define $\mathrm{var}_\ell(C) := \cup_{i:|t_i|\ge \ell}\mathrm{var}(t_i)$, $\mathrm{freq}_\ell(C, x) := |\{t_i \in C \mid x \in \mathrm{var}(t_i), |t_i| \ge \ell\}|$, and $L_\ell(C) := \sum_{i:|t_i|\ge\ell} |t_i|$.

Our second satisfiability algorithm for $\mathbf{SYM}_w \circ \mathbf{AND}_k(n,m)$ is described in Fig. 2. The basic idea is as follows:

Step 1: Choose a positive integer ℓ according to the input. We seek for a variable, say x, that occurs most frequently in terms of width at least ℓ. We recursively run the algorithm for $C|_{x=0}$ and $C|_{x=1}$. Here $C|_{x=a}$ denotes the circuit obtained from C by applying a restriction ρ such that $\rho(x) = a \in \{0,1\}$ and $\rho(x') = *$ for $x' \ne x$.

Step 2: If there is no term of width at least ℓ, we call **Algorithm1**.

We will show the following theorem which implies Theorem 1 by setting $k = n$.

▶ **Theorem 9.** *We can count the number of satisfying assignments for* $C \in \mathbf{SYM}_w \circ \mathbf{AND}_k(n,m)$ *deterministically in time*

$$\mathrm{poly}(n, m, \log w) \cdot 2^{n-\Omega\left((n/\log(mw))^{\log n/4\log(km)}\right)}$$

and exponential space.

Proof. Let us define a sequence of random variables $\{C_i\}$ inductively as $C_0 := C$ and $C_{i+1} := C_i|_{x=a}$, where $x = \arg\max_{x\in\mathrm{var}(C_i)} \mathrm{freq}_\ell(C_i, x)$ and a is a uniform random bit.

We can think of the computation of **Algorithm2** as a rooted binary tree. That is, the root node is labeled with C_0, the left and right children of the root are labeled with $C_0|_{x=0}$ and $C_0|_{x=1}$, and so on. Then, if we pick a node of depth $n - n'$ uniformly at random, the distribution of its label is identical to that of the random variable $C_{n-n'}$.

We would like to bound the running time of **Algorithm2**$(C_{n-n'}, n', n', \ell)$. It is obviously bounded from above by $\mathrm{poly}(n, m, \log w) \cdot 2^{n'}$. Furthermore, if $L_\ell(C_{n-n'}) < \frac{n'}{2}$ holds, the

```
Algorithm2($C = \{g, w_0, (t_1, w_1), \ldots, (t_s, w_s)\}$: instance, $n, n', \ell$: integer)
01: if $n > n'$,
02:     $x = \arg\max_{x \in \mathrm{var}(C)} \mathrm{freq}_\ell(C, x)$.
03:     $N_0 \leftarrow$ Algorithm2($C|_{x=0}, n-1, n', \ell$).
04:     $N_1 \leftarrow$ Algorithm2($C|_{x=1}, n-1, n', \ell$).
05:     return $N_0 + N_1$.
06: else
07:     $N \leftarrow 0$.
08:     for each $\rho : \mathrm{var}(C) \to \{0, 1, *\}$ such that $\rho^{-1}(\{0,1\}) = \mathrm{var}_\ell(C)$,
09:       $w' \leftarrow$ the maximum weight of $C|_\rho$.
10:       $N \leftarrow N+$ Algorithm1($C|_\rho, n - |\mathrm{var}_\ell(C)|, m', \ell - 1, w'$).
11:     return $N$.
```

Figure 2 A Greedy Restriction Algorithm for $\mathbf{SYM}_w \circ \mathbf{AND}_k$.

running time can be bounded by $2^{n'/2}\times$ (the running time of **Algorithm1**$(C', n'/2, m', \ell - 1, w')$) for $C' \in \mathbf{SYM}_{w'} \circ \mathbf{AND}_{\ell-1}(n'/2, m')$ with $m' = \ell \cdot (n')^{\ell-1}$ and $w' = (m+1)w$. We need the following lemma.

▶ **Lemma 10** (Greedy bottom fan-in reduction). *Let $C \in \mathbf{SYM}_w \circ \mathbf{AND}_k(n, m)$. For all $n' \geq 4$, we have*

$$\Pr\left[L_\ell(C_{n-n'}) \geq 2^\ell \cdot L_\ell(C) \cdot \left(\frac{n'}{n}\right)^{\frac{\ell+2}{2}}\right] < 2^{-n'}.$$

Since $L_\ell(C) \leq km$, if we set $n' = \frac{1}{16}\left(\frac{n}{km}\right)^{2/\ell} \cdot n$ in the above lemma, we have

$$2^\ell \cdot L_\ell(C) \cdot \left(\frac{n'}{n}\right)^{\frac{\ell+2}{2}} \leq \frac{n'}{2},$$

that is, we have $L_\ell(C_{n-n'}) < n'/2$ with probability at least $1 - 2^{-n'}$. If we set $\ell = \frac{4\log(km)}{\log n}$, then the total running time of **Algorithm2** is bounded from above by the sum of

$$\mathrm{poly}(n, m, \log w) \cdot 2^{n-n'} \cdot 2^{-n'} \cdot 2^{n'}$$

and

$$\mathrm{poly}(n, m, \log w) \cdot 2^{n-n'} \cdot (1 - 2^{-n'}) \cdot 2^{n'/2} \cdot 2^{n'/2 - \Omega((n'/(\log(m'w')))^{1/\ell})}$$

according to whether $L_\ell(C_{n-n'}) \geq n'/2$ holds or not. An elementary calculation completes the proof. ◀

▶ Remark. The novelty of our algorithm and its analysis is a new way of reducing the bottom fan-in of circuits in a greedy manner. Intuitively, given a $\mathbf{SYM}_w \circ \mathbf{AND}_k$ circuit with m gates, greedy restriction produces a collection of $\mathbf{SYM}_{w'} \circ \mathbf{AND}_{k'}$ circuits with $k' = O(\log(km)/\log n)$ such that at least one of the circuits in the collection is satisfiable if and only if so is the original circuit. Note that previous techniques such as Schuler's width reduction [53, 11] or the standard random restriction achieve $k' = O(\log(m/n))$ and this bound is not sufficient for our purpose.

5 Average-Case Circuit Lower Bounds

Through the analysis of our satisfiability algorithms, we obtain the following average-case lower bounds.

▶ **Theorem 11** (depth 2, weighted symmetric gate at the top, AND gates at the bottom)**.** *There exists a constant $\alpha > 0$ such that for every m, w and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w}$ such that for every $C \in \mathbf{SYM}_w \circ \mathbf{AND}(n, m)$, it holds that*

$$\Pr_{x \in \{0,1\}^n}[f_{n,m,w}(x) = C(x)] \leq \frac{1}{2} + 2^{-\Omega\left((n/\log(mw))^{\alpha \log n/\log(nm)}\right)}.$$

▶ **Theorem 12** (depth d, weighted symmetric gate only at the top)**.** *There exists a constant $\alpha > 0$ such that for every m, w, d and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w,d}$ such that for every $C \in \mathbf{SYM}_w \circ \mathbf{AC}^0_d(n, m)$, it holds that*

$$\Pr_{x \in \{0,1\}^n}[f_{n,m,w,d}(x) = C(x)] \leq \frac{1}{2} + 2^{-\Omega\left((n/2^{2d(\log m)^{4/5}}\log(mw))^{\alpha \log n/\log m}\right)}.$$

▶ **Theorem 13** (depth d, $t(n)$ weighted symmetric gates)**.** *There exists a constant $\alpha > 0$ such that for every m, w and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w,d,t}$ such that for every $C \in \mathbf{AC}^0_d[\mathbf{SYM}_w](n, m, t)$, it holds that*

$$\Pr_{x \in \{0,1\}^n}[f_{n,m,w,d,t}(x) = C(x)] \leq \frac{1}{2} + 2^{-\Omega\left((n/2^{2d(\log m')^{4/5}}\log(m'w'))^{\alpha \log n/\log m'}\right)},$$

where $m' = m2^{t+1}$ and $w' = (mw)^{2^{t+1}}$.

In the rest of this section, we give a proof of Theorem 11. The proof of Theorem 12 is similar and we omit proof. Theorem 13 immediately follows from Theorem 12 with the idea of the proof of Theorem 5.1 in [8].

5.1 Generalized Andreev function

In this section, we review the construction of average-case hard Boolean functions due to [17, 36]. We begin with some definitions.

▶ **Definition 14** (Statistical distance)**.** Two distributions X, Y over a set E are *ε-close* if $|\mathbf{Pr}[X \in A] - \mathbf{Pr}[Y \in A]| \leq \varepsilon$ holds for every $A \subseteq E$.

▶ **Definition 15.** A set $A \subseteq \{0,1\}^n$ is a *subcube of dimension k* if there exist $1 \leq i_1 < \cdots < i_k \leq n$ and $a_{i_1}, \ldots, a_{i_k} \in \{0,1\}$ such that $A = \{x \in \{0,1\}^n \mid x_{i_1} = a_{i_1}, \ldots, x_{i_k} = a_{i_k}\}$.

▶ **Definition 16** (Bit-fixing extractor)**.** A function $f : \{0,1\}^n \to \{0,1\}^m$ is an (n, k, m, ε)-*bit-fixing extractor* if $f(X)$ and the uniform distribution over $\{0,1\}^m$ are ε-close for every distribution X that is uniform over a subcube of $\{0,1\}^n$ of dimension at least k.

We need the following explicit construction due to Rao.

▶ **Lemma 17** (Efficient bit-fixing extractor [45])**.** *There exist constants $\alpha, \beta > 0$ such that for every $k \geq (\log n)^\alpha$, there exists a polynomial time computable $\mathrm{Ext}_{n,k} : \{0,1\}^n \to \{0,1\}^m$ that is an (n, k, m, ε)-bit-fixing extractor with $m = 0.9k$ and $\varepsilon \leq 2^{-k^\beta}$.*

We also need an efficient and explicit construction of list decodable codes.

▶ **Definition 18** (List-Decodable Code). A function $f : \{0,1\}^k \to \{0,1\}^n$ is (p, L)-list-decodable if $|\{y \in \{0,1\}^k \mid \Delta(f(x), f(y)) \le pn\}| \le L$ holds for every $x \in \{0,1\}^k$, where $\Delta(a,b)$ denotes the Hamming distance between a and b.

▶ **Lemma 19** (Efficient List-Decodable Code (Folklore), see Theorem 6.4 in [17]). *There exists a function* $\mathrm{Enc}_{n,r} : \{0,1\}^{4n} \to \{0,1\}^{2^r}$ *that is* (p, L)*-list-decodable with* $p = 1/2 - O(2^{-r/4})$ *and* $L = O(2^{r/2})$*. Furthermore, there exists an algorithm that, given* $x \in \{0,1\}^{4n}$ *and* $z \in \{0,1\}^{2^r}$*, computes* $(\mathrm{Enc}_{n,r}(x))_z$ *in polynomial time.*

We are ready to define the average-case hard Boolean functions: The generalized Andreev function $A_{n,k} : \{0,1\}^{4n} \times \{0,1\}^n \to \{0,1\}$ is defined as $A_{n,k}(x,y) := (\mathrm{Enc}_{n,0.9k}(x))_{\mathrm{Ext}_{n,k}(y)}$. Let $K(x)$ denote the *Kolmogorov complexity* of a string $x \in \{0,1\}^*$. The following lemma plays an important role in the proofs of our average-case lower bounds.

▶ **Lemma 20** (Theorem 6.5 in [17]). *There exist constants* $\alpha, \gamma > 0$ *such that the following holds. Let* $k \ge (\log n)^\alpha$ *and* C *be a* k*-variate circuit whose binary description length is at most* n *in a some fixed encoding scheme. Let* $\rho : \{x_1, \dots, x_n\} \to \{0, 1, *\}$ *be a restriction with* $|\rho^{-1}(*)| = k$*. Fix* $a \in \{0,1\}^{4n}$ *with* $K(a) \ge 3n$ *and define* $f(y) := A_{n,k}(a, y)$*. Then, we have*

$$\Pr_{y' \in \{0,1\}^k}[C(y') = f|_\rho(y')] \le \frac{1}{2} + \frac{1}{2^{k^\gamma}}.$$

The following fact can be shown by a counting argument.

▶ **Fact 21.** *For every* $0 < p < 1$, $\mathbf{Pr}_{x \in \{0,1\}^n}[K(x) \le (1-p)n] \le 2^{-pn+1}$.

5.2 Proof of Theorem 11

Fix n, m, w and let $n' = (n/\log(mw))^{\log n / 4 \log(nm)}$. Select any $a \in \{0,1\}^{4n}$ with $K(a) \ge 3n$ and let $f(y) := A_{n,n'}(a, y)$. We show the following lemma.

▶ **Lemma 22.** *For every* $C \in \mathbf{SYM}_w \circ \mathbf{AND}(n,m)$*, it holds that*

$$\Pr_{y \in \{0,1\}^n}[C(y) = f(y)] \le \frac{1}{2} + 2^{-\Omega(n'^\gamma)},$$

where $\gamma > 0$ *is a universal constant from Lemma 20.*

Assuming this, the proof of Theorem 11 is complete since by Fact 21, we have

$$\begin{aligned}
\Pr_{x,y}[A_{n,n'}(x,y) = C(x,y)] &\le \Pr_x[K(x) < 3n] + \Pr_x[K(x) \ge 3n] \\
&\times \Pr_{x,y}[A_{n,n'}(x,y) = C(x,y) \mid K(x) \ge 3n] \\
&\le 2^{-\Omega(n)} + \max_{x : K(x) \ge 3n} \Pr_y[A_{n,n'}(x,y) = C(x,y)] \\
&\le 2^{-\Omega(n)} + \frac{1}{2} + 2^{-\Omega(n'^\gamma)}.
\end{aligned}$$

.

Proof of Lemma 22. We can see that from the proofs of Theorems 6 and 9, C can be computed by a restriction decision tree T of height $n - n'$ such that (1) each leaf is labeled by a circuit in $\mathbf{SYM}_{w'} \circ \mathbf{AND}_{k'}(n', m')$ for some m', k', w' and (2) except for a $2^{-n^{\Omega(1)}}$ fraction of leaves, such a circuit can be described by using at most n bits (due to Fact 7). Let $\sigma(C)$

denote the description length of a circuit C in a fixed encoding scheme. Let ρ be a random restriction sampled by selecting a leaf of T uniformly at random and y_ρ be a uniform random element of $\{0,1\}^{\rho^{-1}(*)}$. Then, we have

$$\begin{aligned}\mathop{\mathbf{Pr}}_{y}[C(y)=f(y)] &\leq \mathop{\mathbf{Pr}}_{\rho}[\sigma(C|_\rho)>n]+\mathop{\mathbf{Pr}}_{\rho}[\sigma(C|_\rho)\leq n] \\ &\times \mathop{\mathbf{Pr}}_{\rho,y_\rho}[C|_\rho(y_\rho)=f|_\rho(y_\rho)\mid \sigma(C|_\rho)\leq n]\leq 2^{-n^{\Omega(1)}}+\frac{1}{2}+2^{-\Omega(n'^{\gamma})},\end{aligned}$$

where the last inequality is by Item (2) above and Lemma 20. This completes the proof. ◀

6 Worst-Case Lower Bounds

From the average-case lower bounds in Section 5, we obtain the following worst-case lower bounds.

▶ **Theorem 23** (majority vote of depth 2, weighted symmetric gate at the top, AND gates at the bottom). *There exists a constant $\alpha>0$ such that for every m,w and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w}$ such that $C\in\mathbf{MAJ}\circ\mathbf{SYM}_w\circ\mathbf{AND}(n,m)$ cannot compute $f_{n,m,w}$ if the majority gate at the top of C has fan-in at most $2^{o\left((n/\log(mw))^{\alpha\log n/\log(nm)}\right)}$.*

▶ **Theorem 24** (majority vote of depth d, weighted symmetric gate only at the top). *There exists a constant $\alpha>0$ such that for every m,w,d and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w,d}$ such that any $C\in\mathbf{MAJ}\circ\mathbf{SYM}_w\circ\mathbf{AC}^0_d(n,m)$ cannot compute $f_{n,m,w,d}$ if the majority gate at the top of C has fan-in at most $2^{o\left((n/2^{2d(\log m)^{4/5}}\log(mw))^{\alpha\log n/\log m}\right)}$.*

▶ **Theorem 25** (majority vote of depth d, $t(n)$ weighted symmetric gates). *There exists a constant $\alpha>0$ such that for every m,w,d,t and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w,d,t}$ such that any $C\in\mathbf{MAJ}\circ\mathbf{AC}^0_d[\mathbf{SYM}_w](n,m,t)$ cannot compute $f_{n,m,w,d,t}$ if the majority gate at the top of C has fan-in at most $2^{o\left((n/2^{2d(\log m')^{4/5}}\log(m'w'))^{\alpha\log n/\log m'}\right)}$, where $m'=m2^{t+1}$ and $w'=(mw)^{2^{t+1}}$.*

▶ **Theorem 26** (depth d, $t(n)$ weighted symmetric gates). *There exists a constant $\alpha>0$ such that for every m,w,d,t and sufficiently large n, there exists a polynomial time computable function $f_{n,m,w,d,t}$ such that any $C\in\mathbf{AC}^0_d[\mathbf{SYM}_w](n,m,t)$ cannot compute $f_{n,m,w,d,t}$ if*

$$t=o\left((n/2^{2d(\log m')^{4/5}}\log(m'w'))^{\alpha\log n/\log m'}\right)$$

holds, where $m'=m(t+1)$ and $w'=m^tw^{t+1}$.

We need a corollary of the discriminator lemma.

▶ **Lemma 27** (Discriminator Lemma [27]). *If a circuit $C\in\mathbf{MAJ}\circ\mathcal{C}$ is a majority vote of k circuits $C_1,\ldots,C_k\in\mathcal{C}$, then for some $1\leq i\leq k$, we have*

$$|\Pr_x[C_i(x)=1\mid C(x)=1]-\Pr_x[C_i(x)=1\mid C(x)=0]|\geq\frac{1}{k}.$$

For $f,g:\{0,1\}^n\to\{0,1\}$, let $\mathrm{Corr}(f,g):=|\mathbf{Pr}_x[f(x)=g(x)]-\mathbf{Pr}_x[f(x)\neq g(x)]|$.

▶ **Corollary 28.** *For $\varepsilon \geq 0$, if C in Lemma 27 also satisfies that*

$$|\Pr_x[C(x)=0] - \Pr_x[C(x)=1]| = 2\varepsilon,$$

then we have $\mathrm{Corr}(f,g) \geq \frac{1}{k} - 2\varepsilon$.

Theorems 23, 24 and 25 immediately follow from Theorems 11, 12 and 13 with Corollary 28. Theorem 26 can be shown by combining the relation of circuit classes, Theorem 12 and Corollary 28.

References

1 Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 218–230, 2015.

2 Kazuyuki Amano and Atsushi Saito. A nonuniform circuit class with multilayer of threshold gates having super quasi polynomial size lower bounds against NEXP. In *Proceedings of the 9th International Conference on Language and Automata Theory and Applications (LATA)*, pages 461–472, 2015.

3 Kazuyuki Amano and Atsushi Saito. A satisfiability algorithm for some class of dense depth two threshold circuits. *IEICE Transactions*, 98-D(1):108–118, 2015.

4 James Aspnes, Richard Beigel, Merrick L. Furst, and Steven Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994.

5 László Babai, Noam Nisan, and Mario Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *J. Comput. Syst. Sci.*, 45(2):204–232, 1992.

6 David A. Mix Barrington and Howard Straubing. Complex polynomials and circuit lower bounds for modular counting. *Computational Complexity*, 4:325–338, 1994.

7 Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating AC^0 by small height decision trees and a deterministic algorithm for $\#\mathrm{AC}^0$ SAT. In *Proceedings of the 27th Conference on Computational Complexity (CCC)*, pages 117–125, 2012.

8 Richard Beigel. When do extra majority gates help? polylog(n) majority gates are equivalent to one. *Computational Complexity*, 4:314–324, 1994.

9 Richard Beigel, Nick Reingold, and Daniel A. Spielman. PP is closed under intersection. *J. Comput. Syst. Sci.*, 50(2):191–202, 1995.

10 Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 163–173, 2014.

11 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC)*, pages 252–260, 2006.

12 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Revised Selected Papers from the 4th International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 75–85, 2009.

13 Marco L. Carmosino, Russell Impagliazzo, Valentine Kabanets, and Antonina Kolokolova. Algorithms from natural lower bounds. In *Proceedings of the 31st Conference on Computational Complexit (CCC)*, pages 10:1–10:24, 2016.

14 Arkadev Chattopadhyay and Kristoffer Arnsfelt Hansen. Lower bounds for circuits with few modular and symmetric gates. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 994–1005, 2005.

15 Ruiwen Chen. Satisfiability algorithms and lower bounds for Boolean formulas over finite bases. In *Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II*, pages 223–234, 2015.

16 Ruiwen Chen and Valentine Kabanets. Correlation bounds and #SAT algorithms for small linear-size circuits. In *Proceedings of the 21st International Conference on Computing and Combinatorics (COCOON)*, pages 211–222, 2015.

17 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015.

18 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic #SAT algorithm for small De Morgan formulas. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS), Part II*, pages 165–176, 2014.

19 Ruiwen Chen and Rahul Santhanam. Improved algorithms for sparse MAX-SAT and MAX-k-CSP. In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pages 33–45, 2015.

20 Ruiwen Chen, Rahul Santhanam, and Srikanth Srinivasan. Average-case lower bounds and satisfiability algorithms for small threshold circuits. In *Proceedings of the 31st Conference on Computational Complexit (CCC)*, pages 1:1–1:35, 2016.

21 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. In *Proceedings of the 27th Annual IEEE Conference on Computational Complexity (CCC)*, pages 74–84, 2012.

22 Jürgen Forster, Matthias Krause, Satyanarayana V. Lokam, Rustam Mubarakzjanov, Niels Schmitt, and Hans-Ulrich Simon. Relations between communication complexity, linear arrangements, and computational complexity. In *Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 171–182, 2001.

23 Mikael Goldmann. On the power of a threshold gate at the top. *Inf. Process. Lett.*, 63(6):287–293, 1997.

24 Mikael Goldmann and Marek Karpinski. Simulating threshold circuits by majority circuits. *SIAM J. Comput.*, 27(1):230–246, 1998.

25 Alexander Golovnev, Alexander S. Kulikov, Alexander Smal, and Suguru Tamaki. Circuit size lower bounds and #SAT upper bounds through a general framework. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2016, to appear.

26 Parikshit Gopalan and Rocco A. Servedio. Learning and lower bounds for AC^0 with threshold gates. In *Proceedings of the 13th APPROX and the 14th RANDOM*, pages 588–601, 2010.

27 András Hajnal, Wolfgang Maass, Pavel Pudlák, Mario Szegedy, and György Turán. Threshold circuits of bounded depth. *J. Comput. Syst. Sci.*, 46(2):129–154, 1993.

28 Kristoffer Arnsfelt Hansen and Peter Bro Miltersen. Some meet-in-the-middle circuit lower bounds. In *Proceedings of the 29th International Symposium Mathematical Foundations of Computer Science (MFCS)*, pages 334–345, 2004.

29 Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1:113–129, 1991.

30 Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *Electronic Colloquium on Computational Complexity (ECCC)*, TR14-24, 2014.

31 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for AC^0. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 961–972, 2012.

32 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

33 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 479–488, 2013.

34 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

35 Hamid Jahanjou, Eric Miles, and Emanuele Viola. Local reductions. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, pages 749–760, 2015.

36 Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for demorgan formula size. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 588–597, 2013.

37 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

38 Shachar Lovett and Srikanth Srinivasan. Correlation bounds for poly-size AC^0 circuits with $n^{1-o(1)}$ symmetric gates. In *Proceedings of the 14th APPROX 2011 and the 15th RANDOM*, pages 640–651, 2011.

39 Oleg Borisovich Lupanov. On a method of circuit synthesis (in Russian). *Izvestiâ vysših učebnyh zavedenij, Radiofiz*, 1:120–140, 1958.

40 Dániel Marx. Consequences of SETH: Tight bounds for some more problems, 2015. (abstract, slides and archived video). URL: `https://simons.berkeley.edu/talks/daniel-marx-2015-09-04`.

41 Saburo Muroga, Iwao Toda, and Satoru Takasu. Theory of majority decision elements. *Journal of the Franklin Institute*, 271(5):376–418, 1961.

42 Atsuki Nagao, Kazuhisa Seto, and Junichi Teruyama. A moderately exponential time algorithm for k-IBDD satisfiability. In *Proceedings of the 14th International Symposium, on Algorithms and Data Structures (WADS)*, pages 554–565, 2015.

43 Igor Carboni Oliveira. Algorithms versus circuit lower bounds. *Electronic Colloquium on Computational Complexity (ECCC)*, TR13-117, 2013.

44 Vladimir V. Podolskii. Exponential lower bound for bounded depth circuits with few threshold gates. *Inf. Process. Lett.*, 112(7):267–271, 2012.

45 Anup Rao. Extractors for low-weight affine sources. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 95–101, 2009.

46 Alexander Razborov. Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. *Mathematical Notes of the Academy of Sci. of the USSR*, 41(4):333–338, 1987.

47 Alexander Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

48 Alexander Razborov and Avi Wigderson. $n^{\Omega(\log n)}$ lower bounds on the size of depth-3 threshold circuits with AND gates at the bottom. *Inf. Process. Lett.*, 45(6):303–307, 1993.

49 Takayuki Sakai, Kazuhisa Seto, and Suguru Tamaki. Solving sparse instances of Max SAT via width reduction and greedy restriction. *Theory Comput. Syst.*, 57(2):426–443, 2015.

50 Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. A satisfiability algorithm for depth-2 circuits with a symmetric gate at the top and AND gates at the bottom. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-136, 2015.

51 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 183–192, 2010.

52 Rahul Santhanam. Ironic complicity: Satisfiability algorithms and circuit lower bounds. *Bulletin of the EATCS*, 106:31–52, 2012.

53 Rainer Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms*, 54(1):40–44, 2005.

54 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013.

55 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 77–82, 1987.

56 Srikanth Srinivasan. A compression algorithm for $\mathrm{AC}^0[\oplus]$ circuits using certifying polynomials. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-142, 2015.

57 Avishay Tal. #SAT algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, TR15-114, 2015.

58 Emanuele Viola. Pseudorandom bits for constant-depth circuits with few arbitrary symmetric gates. *SIAM J. Comput.*, 36(5):1387–1403, 2007.

59 Fengming Wang. NEXP does not have non-uniform quasipolynomial-size ACC circuits of $o(\log\log n)$ depth. In *Proceedings of the 8th Annual Conference on Theory and Applications of Models of Computation (TAMC)*, pages 164–170, 2011.

60 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.

61 Ryan Williams. Natural proofs versus derandomization. In *Proceedings of the 45th ACM Symposium on Theory of Computing Conference (STOC)*, pages 21–30, 2013.

62 Ryan Williams. Algorithms for circuits and circuits for algorithms. In *Proceedings of the 29th Annual IEEE Conference on Computational Complexity (CCC)*, pages 248–261, 2014.

63 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Proceedings of the 46th Symposium on Theory of Computing (STOC)*, pages 194–202, 2014.

64 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2, 2014.

65 Francis Zane. *Circuits, CNFs, and satisfiability.* PhD thesis, UC San Diego, 1998.

Transducer-Based Rewriting Games for Active XML

Martin Schuster

TU Dortmund, Dortmund, Germany
martin.schuster@tu-dortmund.de

Abstract

Context-free games are two-player rewriting games that are played on nested strings representing XML documents with embedded function symbols. These games were introduced to model rewriting processes for intensional documents in the *Active XML* framework, where input documents are to be rewritten into a given target schema by calls to external services.

This paper studies the setting where dependencies between inputs and outputs of service calls are modelled by transducers, which has not been examined previously. It defines transducer models operating on nested words and studies their properties, as well as the computational complexity of the winning problem for transducer-based context-free games in several scenarios. While the complexity of this problem is quite high in most settings (ranging from NP-complete to undecidable), some tractable restrictions are also identified.

1998 ACM Subject Classification F.2.m Miscellaneous, F.4.2 Grammars and Other Rewriting Systems, H.3.5 Online Information Services

Keywords and phrases Active XML, Computational Complexity, Nested Words, Transducers, Rewriting Games

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.83

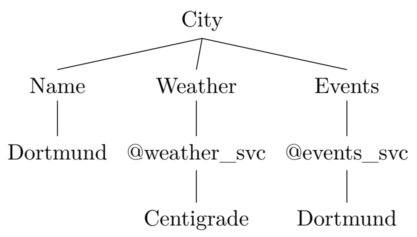
1 Introduction

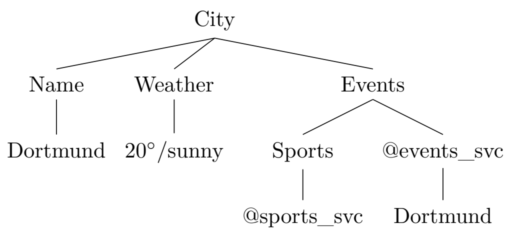
Scientific context

Context-free games on strings are two-player games extending context-free grammars, with the first player (called JULIET) choosing the non-terminal to be replaced and the second player (called ROMEO) choosing a replacement for that non-terminal. The winning condition for JULIET is reaching, at some point during the game, some string in a given target language over the combined alphabet of non-terminals and terminals.

These games were first introduced in [7] to model the rewriting process of *Active XML (AXML)* [1] documents. The intention of AXML is modelling *intensional* documents, i.e. documents that do not store all required information explicitly but instead contain references to external services, from which current information may be materialised on demand, as illustrated in the example below. To this end, AXML extends standard XML with *function nodes* referring to external web services that may be called to insert data into the AXML document when the document is requested. Context-free games abstract from AXML to model the uncertainty inherent in using external data.

A standard example (cf. [6, 7]) of an application for AXML is depicted in Figure 1. In this example, we consider (part of) an AXML document retained by a local online news site providing information about current weather and events. Initially, the server-side document looks like the one in Figure 1a. The nodes labelled @weather_svc and @events_svc are function nodes referring to external weather and event services.





(a) Example document before rewriting. **(b)** Same document after function calls.

Figure 1 Example of Active XML rewriting. After calls to function nodes @weather_svc and @events_svc in Fig. 1a, external data is materialised to yield the document in Fig. 1b.

Figure 1b shows the document from Figure 1a after both function nodes have been called, replacing them by the external services' results. As exemplified by the function node labelled @weather_svc, call results replace the entire subtree rooted at the called function node, with that subtree being passed to the external service as a parameter. Concretely, the @weather_svc node's child tells the weather service that temperatures returned should be in centigrade. As the call result of the @event_svc node shows, returns of external services may contain further function nodes, even copies of the called function node.

The *safe rewriting problem* [6] of determining whether a given AXML document can always be rewritten into a target schema was abstracted in [7] into the problem of determining whether JULIET has a winning strategy in a given context-free game on strings, with JULIET representing a rewriting algorithm and ROMEO representing the uncertainty inherent in function calls. This research assumed DTDs as schema formalisms. Allowing more expressive schema languages such as XML Schema [10] then led to research into context-free games on *nested strings* (i.e. XML-like linearisations of trees) [3]. Even though none of these previous works modelled dependencies between parameters and outputs of function calls, they already showed that the winning problem for JULIET can be undecidable or of a very high complexity, unless strategies for JULIET and allowed schema languages are seriously restricted.

The impact of service call parameters has so far only been studied in a limited fashion. In [10] (and in [6], for AXML rewriting with DTDs), external services were modelled by *input* (or *validation*) and *output* (or *replacement*) schemas, the semantics being that a function node could only be called if its parameter subtree was valid with regard to its corresponding input schema, which would then yield a return conforming to its output schema. This is a purely syntactic handling of input parameters which models a rather simple relationship between input and output of function calls; for instance, an @event_svc call reproducing its input parameters in its output as shown in Figure 1 cannot be enforced in this model.

While dependencies between parameters and outputs of service calls have always been implicit in the AXML model, they have not been studied in detail so far. Therefore, we extend the context-free games on nested words from [10] by (generally non-deterministic) transformations relating function parameters to possible outputs. We define *nested word transducers* (NWT) as a comparatively simple finite representation for transformations on nested words that naturally extends the nested word automata used in [10]. We then study the complexity of the winning problem for JULIET in various restrictions of context-free games with transducer-based replacement. As auxiliary results of potentially independent interest, we also examine closure properties and basic algorithmic problems of NWT.

Contributions

In light of prior undecidability and complexity results, the main objective of this paper is finding suitable restrictions to transducer-based context-free games that render the winning problem for Juliet decidable with as low complexity as possible. The two basic types of restrictions we examine are strategy restrictions (i.e. restrictions to Juliet's capabilities of calling function symbols) and restrictions to the type of NWT used for rewriting. To avoid undecidability, we only allow left-to-right strategies, i.e. once a function node has been called, no function calls to nodes preceding it (in post-order) are possible (cf. [7]).

The most important class of strategy restrictions considered here are restrictions to games with limited *replay*. In general context-free games, after a function call, Juliet continues her rewriting on that function call's result; we call this the *unbounded replay* case, as Juliet may continue rewriting function call results for as long as new function nodes are returned. In the *replay-free* case, Juliet is instead forbidden to call any function nodes inside results of function calls. As an intermediate case between unbounded replay and replay-free games, we also consider *bounded replay* games, where Juliet may only call functions returned by function calls up to a fixed maximum recursion depth. For instance, in a replay-free game, Juliet could call neither of the function nodes labelled @sports_svc and @events_svc in the situation of Figure 1b; in a bounded replay game of depth 2, on the other hand, she could call these nodes, but not any function nodes returned by those secondary calls.

The second type of restrictions comes from limiting expressiveness of the transducer used in games. Generally, transducers are allowed to be *non-functional*, i.e. any input string, may have several transducts (to model the fact that function call results are dependent upon, but not uniquely determined by, input parameters). The main types of transducers examined here are the following:

- *Nested word transducers (NWT)* allow for transforming input strings into output strings that are arbitrarily long, regardless of the input string's size.
- *Nested word transducers without ϵ-transitions (ϵ-free NWT)* may only increase the size of an input string by no more than a linear factor.
- *Relabelling transducers* may only change labels of input strings, not their structure.
- As a special case, *functional relabelling transducers* are relabelling transducers whose output string is uniquely determined by their input.

This paper's main complexity results are summarised in Table 1. The central insight here is that the least restricted settings yield an undecidable or non-elementary winning problem, and even for strong restrictions, the complexity of the winning problem is generally quite high, with no tractable case among the standard settings. For this reason, we also study several limitations of these settings, derived from our lower bound proofs, in order to reduce complexity:

- *Depth-bounded NWT* lower the complexity of the replay-free case to EXPSPACE-complete (in comparison to 2-EXPTIME for general NWT).
- Strategies with *bounded Call width* lower the complexity of the bounded-replay case for ϵ-free NWT from non-elementary to co-NEXPTIME-complete or co-NP-complete, depending on the precise formalisation of bounded Call width.
- *Write-once* strategies yield a tractable case for functional relabelling transducers in a setting even more restrictive than the replay-free one.

Table 1 Summary of complexity results. All results are completeness results.

	No replay	Bounded	Unbounded
NWT	2-EXPTIME	undecidable	undecidable
ϵ-free NWT	co-NEXPTIME	non-elementary	undecidable
Relabelling	PSPACE	PSPACE	EXPTIME
Functional relabelling	NP	NP	PSPACE

Related Work

Beyond the work already discussed, further complexity and decidability results for context-free games can be found in [2, 4], and [7] contains further references to related work.

Our concept of nested word transducers is based on Visibly Pushdown Transducers [8, 12], specifically the well-nested VPT of [5]. The original definitions of VPT in [8, 12] included ϵ-transitions, which were dropped from later definitions, as they caused several algorithmic problems, such as functionality and equivalence, to become undecidable (cf. [11]). Different from these approaches, this paper combines ϵ-transitions with the restriction to well-nested words, which is (to the best of the author's knowledge) new research.

Organisation

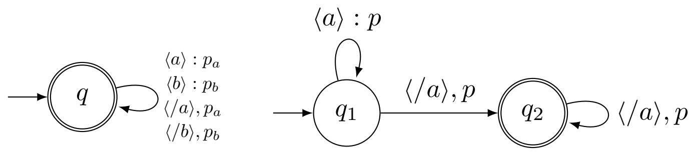
Section 2 gives basic notation and definitions. Section 3 defines nested word transducers and examines their structural and algorithmic properties. The next three sections give results on the complexity of the winning problem for games with transducer-based replacement, from most to least expressive – general nested word transducers (Section 4), nested word transducers without ϵ-transitions (Section 5), and relabelling transducers (Section 6). Each of these sections also discusses one of the restrictions with reduced complexity mentioned above. Section 7 concludes the paper. Due to space limitations, proofs and technical definitions are omitted here; for more details, an extended version is available online [9]. The author is grateful to Gaetano Geck and Thomas Schwentick for careful proof-reading and valuable suggestions, and to the anonymous reviewers for their insightful and constructive comments.

2 Preliminaries

For any natural number $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \dots, n\}$. For finite sets M, $\mathcal{P}(M)$ denotes the powerset of M, i.e. the set of all subsets of M. For an alphabet Σ, we denote the set of finite strings over Σ by Σ^* and ϵ denotes the empty string.

Nested words

For a finite alphabet Σ, $\langle\Sigma\rangle \overset{\text{def}}{=} \{\langle a\rangle \mid a \in \Sigma\}$ denotes the set of all *opening* Σ*-tags* and $\langle/\Sigma\rangle \overset{\text{def}}{=} \{\langle/a\rangle \mid a \in \Sigma\}$ the set of all *closing* Σ*-tags*. We denote by $\hat{\Sigma} \overset{\text{def}}{=} \langle\Sigma\rangle \cup \langle/\Sigma\rangle$ the set of all Σ-tags. The set $\mathrm{NW}(\Sigma) \subseteq \hat{\Sigma}^*$ of *(well-)nested words* (or *(well-)nested strings*) over Σ is the smallest set such that $\epsilon \in \mathrm{NW}(\Sigma)$, and if $u, v \in \mathrm{NW}(\Sigma)$ and $a \in \Sigma$, then also $u\langle a\rangle v\langle/a\rangle \in \mathrm{NW}(\Sigma)$. We (informally) associate with every nested word w its *canonical forest representation*, such that words $\langle a\rangle\langle/a\rangle$, $\langle a\rangle v\langle/a\rangle$ and uv correspond to an a-labelled leaf, a tree with root a (and subforest corresponding to v), and the forest of u followed by the forest of v, respectively. A nested string w is *rooted* if its corresponding forest is a tree. We denote the set of rooted nested strings over Σ by $\mathrm{rNW}(\Sigma)$. In a string $w = w_1 \dots w_n \in \hat{\Sigma}^*$, two tags $w_i \in \langle\Sigma\rangle$ and $w_j \in \langle/\Sigma\rangle$ with $i < j$ are *associated* if the substring $w_i \dots w_j$ of w is a rooted nested string. An opening (closing) tag w_i in w is *unmatched*, if it has no associated



(a) NWA A_1 **(b)** NWA A_2

Figure 2 NWAs A_1 and A_2 from Example 2.1.

closing (opening) tag in w. To stress the distinction from nested strings in $\mathrm{NW}(\Sigma)$, we refer to strings in Σ^* as *flat strings*.

Nested word automata

A *nested word automaton (NWA)* $A = (Q, P, \Sigma, \delta, q_0, F)$ [3] is basically a pushdown automaton which performs a push operation on every opening tag and a pop operation on every closing tag, and in which the pushdown symbols are just states. More formally, A consists of a set Q of *linear states*, a set P of *hierarchical states*, an alphabet Σ, a *transition relation* δ, an *initial state* $q_0 \in Q$, and a set $F \subseteq Q$ of *accepting (linear) states*. The relation δ is a subset of the union of sets $(Q \times \langle\Sigma\rangle \times Q \times P)$ and $(Q \times P \times \langle/\Sigma\rangle \times Q)$. We sometimes interpret δ as the union of two functions from $(Q \times \langle\Sigma\rangle)$ to $\mathcal{P}(Q \times P)$ and from $(Q \times P \times \langle/\Sigma\rangle)$ to $\mathcal{P}(Q)$ and write accordingly $(q', p) \in \delta(q, \langle a\rangle)$ for $(q, \langle a\rangle, q', p) \in \delta$ and $q' \in \delta(q, p, \langle/a\rangle)$ for $(q, p, \langle/a\rangle, q') \in \delta$. The semantics of NWA as well as the language $L(A)$ decided by a NWA A are defined in the natural way, with a NWA accepting if it reaches a configuration with an accepting state and empty stack. If A is a NWA, we call $L(A)$ a *regular language* (of nested words). A NWA is *deterministic* (or DNWA) if $|\delta(q, \langle a\rangle)| = 1 = |\delta(q, p, \langle/a\rangle)|$ for all $q \in Q$, $p \in P$ and $a \in \Sigma$. In this case, we simply write $\delta(q, \langle a\rangle) = (q', p')$ instead of $\delta(q, \langle a\rangle) = \{(q', p')\}$ (and accordingly for $\delta(q, p, \langle/a\rangle)$).

▶ **Example 2.1.** The NWA A_1 (Fig. 2a) checks that its input string is well-nested by pushing hierarchical state p_a (resp. p_b) to the stack on each opening $\langle a\rangle$ (resp. $\langle b\rangle$) tag and popping an according hierarchical state with each matching closing tag. In this manner, A_1 decides the set of all well-nested strings over $\{a, b\}$. The NWA A_2 (Fig. 2b) initially pushes a hierarchical state p each time it reads $\langle a\rangle$ in linear state q_1, then changes linear state to q_2 on reading the first $\langle/a\rangle$ and accepts iff each initial $\langle a\rangle$ is matched by a $\langle/a\rangle$. In this manner, it decides the language $\{\langle a\rangle^n \langle/a\rangle^n \mid n \geq 1\}$.

Context-free games

A *context-free game (with transduction) on nested words (cfG)* $G = (\Sigma, \Gamma, R, T)$ consists of a finite alphabet Σ, a set $\Gamma \subseteq \Sigma$ of *function symbols*, a *(replacement) rule set* $R \subseteq \mathrm{rNW}(\Sigma) \times \mathrm{NW}(\Sigma)$ and a *target language* $T \subseteq \mathrm{NW}(\Sigma)$. We will only consider the case where T is a non-empty regular nested word language and replacement rules are given by nested word transducers, to be defined in Section 3. A play of G is played by two players, Juliet and Romeo, on a word $w \in \mathrm{NW}(\Sigma)$. In a nutshell, Juliet moves the focus along w from left to right and decides for each closing tag $\langle/a\rangle$, whether she plays a *Read* or, in case $a \in \Gamma$, a *Call* move. In the latter case, Romeo then replaces the rooted word u ending at the position of $\langle/a\rangle$ with some word v with $(u, v) \in R$ and the focus is set on the first symbol of v. If no such word v exists, Romeo immediately wins the play. In case of a Read move, the focus just moves further on. Juliet wins a play if the word obtained at its end is in T.

Strategies

A *strategy* for player $p \in \{\mathrm{J}, \mathrm{R}\}$ maps game states where player p is to move into allowed moves for player p, i.e. strategies σ for JULIET return moves in $\{\text{Read}, \text{Call}\}$ while strategies τ for ROMEO return replacement strings in $\mathrm{NW}(\Sigma)$. Given an initial word w and strategies σ, τ the play $\Pi(\sigma, \tau, w)$ according to σ and τ on w is uniquely determined. A *winning strategy* for JULIET is a strategy σ such that JULIET wins the play $\Pi(\sigma, \tau, w)$, for every τ of ROMEO. By $\mathrm{JWin}(G)$ we denote the set of all words for which JULIET has a winning strategy in G.

The *Call depth* of a play Π is the maximum nesting depth of Call moves in Π, if this maximum exists. That is, the Call depth of a play is zero, if no Call is played at all, and one, if no Call is played inside a string yielded by a replacement move. For a strategy σ of JULIET and a string $w \in \mathrm{NW}(\Sigma)$, the *Call depth* $\mathrm{Depth}^G(\sigma, w)$ of σ on w is the maximum Call depth in any play $\Pi(\sigma, \tau, w)$. A strategy σ has *k-bounded Call depth* if $\mathrm{Depth}^G(\sigma, w) \leq k$ for all $w \in \mathrm{NW}(\Sigma)$. As a more intuitive formulation, we use the concept of *replay*: Strategies for JULIET of Call depth one are called *replay-free*, and strategies of k-bounded Call depth, for any k, have *bounded replay.*

Algorithmic problems

In this paper, we study the following algorithmic problem $\mathrm{JWIN}(\mathcal{G})$ for various classes $\mathcal{G}$ of context-free games with replacement transducers.

$\mathrm{JWIN}(\mathcal{G})$	
Given:	A context-free game $G \in \mathcal{G}$ and a string w.
Question:	Is $w \in \mathrm{JWin}(G)$?

A class $\mathcal{G}$ of context-free games in $\mathrm{JWIN}(\mathcal{G})$ generally comes with three parameters:

- the representation of the target language T,
- the representation of the replacement relation R, and
- to which extent replay is restricted.

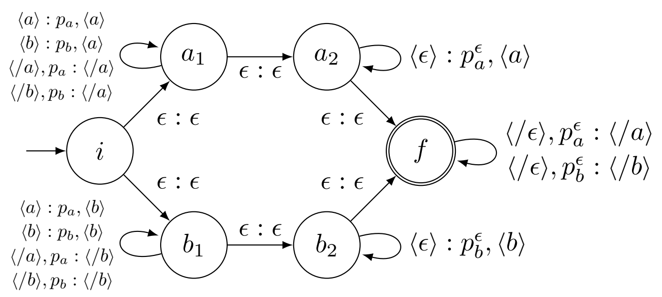
We generally assume target languages to be represented by DNWAs, because the complexity of the winning problem is already quite high under that assumption, and our main interest is in finding classes $\mathcal{G}$ for which $\mathrm{JWIN}(\mathcal{G})$ is tractable. Replacement relations will be given as different types of nested word transducers (defined in Section 3). By a slight abuse of notation, the replacement transducer implementing a replacement relation R will also be referred to as R.

In each setting, we consider the cases of unrestricted replay, bounded replay (Call depth k, for some k), and no replay (Call depth 1). We note that replay depth is formally not an actual game parameter, but the algorithmic problem can be restricted to strategies of JULIET of the stated kind. If the class $\mathcal{G}$ of games is clear from the context, we often simply write JWIN instead of $\mathrm{JWIN}(\mathcal{G})$.

3 Nested Word Transducers

In this section, we define nested word transducers and examine their closure properties and complexities of algorithmic problems. Thanks to our definition putting some rather severe restrictions on the use of ϵ-transitions and the allowed output of transducers, we obtain advantageous closure properties and comparatively low complexities.

Intuitively, a NWT T works much like a NWA with output and additional ϵ-transitions – T reads its input from left to right and decides nondeterministically which available transition



$\langle a\rangle : p_a, \langle a\rangle$
$\langle b\rangle : p_b, \langle a\rangle$
$\langle /a\rangle, p_a : \langle /a\rangle$
$\langle /b\rangle, p_b : \langle /a\rangle$
a_1
$\epsilon : \epsilon$
a_2
$\langle \epsilon\rangle : p_a^\epsilon, \langle a\rangle$
$\epsilon : \epsilon$
$\epsilon : \epsilon$
i
f
$\langle /\epsilon\rangle, p_a^\epsilon : \langle /a\rangle$
$\langle /\epsilon\rangle, p_b^\epsilon : \langle /b\rangle$
$\epsilon : \epsilon$
$\epsilon : \epsilon$
$\langle a\rangle : p_a, \langle b\rangle$
$\langle b\rangle : p_b, \langle b\rangle$
$\langle /a\rangle, p_a : \langle /b\rangle$
$\langle /b\rangle, p_b : \langle /b\rangle$
b_1
$\epsilon : \epsilon$
b_2
$\langle \epsilon\rangle : p_b^\epsilon, \langle b\rangle$

Figure 3 Nested Word Transducer T_{ab} from Example 3.2.

to use; on an opening (resp. closing) transition, it reads an opening (closing) input tag, changes its linear state and pushes (pops) a hierarchical state while producing an output. Opening (closing, internal) ϵ-transitions do not consume input symbols but induce state changes and outputs. T only produces an output string if it accepts the input string.

▶ **Definition 3.1.** A *nested word transducer* (or *NWT*) is a tuple $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ consisting of a set Q of *linear states*, a set P of *hierarchical states*, a set $P_\epsilon \subseteq P$ of *hierarchical ϵ-states*, an alphabet Σ, a *transition relation* δ, which is the union of three relations from $(Q \times (\langle\Sigma\rangle \cup \{\langle\epsilon\rangle\}) \times Q \times P \times \hat{\Sigma}^*)$ (called *opening* transitions), $(Q \times \{\epsilon\} \times Q \times \mathrm{NW}(\Sigma))$ (called *internal* transitions) and $(Q \times P \times (\langle/\Sigma\rangle \cup \{\langle/\epsilon\rangle\}) \times Q \times \hat{\Sigma}^*)$ (called *closing* transitions), an *initial state* $q_0 \in Q$, and a set of *accepting states* $F \subseteq Q$, such that for all $q, q', r, r' \in Q$, $p \in P$, $a \in \Sigma \cup \{\epsilon\}$ and $u, v \in \hat{\Sigma}^*$ it holds that[1]

- $(q, \langle\epsilon\rangle, q', p, u) \in \delta$ or $(q, p, \langle/\epsilon\rangle, q', u) \in \delta$ if and only if $p \in P_\epsilon$ (*ϵ-consistency*),
- if $(q, \langle a\rangle, q', p, u) \in \delta$ and $(r, p, \langle/a\rangle, r', v) \in \delta$, then $uv \in \mathrm{NW}(\Sigma)$ (*well-formedness*), and
- for each $(q, \langle a\rangle, q', p, u) \in \delta$ (resp. $(r, p, \langle/a\rangle, r', u) \in \delta$) with $u \neq \epsilon$, u contains at least one unmatched opening (resp. closing) tag (*synchronisation*).

As for standard NWA, we also write $(q', p, u) \in \delta(q, \langle a\rangle)$ (resp. $(q', u) \in \delta(q, p, \langle/a\rangle)$, $(q', u) \in \delta(q, \epsilon)$) instead of $(q, \langle a\rangle, q', p, u) \in \delta$ (resp. $(q, p, \langle/a\rangle, q', u), (q, \epsilon, q', u) \in \delta$).

A detailed semantics definition can be found in the extended version.

▶ **Example 3.2.** Figure 3 shows a NWT T_{ab}, with linear states displayed as circles and transitions as arrows. From the initial state i, T_{ab} branches nondeterministically into either state a_1 or b_1. In state a_1, T_{ab} checks that the input string is well-nested just as the NWA A_1 from Example 2.1. During this check, T_{ab} outputs $\langle a\rangle$ (resp. $\langle/a\rangle$) for each opening (resp. closing) input tag, effectively relabelling the input string to consist exclusively of a-labelled tags. In state a_2, T_{ab} inserts into the output string an arbitrary number of opening $\langle a\rangle$ tags, for which a matching number of $\langle/a\rangle$ tags are inserted in state f before T_{ab} accepts. The behaviour of T_{ab} in states b_1 and b_2 is analogous, but outputs consist only of b-labelled tags. Altogether, T_{ab} chooses nondeterministically some $x \in \{a, b\}$, relabels all tags of a well-nested input string into x-labelled tags and then appends a string of the form $\langle x\rangle^n \langle/x\rangle^n$.

The *image* $T(w)$ of a well-nested string $w \in \mathrm{NW}(\Sigma)$ under T is the set of all outputs of T on w according to some accepting run of T on w. This definition extends to sets of input strings in the natural way: For a set $S \subseteq \mathrm{NW}(\Sigma)$, we define $T(S) = \bigcup_{w \in S} T(w)$. The

[1] These three conditions make NWT roughly correspond to *synchronized visibly pushdown transducers* [8]; we mainly require them to ensure closure of regular nested word languages under NWT transduction.

domain $\mathcal{D}(T)$ of T is the set of all strings w such that $T(w) \neq \emptyset$, and the *range* $\mathcal{R}(T)$ of T is the set of all strings u such that there exists a $w \in \mathrm{NW}(\Sigma)$ with $u \in T(w)$, i.e. the set of all possible outputs of T.

We next define several restrictions on the expressiveness of NWT.

▶ **Definition 3.3.** Let $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ be a NWT. We call T

- *ϵ-free* if $P_\epsilon = \emptyset$ and δ contains no ϵ-transitions.
- *non-deleting* if the output component of every non-internal transition in δ is a non-empty string;
- *deterministic* (or a DNWT) if for every $q \in Q, p \in P$ and $a \in \Sigma$, it holds that $|\delta(q, \langle a\rangle)| = |\delta(q, p, \langle /a\rangle)| = 1$;
- a *relabelling* transducer if it is ϵ-free and for every $q, q' \in Q, p \in P$, $a \in \Sigma$ and $u \in \Sigma^*$, if $(q', p, u) \in \delta(q, \langle a\rangle)$, then $u \in \langle\Sigma\rangle$, and if $(q', u) \in \delta(q, p, \langle /a\rangle)$, then $u \in \langle /\Sigma\rangle$;
- *functional*, if for every $w \in \mathrm{NW}(\Sigma)$, it holds that $|T(w)| = 1$.

It is easy to see that the length of any output of an ϵ-free NWT is at most linear in the length of the input string, while outputs of general NWT may grow to an arbitrary length. We note that functionality, unlike the other restrictions defined here, is a *semantic* condition. We do not investigate in this paper the decidability or complexity of determining whether a NWT is functional; likely, techniques for Visibly Pushdown Transducers in [5] could be adapted for this purpose. Also, while determinism implies functionality, the converse does not hold.

The following lemma shows that we can assume without loss of generality that each transition of a NWT involves at most one input and at most one output symbol, i.e. each opening (closing) transition outputs at most one opening (closing) tag and each internal ϵ-transition outputs nothing.

▶ **Lemma 3.4.** *Each NWT $T = (Q, P, P_\epsilon, \Sigma, \delta, q_0, F)$ can be transformed in polynomial time into an NWT $T' = (Q', P', P'_\epsilon, \Sigma, \delta', q_0, F)$ with $T(w) = T'(w)$ for each $w \in NW(\Sigma)$, such that for any transition in δ' with output u, it holds that $|u| \leq 1$.*

We say that a NWT of this shape is in normal form.

In most of this paper, we restrict our attention to non-deleting transducers. This is because regular nested word languages are closed under transduction by non-deleting NWT, which does not hold in the presence of deletions (consider, for instance, a NWT deleting all matched opening and closing c-labelled tags on the regular input language $\{(\langle a\rangle\langle /a\rangle\langle c\rangle)^n(\langle /c\rangle\langle b\rangle\langle /b\rangle)^n \mid n \geq 0\}$). The practical motivation for desiring this property is the fact that the AXML setting assumes that function call results can be specified by standard XML schema languages, which are subclasses of regular nested word languages.

Moreover, for most of the transducer models examined here, non-deleting transducers are not a significant restriction when it comes to context-free games, as the following result states.

▶ **Lemma 3.5.** *Any context-free game $G = (\Sigma, \Gamma, R, T)$ with NWT R can be transformed in polynomial time into a game $G' = (\Sigma', \Gamma, R', T')$ such that R' is non-deleting and it holds that $JWin(G') \cap NW(\Sigma) = JWin(G)$.*

Using Lemma 3.4, it is comparatively easy (if tedious) to prove that non-deleting NWTs are closed under composition. This proof, like most proofs for properties of NWT in this section, follows proof ideas used in [5, 8] adapted to the specifics of NWT.

▶ **Proposition 3.6.** *Let T_1, T_2 be non-deleting NWT. Then there exists a non-deleting NWT T such that for all $w \in NW(\Sigma)$, it holds that $T(w) = (T_2 \circ T_1)(w) \overset{def}{=} T_2(T_1(w))$. This NWT T can be computed from T_1 and T_2 in polynomial time and is of size $\mathcal{O}(|T_1| \cdot |T_2|)$.*

Since we are solely interested in NWTs operating on well-nested strings, we restrict our attention to NWTs with well-nested domains. The following corollary to Proposition 3.6 justifies this restriction.

▶ **Corollary 3.7.** *Let T be a non-deleting NWT and A a NWA over alphabet Σ. Then, there exists a non-deleting NWT T' of size $\mathcal{O}(|T| \cdot |A|)$ such that $\mathcal{D}(T') = \mathcal{D}(T) \cap L(A)$ and $T'(w) = T(w)$ for each $w \in \mathcal{D}(T) \cap L(A)$.*

In order to prove closure of regular nested word languages under transduction by non-deleting NWT, we observe another helpful property of these transducers.

▶ **Lemma 3.8.** *Let T be a non-deleting NWT with $\mathcal{D}(T) \subseteq NW(\Sigma)$. Then $\mathcal{R}(T)$ is a regular language of nested words.*

▶ **Corollary 3.9.** *Regular nested word languages are closed under transduction by non-deleting NWT, i.e. if $L \subseteq NW(\Sigma)$ is regular and T an NWT, then $T(L)$ is regular.*

We now turn to the complexity of standard decision problems for NWT. The upper bounds use relatively simple constructions based on Proposition 3.6, while lower bounds follow from comparable results for NWA.

▶ **Theorem 3.10.** *The membership problem for non-deleting NWT (Given a non-deleting NWT T and strings $w, u \in NW(\Sigma)$, is $u \in T(w)$?) is in* PTIME.

▶ **Theorem 3.11.** *The nonemptiness problem for non-deleting NWT (Given a non-deleting NWT T, is there a string $w \in NW(\Sigma)$ with $T(w) \neq \emptyset$?) is* PTIME*-complete with regard to logspace reductions.*

▶ **Theorem 3.12.** *The type checking problem for non-deleting NWT (Given a non-deleting NWT T and NWA A_1, A_2, is $T(L(A_1)) \subseteq L(A_2)$?) is*
(a) EXPTIME*-complete in general, and*
(b) PTIME*-complete (w.r.t. logspace reductions) if A_2 is a DNWA.*

4 Games with general NWT replacement

Having laid the foundation with basic results on NWT, we now examine context-free games with NWT-based replacement. The main characteristic distinguishing general NWT from ϵ-free NWT is the fact that, for any input string w and NWT T, transducts in $T(w)$ may be arbitrarily large in the size of w. This behaviour is necessary if we want to simulate games with regular replacement languages (in the sense of [10]) by transducer-based games. As it turns out, however, NWT-based replacement is much more complex than that: the winning problem in games with replay becomes undecidable (as opposed to 2-EXPTIME with regular replacement languages), which may be proven by a rather straightforward reduction from the complement of the halting problem for Turing machines.

▶ **Theorem 4.1.** *For the class of games with NWT and Call depth $k \geq 2$,* JWin *is not recursively enumerable.*

Even the replay-free winning problem for JULIET is quite hard when using NWT for replacement – we show that this problem is complete for doubly exponential time. The lower bound uses a rather intricate reduction from a two-player tiling problem, while the upper bound is proven by reduction to the purely NWT-based problem of *alternating iterated transduction*, which can be proven to be in 2-EXPTIME.

▶ **Theorem 4.2.** *For the class of replay-free games with NWT,* JWIN *is* 2-EXPTIME-*complete.*

The lower bound proofs for both of these results require replacement transducers to output nested words of arbitrary depth. Considering our practical motivation, it is rarely required that function calls in Active XML documents return arbitrarily deep trees. Therefore, we now investigate the impact of limiting replacement transducers' output depth.

For simplicity's sake, we assume depth-boundedness as a *semantic* restriction, i.e. we assert that all outputs in $R(w)$ produced by a depth-bounded replacement transducer R on a string w obey a given upper bound on their depth, without examining the decidability and complexity of determining whether or not a given transducer is depth-bounded.

We note that NWT with an output depth linear in the size of the input string are already strictly more expressive than ϵ-free NWT, so the lower bounds from Section 5 also hold for NWT with linear output depth. As these lower bounds are already quite high, we focus only on transducers whose output depth is bounded by a constant.

▶ **Definition 4.3.** An NWT R is called *depth-bounded* if there is some constant $d \geq 0$ such that for any $w \in \mathrm{NW}(\Sigma)$ and any $w' \in R(w)$, the depth of w' is at most d.

Using depth-bounded NWT as replacement transducers places the complexity of the winning problem between those for general NWT and for ϵ-free NWT. The upper and lower bounds are proven similarly to those of Theorem 4.2, but use the fact that the stack size of a depth-bounded NWT on a fixed input is bounded by a constant.

▶ **Theorem 4.4.** *For the class of replay-free games with depth-bounded NWT,* JWIN *is* EXPSPACE-*complete.*

5 Games with ϵ-free NWT replacement

In this section, we examine context-free games with replacement relations given by ϵ-free NWT. As we shall see, this leads to a decidable winning problem for games with bounded replay, but non-elementary complexity in all but the easiest case. For the unbounded replay case, we can construct a rather straightforward reduction from the halting problem for TMs.

▶ **Theorem 5.1.** *For the class of games with ϵ-free NWT and unbounded replay,* JWIN *is undecidable.*

Different from games with general NWT, the winning problem for JULIET in games with ϵ-free NWT and fixed Call depth is decidable; however, the complexity of deciding JWIN is already non-elementary for Call depth 2.

▶ **Theorem 5.2.** *For the class of games with ϵ-free NWT and Call depth bounded by $d \geq 2$,* JWIN *is decidable, but not decidable in elementary time.*

Even for replay-free games with ϵ-free NWT, the complexity of deciding the winning problem for JULIET is still rather high. The lower bound is proven by reduction from a tiling problem, and the CO-NEXPTIME algorithm uses non-determinism to guess moves for ROMEO while trying out all possible strategies for JULIET by backtracking.

▶ **Theorem 5.3.** *For the class of replay-free games with ϵ-free NWT,* JWin *is complete for* co-NEXPTIME*.*

The non-elementary lower bound in Theorem 5.2 follows from the fact that, in each string returned by Romeo, Juliet may play Call arbitrarily often. On a return string corresponding to a path of length n, Juliet may play Call on all n nodes bottom-up, with each such Call doubling the number of nodes below the called node, inducing a non-elementary blow-up.

To avoid this, we now examine games with bounded *Call width*, where, intuitively, Juliet may only play Call for a bounded number of times in each replacement string given by Romeo. Note that Call width is counted within each individual replacement string – so, in a game of Call depth 3 and Call width c, if Juliet plays Call on some position of the input string, she may then place up to c calls within the string returned by Romeo, and again up to c calls in *each* of the depth-2 replacement strings resulting from those calls.

More formally, the Call width of a play Π is the maximum number of times Juliet plays Call in any replacement string given by Romeo in Π. This definition extends naturally into that of Call width of a strategy. Note that Call width only applies to *replacement* strings, so Juliet may still call arbitrarily many positions of the *input* string, even for games with Call width 0. For this reason, replay-free strategies always have bounded Call width.

The proof of Theorem 5.1 shows that JWin remains undecidable for games with unbounded Call depth, even with Call width bounded by 1. For bounded replay, though, the complexity of JWin collapses to that of the replay-free case if Call width is bounded.

▶ **Theorem 5.4.** *For the class of games with ϵ-free NWT, Call depth bounded by $d \geq 1$ and Call width bounded by $k \geq 1$,* JWin *is* co-NEXPTIME*-complete.*

As mentioned above, bounded Call width does not affect Juliet's options for Call moves on the input string, as we generally want Juliet to be able to at least process *all* function symbols in the input. Dropping this requirement (i.e. bounding *Call width including input*) yields at least an exponential improvement in complexity.

▶ **Theorem 5.5.** *For the class of games with ϵ-free NWT, Call depth bounded by d and Call width including input bounded by k,* JWin *is*

(a) co-NP*-complete for $d \geq 1$ and $k \geq 2$,*
(b) co-NP*-complete for $d \geq 2$ and $k \geq 1$, and*
(c) *in* PTIME *for $d = k = 1$.*

The upper bounds in Theorems 5.4 and 5.5 use a backtracking algorithm like the one for Theorem 5.3; in this case, however, bounded Call width reduces the size of both the decision tree for Juliet and the occurring replacement strings.

6 Games with relabelling replacement

As seen before, even the limited amount of insertion allowed by ϵ-free NWT renders the winning problem for Juliet quite complex. We now examine how this changes if we disallow insertion entirely. First, we show that the winning problem is greatly simplified by the fact that transducts of relabelling transducers do not require any additional space beyond that provided by the input. In fact, the upper bounds of Theorems 6.1 to 6.4 all use a nigh-trivial (alternating or nondeterministic) algorithm that simply simulates the game. Lower bounds, on the other hand, are proven by reduction from the word problem for linearly bounded (alternating) Turing machines (Theorems 6.1 and 6.3) and from standard logic-based problems (Theorems 6.2 and 6.4).

▶ **Theorem 6.1.** *For the class of games with relabelling transducers and unbounded replay,* JWin *is* EXPTIME*-complete.*

With limited or no replay, the complexity decreases even further.

▶ **Theorem 6.2.** *For any $k \geq 1$, for the class of games with relabelling transducers and bounded Call depth k,* JWin *is* PSPACE*-complete.*

As the winning problem for Juliet remains intractable (assuming PTIME $\neq$ PSPACE) for replay-free games with relabelling transducers, we now turn to the even more limited class of functional relabellings. Note that games with functional transducers are essentially "solitaire games" for Juliet, as they do not allow for any choice of transducts by Romeo.

▶ **Theorem 6.3.** *For the class of games with functional relabelling transducers and unbounded replay,* JWin *is* PSPACE*-complete.*

As for general relabelling transducers, the complexity of JWin is the same for games with bounded replay and no replay when restricted to functional relabelling transducers.

▶ **Theorem 6.4.** *For any $k \geq 1$, for the class of games with functional relabelling transducers and bounded Call depth k,* JWin *is* NP*-complete.*

We see that even in this very simple class of games, we still fail to obtain a PTIME upper bound. Careful examination of lower bound proofs shows that our semantics for replay-free games still allows for a sort of "hidden replay": On a string of the form $\langle a\rangle\langle b\rangle v\langle /b\rangle\langle /a\rangle$, if Juliet plays Call first on $\langle /b\rangle$ then on $\langle /a\rangle$, the substring v undergoes *two* transductions – one from the Call to $\langle /b\rangle$, another from the Call to $\langle /a\rangle$. This allows us to perform any number d of transductions on a given string by enclosing it inside d nested function symbols.

Excluding this hidden replay yields a very narrow restriction of context-free games, which we call *write-once* games. In these, no substring may be transduced more than once, i.e. Juliet may only play Call on any closing tag $\langle /a\rangle$ if the substring enclosed in it does not contain a substring on which Juliet has played Call before. Note that write-once games are always replay-free, but even weaker as far as Juliet's rewriting capabilities are concerned.

A slight adaptation of the proof of Theorem 6.2 shows that JWin remains PSPACE-hard for write-once games with arbitrary relabelling transducers; for functional (and deterministic) relabelling transducers, however, we can prove tractability. The proof constructs from a given game G a NWT R_{J} such that for each $w \in \text{NW}(\Sigma)$, the set of all strings into which Juliet way rewrite w in G is given by $R_{\text{J}}(w)$.

▶ **Theorem 6.5.** *For the class of write-once games with functional relabelling transducers,* JWin *is in* PTIME*.*

7 Conclusion

The research presented in this paper shows that a major challenge in using transducers for context-free games is finding sensible transducer models and strategy restrictions that do not cause a prohibitive increase in the complexity of the winning problem compared to context-free games without parameter transformation. This paper has made a first step towards identifying what suitable restrictions may look like; however, the few tractable cases identified here are still so restricted that they may be only of limited practical interest.

It is possible that the complexity of the winning problem in games with replacement transducers may be further reduced by restricting relevant schemas to be closer to practical schema specifications for XML (such as DTDs or XML Schema). However, since research in [10] indicates that specifications of input schemas for external services influence the complexity of the safe rewriting problem, further research might be necessary to find transducer models whose input and output schemas can be described by DTDs or XML Schema.

References

1 Serge Abiteboul, Omar Benjelloun, and Tova Milo. The Active XML project: an overview. *VLDB J.*, 17(5):1019–1040, 2008. `doi:10.1007/s00778-007-0049-y`.

2 Serge Abiteboul, Tova Milo, and Omar Benjelloun. Regular rewriting of active XML and unambiguity. In *PODS*, pages 295–303, 2005. `doi:10.1145/1065167.1065204`.

3 Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009. `doi:10.1145/1516512.1516518`.

4 Henrik Björklund, Martin Schuster, Thomas Schwentick, and Joscha Kulbatzki. On optimum left-to-right strategies for active context-free games. In *ICDT*, pages 105–116, 2013. `doi:10.1145/2448496.2448510`.

5 Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Properties of visibly pushdown transducers. In *MFCS*, pages 355–367, 2010. `doi:10.1007/978-3-642-15155-2_32`.

6 Tova Milo, Serge Abiteboul, Bernd Amann, Omar Benjelloun, and Frederic Dang Ngoc. Exchanging intensional XML data. *ACM Trans. Database Syst.*, 30(1):1–40, 2005. `doi:10.1145/1061318.1061319`.

7 Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Active context-free games. *Theory Comput. Syst.*, 39(1):237–276, 2006. `doi:10.1007/s00224-005-1278-3`.

8 Jean-François Raskin and Frédéric Servais. Visibly pushdown transducers. In *ICALP (2)*, pages 386–397, 2008. `doi:10.1007/978-3-540-70583-3_32`.

9 Martin Schuster. Transducer-based rewriting games for Active XML. *CoRR*, abs/1606.02879, 2016. URL: `http://arxiv.org/abs/1606.02879`.

10 Martin Schuster and Thomas Schwentick. Games for Active XML revisited. In *ICDT*, pages 60–75, 2015. `doi:10.4230/LIPIcs.ICDT.2015.60`.

11 Frédéric Servais. *Visibly pushdown transducers.* Dissertation, ULB Belgique, 2011. URL: `http://theses.ulb.ac.be/ETD-db/collection/available/ULBetd-09292011-142239/`.

12 Alex Thomo, S. Venkatesh, and Ying Ying Ye. Visibly pushdown transducers for approximate validation of streaming xml. In *FoIKS*, pages 219–238, Berlin, Heidelberg, 2008. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1786094.1786112`.

Vector Reachability Problem in $\mathrm{SL}(2,\mathbb{Z})$[*]

Igor Potapov[1] **and Pavel Semukhin**[2]

1 **Department of Computer Science, University of Liverpool, United Kingdom**
potapov@liverpool.ac.uk

2 **Department of Computer Science, University of Liverpool, United Kingdom**
semukhin@liverpool.ac.uk

Abstract

The decision problems on matrices were intensively studied for many decades as matrix products play an essential role in the representation of various computational processes. However, many computational problems for matrix semigroups are inherently difficult to solve even for problems in low dimensions and most matrix semigroup problems become undecidable in general starting from dimension three or four.

This paper solves two open problems about the decidability of the vector reachability problem over a finitely generated semigroup of matrices from $\mathrm{SL}(2,\mathbb{Z})$ and the point to point reachability (over rational numbers) for fractional linear transformations, where associated matrices are from $\mathrm{SL}(2,\mathbb{Z})$. The approach to solving reachability problems is based on the characterization of reachability paths between points which is followed by the translation of numerical problems on matrices into computational and combinatorial problems on words and formal languages. We also give a geometric interpretation of reachability paths and extend the decidability results to matrix products represented by arbitrary labelled directed graphs. Finally, we will use this technique to prove that a special case of the scalar reachability problem is decidable.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems, F.1.1 Models of Computation

Keywords and phrases Matrix Semigroup, Vector Reachability Problem, Special Linear Group, Linear Fractional Transformation, Automata and Formal Languages

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.84

1 Introduction

Decision problems on matrices were intensively studied from 1947 when A. Markov showed the connection between classical computations and problems for matrix semigroups [24]. Moreover matrix products play an essential role in the representation of various computational processes, i.e., linear recurrent sequences [18, 27, 28], arithmetic circuits [14], hybrid and dynamical systems [26, 3], probabilistic and quantum automata [7], stochastic games, broadcast protocols [13], optical systems, etc. New algorithms for solving reachability problems in matrix semigroups can be incorporated into software verification tools and used for analysis of mathematical models in physics, chemistry, biology, ecology, and economics.

However, many computational problems for matrix semigroups are inherently difficult to solve even when the problems are considered in dimension two, and most of these problems become undecidable in general starting from dimension three or four. Examples of such problems are the Membership problem (including the special cases of the Mortality and

[*] This work was supported by EPSRC grant "Reachability problems for words, matrices and maps" (EP/M00077X/1).

Identity problems), vector reachability, scalar reachability, freeness problem and the emptiness problem of matrix semigroups intersection [6]. All above problems are tightly connected, including three central problems:

- **The membership problem:** Let $S = \langle G \rangle$ be a semigroup generated by a finite set G of $n \times n$ matrices. Determine whether a given matrix M belongs to S, that is, determine whether there exists a sequence of matrices $M_1, M_2, \ldots, M_k$ in G such that $M = M_1 \cdot M_2 \cdot \ldots \cdot M_k$
- **The vector reachability problem:** Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors and S be a given finitely generated semigroup of $n \times n$ matrices. Determine whether there is a matrix $M \in S$ such that $M\mathbf{x} = \mathbf{y}$.
- **The scalar reachability problem:** Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors, λ be a scalar, and S be a given finitely generated semigroup of $n \times n$ matrices. Determine whether there is a matrix $M \in S$ such that $\mathbf{x}^\top M\mathbf{y} = \lambda$.

The vector reachability problem can be seen as a parameterized version of the membership problem, where some elements of a matrix M are either independent variables or variables linked by some equations. In contrast to the original membership problem, where all values of M are constants, in vector reachability we may have an infinite set of matrices that can transform a vector $\mathbf{x}$ to $\mathbf{y}$. Thus the decidability results for the membership cannot be directly applied to the vector reachability problem.

The scalar reachability can be viewed as a vector to hyperplane reachability problem. Indeed, we can rewrite the equation $\mathbf{x}^\top M\mathbf{y} = \lambda$ as a system of two equations: $M\mathbf{y} = \mathbf{z}$ and $\mathbf{x}^\top\mathbf{z} = \lambda$. So, the question becomes if there is a matrix $M \in S$ that maps a given vector $\mathbf{y}$ to a vector $\mathbf{z}$ that lies on a hyperplane $\mathbf{x}^\top\mathbf{z} = \lambda$. Because there are infinitely many vectors on a hyperplane, decidability of the scalar reachability problem does not follow directly from the decidability of the vector reachability problem.

Most of the problems such as membership, vector reachability and freeness are undecidable for 3×3 integer matrices. The undecidability proofs in matrix semigroups are mainly based on various techniques and methods of embedding universal computations into three and four dimensional matrices and their products. The case of dimension two is the most intriguing one since there is some evidence that if these problems are undecidable, then this cannot be proved using a construction similar to the one used for dimensions 3 and 4. In particular, there is no injective semigroup morphism from pairs of words over any finite alphabet (with at least two elements) into complex 2×2 matrices [8], which means that the coding of independent pairs of words in 2×2 complex matrices is impossible and the exact encoding of the Post Correspondence Problem or a computation of a Turing Machine cannot be used directly for proving undecidability in 2×2 matrix semigroups over $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{C}$. The only undecidability result in dimension two for the vector reachability and the membership problems has been shown in the case of 2×2 matrices over quaternions [4].

The main hypothesis is that problems for 2×2 matrix semigroups over integers, rationals or complex numbers could be decidable, but not much is known about the status of these problems. Recently, there was some progress on the *Membership problem*, which was shown to be decidable in $\mathrm{SL}(2,\mathbb{Z})$, and the *Identity problem*, which was shown to be decidable in $\mathbb{Z}^{2\times 2}$ [11]. Later the decidability of the *Freeness problem* (that is, to decide whether each element can be expressed uniquely as a product of generating matrices) was shown for $\mathrm{SL}(2,\mathbb{Z})$ [9]. On the other hand, the Mortality, Identity and vector reachability problems were shown to be at least NP-hard for $\mathrm{SL}(2,\mathbb{Z})$ in [5, 6], but for the modular group the membership was shown to be decidable in polynomial time by Gurevich and Schupp [16].

The algorithmic properties of $\mathrm{SL}(2,\mathbb{Z})$ are important in the context of many fundamental problems in hyperbolic geometry [34, 10, 12], dynamical systems [29], Lorenz/modular knots [22], braid groups [30], particle physics, high energy physics [33], M/string theories [15], ray tracing analysis, music theory [25] and can lead to further decidability results in $\mathbb{Z}^{2\times 2}$ using matrix presentation in the Smith normal form.

This paper solves two open problems about the decidability of the vector reachability problem for finitely generated semigroups of matrices from $\mathrm{SL}(2,\mathbb{Z})$ and the point to point reachability (over rational numbers) for fractional linear transformations $f_M(x) = \frac{ax+b}{cx+d}$, where the associated matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ belongs to $\mathrm{SL}(2,\mathbb{Z})$. The approach to solving these reachability problems for 2×2 matrix semigroups is based on the analysis of reachability paths between vectors or points. This analysis is then used to translate the numerical reachability problems into computational problems on words and regular languages. We also present several extensions of our main results, give a geometric interpretation of reachability paths, and use this technique to solve a special case of the scalar reachability problem.

The decidability proof of the vector reachability problem in dimension two presented in this paper is the first nontrivial new result for solving vector reachability problems since 1996 when it was shown that the problem is decidable for any commutative matrix semigroup in any dimension [1] and for a special case of non-commuting matrices [20]. On the other hand, in the general case of non-commuting matrices the problem is known to be undecidable already for integer matrices in dimension three [17].

2 Preliminaries

The integers and rationals are denoted by $\mathbb{Z}$ and $\mathbb{Q}$, respectively, and $\mathrm{SL}(2,\mathbb{Z})$ is a group of 2×2 integer matrices with determinant 1. The notation $a \mid b$ means that a divides b, and $a \nmid b$ means that a does not divide b, when a and b are integer numbers.

▶ **Definition 1.** With each matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathrm{SL}(2,\mathbb{Z})$ we associate a fractional linear map (also called Möbius transformation) $f_M : \mathbb{Q} \to \mathbb{Q}$ defined as $f_M(x) = \frac{ax+b}{cx+d}$. This definition can be extended to $f : \mathbb{Q}\cup\{\infty\} \to \mathbb{Q}\cup\{\infty\}$ in a natural way by setting $f_M(\infty) = \frac{a}{c}$ if $c \neq 0$, $f_M(\infty) = \infty$ if $c = 0$, and $f_M(x) = \infty$ if $cx + d = 0$.

Note that we have $f_{M_1} \circ f_{M_2} = f_{M_1 M_2}$ for any matrices M_1 and M_2.

Let $M_1, \dots, M_n$ be a finite collection of matrices. Then $\langle M_1, \dots, M_n\rangle$ denotes the multiplicative semigroup (including the identity matrix) generated by $M_1, \dots, M_n$.

▶ **Definition 2.** The *vector reachability problem* in $\mathrm{SL}(2,\mathbb{Z})$ is defined as follows: Given two vectors $\mathbf{x}$ and $\mathbf{y}$ with integer coefficients and a finite collection of matrices $M_1, \dots, M_n$ from $\mathrm{SL}(2,\mathbb{Z})$, decide whether there exists a matrix $M \in \langle M_1, \dots, M_n\rangle$ such that $M\mathbf{x} = \mathbf{y}$.

▶ **Definition 3.** The *reachability problem by fractional linear transformations* in $\mathrm{SL}(2,\mathbb{Z})$ is defined as follows: Given two rational numbers x and y and a finite collection of matrices $M_1, \dots, M_n$ from $\mathrm{SL}(2,\mathbb{Z})$, decide whether there exists a matrix $M \in \langle M_1, \dots, M_n\rangle$ such that $f_M(x) = y$.

The main result of our paper is that the vector reachability problem and the reachability problem by fractional linear transformations for $\mathrm{SL}(2,\mathbb{Z})$ are decidable (Theorem 14). Both proofs follow the same pattern. We will use the fact that any matrix M from $\mathrm{SL}(2,\mathbb{Z})$ can be expressed as product of matrices $S = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ and $R = \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}$. So we can represent any $M \in \mathrm{SL}(2,\mathbb{Z})$ by a word w in the alphabet $\Sigma = \{S, R\}$.

The main idea of the proof is to show that the solution set of the equation $M\mathbf{x} = \mathbf{y}$ has the form $\left\{ B \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^t C \;:\; t \in \mathbb{Z} \right\}$, where B and C are some matrices from $\mathrm{SL}(2,\mathbb{Z})$ that can be computed in PTIME from $\mathbf{x}$, $\mathbf{y}$ (Theorem 8). Similarly, the solution set of the equation $f_M(x) = y$ can be presented as a union of two sets of such form (Theorem 10). After translating matrices into words, these sets become regular languages. On the other hand, the language that corresponds to the semigroup $\langle M_1, \dots, M_n \rangle$ is also regular. Indeed, if M_i corresponds to the word w_i, then the semigroup $\langle M_1, \dots, M_n \rangle$ translates into the language $(w_1 + \dots + w_n)^*$. The last step of the proof is to show that the emptiness problem of the intersection of two such languages is decidable (Proposition 13).

Here is a more detailed description of our proofs. Let $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$. To show that the equation $M\mathbf{x} = \mathbf{y}$ defines a regular language we must solve the following system of three equations in four unknown variables:

$$x_1 a + x_2 b = y_1 \qquad x_1 c + x_2 d = y_2 \qquad ad - bc = 1$$

Choosing b as a free parameter, we can reduce it to the following system of linear congruence equations:

$$x_2 b \equiv y_1 \pmod{x_1} \qquad y_2 b \equiv -x_1 \pmod{y_1} \qquad x_2 y_2 b \equiv y_1 y_2 - x_1 x_2 \pmod{x_1 y_1}$$

It can be shown that the above system either has no solutions or it has a solution of the form $b = b_1 t + b_2$, where $t \in \mathbb{Z}$, and hence all coefficients of the matrix M are linear functions of t. In Proposition 7 we will show that such matrices can be written in the form $M = B \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}^t C$, where B, C are some matrices from $\mathrm{SL}(2,\mathbb{Z})$, k is a fixed integer number and $t \in \mathbb{Z}$ is a free parameter. After that it is not hard to see that such solution translates into a regular language.

We will use a similar approach to prove that the equation $f_M(x) = y$ also defines a regular language. In fact, we will do it by showing that the solution set of $f_M(x) = y$ is equal to the union of the solution sets of the equations $M\mathbf{x} = \mathbf{y}$ and $M\mathbf{x} = -\mathbf{y}$ for suitable vectors $\mathbf{x}$ and $\mathbf{y}$.

The final step is to show that there is an algorithm that decides whether the intersection of two regular subsets of $\mathrm{SL}(2,\mathbb{Z})$ is empty or not. Our idea relies on the fact that the intersection of two regular languages is regular, and that the emptiness problem for regular languages is decidable. The problem here is that we cannot apply these facts directly because for each matrix $M \in \mathrm{SL}(2,\mathbb{Z})$ there are infinitely many words $w \in \{S, R\}^*$ that correspond to M, and only some of them may appear in the given language. However there is only one *reduced* word that corresponds to M, that is, the word that does not have a substring of the form SS or RRR. So, our solution is to take any automaton $\mathcal{A}$ and turn it into a new automaton $\widetilde{\mathcal{A}}$ that accepts the same language as $\mathcal{A}$ plus all reduced words w that correspond to non-reduced words w' accepted by $\mathcal{A}$.

The construction of the automaton $\widetilde{\mathcal{A}}$ was inspired by a similar construction from [11]. Note that in $\mathrm{SL}(2,\mathbb{Z})$ we have an equality $S^2 = R^3 = -I$. Thus to construct $\widetilde{\mathcal{A}}$ we add to $\mathcal{A}$ a new ε-transition from a state q_1 to a state q_2 if there is a run of $\mathcal{A}$ from q_1 to q_2 labelled by SS or RRR. We will apply this procedure iteratively until no new ε-transitions can be added. However we need to keep track of sign changes when we add new ε-transitions. To achieve this we will use *signed automata*, which are slight modifications of the usual finite automata but they take into account such sign changes.

Now to solve the emptiness problem for the intersection of two regular languages L_1 and L_2, we take the signed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ that accept L_1 and L_2, respectively, and construct new automata $\widetilde{\mathcal{A}}_1$ and $\widetilde{\mathcal{A}}_2$ as described above. After that we can check whether $L(\widetilde{\mathcal{A}}_1) \cap L(\widetilde{\mathcal{A}}_2) \neq \emptyset$.

In the Section 4 we will show how to extend these decidability results to arbitrary regular subsets of $\mathrm{SL}(2,\mathbb{Z})$, i.e., subsets that are defined by finite automata. Using this technique we will show how to algorithmically solve the equation $M_1^{x_1} \cdots M_k^{x_k}\mathbf{x} = N_1^{y_1} \cdots N_l^{y_l}\mathbf{y}$, where $\mathbf{x}, \mathbf{y}$ are given vectors from $\mathbb{Z} \times \mathbb{Z}$, the matrices $M_1, \dots, M_k$ and $N_1, \dots, N_l$ are from $\mathrm{SL}(2,\mathbb{Z})$, and $x_1, \dots, x_k$ and $y_1, \dots, y_l$ are unknown non-negative integers. Furthermore, we will show how to apply this method to prove that a special case of the scalar reachability problem is decidable.

All missing proofs can be found in the extended version of this paper, which is available online on arXiv.org [31].

3 Main results

The characterization of the solution set of the equation $M\mathbf{x} = \mathbf{y}$ given in Theorem 8 will follow from Propositions 5 and 7. First, we prove one simple lemma which we will use several times in our arguments.

▶ **Lemma 4.** *Let* $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ *and* $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ *be vectors from* $\mathbb{Z} \times \mathbb{Z}$ *and* M *be a matrix from* $\mathrm{SL}(2,\mathbb{Z})$ *such that* $M\mathbf{x} = \mathbf{y}$. *Then* $\gcd(x_1, x_2) = \gcd(y_1, y_2)$.

Proof. Take any $k \in \mathbb{Z}$ such that $k \mid x_1, x_2$ and let $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Then from $M\mathbf{x} = \mathbf{y}$ we have $y_1 = ax_1 + bx_2$ and $y_2 = cx_1 + dx_2$. Thus $k \mid y_1, y_2$. Now since $M \in \mathrm{SL}(2,\mathbb{Z})$, M^{-1} is also in $\mathrm{SL}(2,\mathbb{Z})$, and $M\mathbf{x} = \mathbf{y}$ is equivalent to $M^{-1}\mathbf{y} = \mathbf{x}$. So, if $k \in \mathbb{Z}$ is any number such that $k \mid y_1, y_2$, then $k \mid x_1, x_2$. Therefore, $\gcd(x_1, x_2) = \gcd(y_1, y_2)$. ◀

▶ **Proposition 5.** *Let* $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ *and* $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ *be two vectors from* $\mathbb{Z} \times \mathbb{Z}$, *such that* $\mathbf{x}$ *is not equal to the zero vector* $\mathbf{0}$, *and consider the matrix equation* $M\mathbf{x} = \mathbf{y}$, *where* M *is an unknown matrix from* $\mathrm{SL}(2,\mathbb{Z})$. *Then either this equation does not have a solution or all its solutions are given by* $M = tA_1 + A_2$, *where* t *is any integer number,* A_1, A_2 *are some matrices from* $\mathbb{Z}^{2\times 2}$ *such that* A_1 *is a nonzero matrix. Moreover, there is a polynomial time algorithm that determines whether such an equation has a solution and if so, finds it.*

Proof. See Section A of the Appendix in [31]. ◀

For the next proposition we will need the following theorem about the Smith normal form of a matrix.

▶ **Theorem 6** (Smith normal form [19])**.** *For any nonzero matrix* $A \in \mathbb{Z}^{2\times 2}$, *there are matrices* B, C *from* $\mathrm{SL}(2,\mathbb{Z})$ *such that* $A = B\begin{bmatrix} t_1 & 0 \\ 0 & t_2 \end{bmatrix}C$ *for some* $t_1, t_2 \in \mathbb{Z}$ *such that* $t_1 \neq 0$ *and* $t_1 \mid t_2$. *Moreover,* B, C, t_1, t_2 *can be computed in polynomial time.*

▶ **Proposition 7.** *Let A_1 and A_2 be matrices from $\mathbb{Z}^{2\times 2}$ such that A_1 is a nonzero matrix and, for every $t \in \mathbb{Z}$, we have $tA_1 + A_2 \in \mathrm{SL}(2,\mathbb{Z})$. Then there are matrices B and C from $\mathrm{SL}(2,\mathbb{Z})$ and $k \in \mathbb{Z}$ such that*

$$tA_1 + A_2 = BT^{kt}C \quad \textit{for every } t \in \mathbb{Z},$$

where $T = \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix} \in \mathrm{SL}(2,\mathbb{Z})$. Moreover, B, C, and k can be computed in polynomial time.

Proof. Let $A_1 = \begin{bmatrix}a_1 & b_1\\ c_1 & d_1\end{bmatrix}$ and $A_2 = \begin{bmatrix}a_2 & b_2\\ c_2 & d_2\end{bmatrix}$. By the assumption, for every $t \in \mathbb{Z}$,

$$\begin{vmatrix}a_1t + a_2 & b_1t + b_2\\ c_1t + c_2 & d_1t + d_2\end{vmatrix} = 1.$$

That is $(a_1t+a_2)(d_1t+d_2)-(b_1t+b_2)(c_1t+c_2) = 1$ or $(a_1d_1 - b_1c_1)t^2 + (a_1d_2 + a_2d_1 - b_1c_2 - b_2c_1)t + a_2d_2 - b_2c_2 = 1$ for all $t \in \mathbb{Z}$. Therefore, $a_1d_1 - b_1c_1 = 0$, $a_1d_2 + a_2d_1 - b_1c_2 - b_2c_1 = 0$, and $a_2d_2 - b_2c_2 = 1$. In particular, $\det(A_1) = 0$ and $\det(A_2) = 1$.

By Theorem 6, there are matrices $F, G \in \mathrm{SL}(2,\mathbb{Z})$ such that $A_1 = F\begin{bmatrix}k & 0\\ 0 & l\end{bmatrix}G$ for some $k, l \in \mathbb{Z}$ such that $k \mid l$. Since $\det(A_1) = 0$ we have that $kl = 0$. However if $k = 0$ and $l = 0$, then A_1 is equal to the zero matrix, contrary to the assumption. Hence we must have that $k \neq 0$ and $l = 0$.

Now $F^{-1}(tA_1 + A_2)G^{-1} = \begin{bmatrix}kt + a & b\\ c & d\end{bmatrix}$, for some $a, b, c, d \in \mathbb{Z}$. Note that since $\det(F) = \det(G) = \det(tA_1 + A_2) = 1$, we have $\begin{vmatrix}kt + a & b\\ c & d\end{vmatrix} = dkt + ad - bc = 1$ for every $t \in \mathbb{Z}$. Hence $dk = 0$ and so $d = 0$. Substituting $d = 0$ in the above equation, we obtain $bc = -1$. Since b and c are integers, there are only two possibilities: $b = 1$, $c = -1$, or $b = -1$, $c = 1$. So the above matrix actually looks like $F^{-1}(tA_1 + A_2)G^{-1} = \begin{bmatrix}kt + a & \mp 1\\ \pm 1 & 0\end{bmatrix}$. Therefore, $T^{-c(kt+a)}F^{-1}(tA_1 + A_2)G^{-1} = D$, where $c = \pm 1$ and $D = \begin{bmatrix}0 & \mp 1\\ \pm 1 & 0\end{bmatrix} \in \mathrm{SL}(2,\mathbb{Z})$. Hence $tA_1 + A_2 = FT^{(ck)t}T^{ca}DG$. Note that F and $T^{ca}DG$ are in $\mathrm{SL}(2,\mathbb{Z})$. This completes the proof. The bound on complexity follows from the fact that F and G can be computed in PTIME by Theorem 6. ◀

As a corollary of Propositions 5 and 7 we obtain the following theorem.

▶ **Theorem 8.** *Let $\mathbf{x} = \begin{bmatrix}x_1\\ x_2\end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix}y_1\\ y_2\end{bmatrix}$ be vectors from $\mathbb{Z} \times \mathbb{Z}$ such that $\mathbf{x} \neq \mathbf{0}$, and consider the matrix equation $M\mathbf{x} = \mathbf{y}$, where M is an unknown matrix from $\mathrm{SL}(2,\mathbb{Z})$. Then either this equation does not have a solution or all its solutions are given by the following formula $M = B\begin{bmatrix}1 & k\\ 0 & 1\end{bmatrix}^t C$, where $t \in \mathbb{Z}$.*

In the above expression B and C are some matrices from $\mathrm{SL}(2,\mathbb{Z})$, and k is an integer number. Moreover, there is a polynomial time algorithm that determines whether such an equation has a solution and if so, finds the suitable matrices B, C and the integer k.

In Section 4 we will give a geometric interpretation of reachability paths (Figure 1 and Proposition 16), using which we can prove the following corollary.[1] The proof itself can be found in Section D of the Appendix in [31].

▶ **Corollary 9.** *The value of the parameter k in Theorem 8 is equal to 1.*

Theorem 8 provides us with a characterization of the matrices $M \in \mathrm{SL}(2,\mathbb{Z})$ that map vector $\mathbf{x}$ to vector $\mathbf{y}$. This characterization will be used later to prove the decidability of the vector reachability problem. We now give a similar characterization of the matrices $M \in \mathrm{SL}(2,\mathbb{Z})$ for which the fractional linear transformation f_M maps a number x to number y. In fact, we will do this by reducing the problem to finding the solutions of the equation $M\mathbf{x} = \mathbf{y}$ which we discussed above.

▶ **Theorem 10.** *Let x and y be rational numbers and let $\mathcal{F}(x,y)$ be the following set of matrices from* $\mathrm{SL}(2,\mathbb{Z})$:

$$\mathcal{F}(x,y) = \{M \in \mathrm{SL}(2,\mathbb{Z}) \ :\ f_M(x) = y\}.$$

Then $\mathcal{F}(x,y) = \mathcal{F}_1(x,y) \cup \mathcal{F}_2(x,y)$, where each $\mathcal{F}_i(x,y)$ is either empty or has the form

$$\mathcal{F}_i(x,y) = \{B_i T^t C_i \ :\ t \in \mathbb{Z}\},$$

where B_i and C_i are some matrices from $\mathrm{SL}(2,\mathbb{Z})$*. Moreover, there is a polynomial time algorithm that determines whether each $\mathcal{F}_i(x,y)$ is empty or not and in the latter case finds corresponding matrices B_i and C_i.*

Proof. Let us write the numbers x and y as $x = \frac{x_1}{x_2}$ and $y = \frac{y_1}{y_2}$, where we assume that $\gcd(x_1,x_2) = \gcd(y_1,y_2) = 1$. Consider the equation $f_M(x) = y$, where $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is an unknown matrix from $\mathrm{SL}(2,\mathbb{Z})$. We can rewrite it as

$$\frac{a\frac{x_1}{x_2} + b}{c\frac{x_1}{x_2} + d} = \frac{y_1}{y_2} \quad \text{or} \quad \frac{ax_1 + bx_2}{cx_1 + dx_2} = \frac{y_1}{y_2}. \tag{1}$$

Consider the vectors $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$, and $\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$, where $\mathbf{z}$ is the vector with coordinates $z_1 = ax_1 + bx_2$ and $z_2 = cx_1 + dx_2$. So we have that $\mathbf{z} = M\mathbf{x}$. In this notation Equation (1) is equivalent to the fact that vector $\mathbf{z} = M\mathbf{x}$ belongs to the set $\{k\mathbf{y} \ :\ k \in \mathbb{Z}\}$.

Recall that $\gcd(x_1,x_2) = 1$ and hence, by Lemma 4, we also have that $\gcd(z_1,z_2) = 1$. Thus if $\mathbf{z} = k\mathbf{y}$ for some $k \in \mathbb{Z}$, then we must have that $k = \pm 1$. In other words, we showed that Equation (1) is equivalent to two matrix equations: $M\mathbf{x} = \mathbf{y}$ and $M\mathbf{x} = -\mathbf{y}$. So we have that $\mathcal{F}(x,y) = \mathcal{F}_1(x,y) \cup \mathcal{F}_2(x,y)$, where

$$\mathcal{F}_1(x,y) = \{M \in \mathrm{SL}(2,\mathbb{Z}) \ :\ M\mathbf{x} = \mathbf{y}\} \quad \text{and} \quad \mathcal{F}_2(x,y) = \{M \in \mathrm{SL}(2,\mathbb{Z}) \ :\ M\mathbf{x} = -\mathbf{y}\}.$$

Note that $\mathbf{x} \neq \mathbf{0}$ because $x_2 \neq 0$. Hence by Theorem 8 and Corollary 9, each $\mathcal{F}_i(x,y)$ is either empty or has the form $\mathcal{F}_i(x,y) = \{B_i T^t C_i \ :\ t \in \mathbb{Z}\}$ for some B_i and C_i from $\mathrm{SL}(2,\mathbb{Z})$ which can be computed in polynomial time. ◀

[1] Even though we use Corollary 9 in the proofs of Theorem 10 and Proposition 18, it is not essential there for proving decidability. Namely, all references to Corollary 9 in these proofs can be replaced by references to Theorem 8, at the same time replacing T with T^k where appropriate.

Now we will use signed automata to prove that the emptiness problem for the intersection of two regular subsets of $\mathrm{SL}(2,\mathbb{Z})$ is decidable.

Consider an alphabet $\Sigma = \{S, R\}$ consisting of two symbols S and R and define the mapping $\varphi : \Sigma \to \mathrm{SL}(2,\mathbb{Z})$ as follows: $\varphi(S) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ and $\varphi(R) = \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}$. We can extend this mapping to the morphism $\varphi : \Sigma^* \to \mathrm{SL}(2,\mathbb{Z})$ in a natural way. The matrices $\varphi(S)$ and $\varphi(R)$ are in fact generators of $\mathrm{SL}(2,\mathbb{Z})$, so φ is surjective. We call a word $w \in \Sigma^*$ *reduced* if it does not have substrings of the form SS or RRR. In our proof we will make use of the following well-known fact.

▶ **Theorem 11** ([21, 23, 32]). *For every $M \in \mathrm{SL}(2,\mathbb{Z})$, there exists a unique reduced word $w \in \Sigma^*$ such that either $M = \varphi(w)$ or $M = -\varphi(w)$.*

▶ **Definition 12.** A *signed automaton* $\mathcal{A} = (\Sigma, Q, I, \Delta, F^+, F^-)$ is a (non-deterministic) finite automaton whose final states are divided into two (not necessarily disjoint) subsets F^+ and F^-.

A *signed language* accepted by a signed automaton $\mathcal{A}$ is a pair $L(\mathcal{A}) = (L(\mathcal{A})^+, L(\mathcal{A})^-)$, where $L(\mathcal{A})^+$ and $L(\mathcal{A})^-$ consists of the words $w \in \Sigma^*$ for which there is a run of $\mathcal{A}$ that ends in the set F^+ or F^-, respectively. Note that we do not assume that $L(\mathcal{A})^+$ and $L(\mathcal{A})^-$ are disjoint.

Let $L = (L^+, L^-)$ be a signed language, then we define a regular subset of $\mathrm{SL}(2,\mathbb{Z})$ corresponding to this language as $\varphi(L) = \{\varphi(w) : w \in L^+\} \cup \{-\varphi(w) : w \in L^-\}$.

The following proposition is an important ingredient of our main results.

▶ **Proposition 13.** *There is an algorithm that for any given regular signed languages L_1 and L_2 over the alphabet Σ, decides whether $\varphi(L_1) \cap \varphi(L_2)$ is empty or not.*

Proof. See Section B of the Appendix in [31]. ◀

We are now ready to prove our main results.

▶ **Theorem 14.** *The vector reachability problem and the reachability problem by fractional linear transformations in $\mathrm{SL}(2,\mathbb{Z})$ are decidable.*

Proof. Suppose $M_1, \dots, M_n$ is a given finite collection of matrices from $\mathrm{SL}(2,\mathbb{Z})$. Let $w_1, \dots, w_n \in \Sigma^*$ be some words, not necessarily reduced, such that $M_i = \varphi(w_i)$, for $i = 1, \dots, n$. Define the language $\mathcal{L}_{semigr}$ that corresponds to the semigroup $\langle M_1, \dots, M_n \rangle$ as $\mathcal{L}_{semigr} = (w_1 + w_2 + \cdots + w_n)^*$.

Recall that in the vector reachability problem we are given two vectors $\mathbf{x}$ and $\mathbf{y}$ from $\mathbb{Z} \times \mathbb{Z}$, and we ask if there is a matrix $M \in \langle M_1, \dots, M_n \rangle$ such that $M\mathbf{x} = \mathbf{y}$. We want to construct a regular language $\mathcal{L}^{\mathrm{vrp}}_{\mathbf{x},\mathbf{y}}$ that corresponds to these matrices.

If $\mathbf{x} = \mathbf{0}$ and $\mathbf{y} \neq \mathbf{0}$, then we set $\mathcal{L}^{\mathrm{vrp}}_{\mathbf{x},\mathbf{y}} = \emptyset$ because in this case the equation $M\mathbf{x} = \mathbf{y}$ does not have a solution. On the other hand, if $\mathbf{x} = \mathbf{0}$ and $\mathbf{y} = \mathbf{0}$, then we set $\mathcal{L}^{\mathrm{vrp}}_{\mathbf{x},\mathbf{y}} = \{S, R\}^*$ because any matrix $M \in \mathrm{SL}(2,\mathbb{Z})$ satisfies the equation $M\mathbf{0} = \mathbf{0}$.

Now assume that $\mathbf{x} \neq \mathbf{0}$. Then by Theorem 8, the matrix equation $M\mathbf{x} = \mathbf{y}$ either has no solution, or its solution has the form $\{BT^tC : t \in \mathbb{Z}\}$, where $T = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, and B and C are some matrices from $\mathrm{SL}(2,\mathbb{Z})$. Moreover, B and C can be computed from $\mathbf{x}$ and $\mathbf{y}$ in PTIME. In the case when $M\mathbf{x} = \mathbf{y}$ has no solution, we set $\mathcal{L}^{\mathrm{vrp}}_{\mathbf{x},\mathbf{y}} = \emptyset$. If the solution set in non-empty, then we can rewrite it as

$$\{BT^tC : t \in \mathbb{Z}\} = \{BT^tC : t \geq 0\} \cup \{BT^{-t}C : t \geq 0\}.$$

Let u and v be words from Σ^* such that $B = \varphi(u)$ and $C = \varphi(v)$. It is easy to check that $T = \varphi(S^3R)$ and $T^{-1} = \varphi(R^5S)$. Hence $\mathcal{L}^{\text{vrp}}_{\mathbf{x},\mathbf{y}} = u(S^3R)^*v + u(R^5S)^*v$ is a regular language that describes the solutions of the equation $M\mathbf{x} = \mathbf{y}$ in $\mathrm{SL}(2,\mathbb{Z})$.

In a similar way we can construct a regular language $\mathcal{L}^{\text{flt}}_{x,y}$ that corresponds to the reachability problem by fractional linear transformations from x to y. By Theorem 10, the set $\mathcal{F}(x,y)$ of matrices from $\mathrm{SL}(2,\mathbb{Z})$ that satisfy the equation $f_M(x) = y$ is equal to $\mathcal{F}(x,y) = \mathcal{F}_1(x,y) \cup \mathcal{F}_2(x,y)$, where each $\mathcal{F}_i(x,y)$ is either empty or has the form $\mathcal{F}_i(x,y) = \{B_iT^tC_i : t \in \mathbb{Z}\}$, where T is as above, and B_i and C_i are some matrices from $\mathrm{SL}(2,\mathbb{Z})$. All these matrices can be computed in PTIME from x and y.

We define $\mathcal{L}^{\text{flt}}_{x,y}$ as the union $\mathcal{L}^{\text{flt}}_{x,y} = \mathcal{L}_1 \cup \mathcal{L}_2$ of two regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$. If $\mathcal{F}_i(x,y)$ is empty, then we set $\mathcal{L}_i = \emptyset$. Otherwise, let u_i and v_i be words from Σ^* such that $B_i = \varphi(u_i)$ and $C_i = \varphi(v_i)$. Then we can define $\mathcal{L}_i$ as $\mathcal{L}_i = u_i(S^3R)^*v_i + u_i(R^5S)^*v_i$. Thus we defined a regular language $\mathcal{L}^{\text{flt}}_{x,y}$ that corresponds the solution set of the equation $f_M(x) = y$ in $\mathrm{SL}(2,\mathbb{Z})$.

We remind that in Proposition 13 we work with signed languages. Therefore, in what follows we convert every regular language L that we have constructed so far into a corresponding signed language $(L, \emptyset)$.

Finally, the vector reachability problem for $\mathbf{x}$ and $\mathbf{y}$ has a solution if and only if

$$\varphi\big((\mathcal{L}^{\text{vrp}}_{\mathbf{x},\mathbf{y}}, \emptyset)\big) \cap \varphi\big((\mathcal{L}_{semigr}, \emptyset)\big) \neq \emptyset.$$

Similarly, the reachability problem by fractional linear transformations for x and y has a solution if and only if

$$\varphi\big((\mathcal{L}^{\text{flt}}_{x,y}, \emptyset)\big) \cap \varphi\big((\mathcal{L}_{semigr}, \emptyset)\big) \neq \emptyset.$$

By Proposition 13 these questions are algorithmically decidable. ◀

A characterization of the matrices M from $\mathrm{SL}(2,\mathbb{Z})$ that satisfy the equation $M\mathbf{x} = \mathbf{y}$, which is given in Theorem 8, can be computed in polynomial time. However the overall complexity of the algorithm is EXPTIME if the entries of the matrices are given in binary presentation. This is due to the fact that a reduced word w that corresponds to a given matrix M, i.e., such that $M = \pm\varphi(w)$, has length exponential in the binary presentation of M. So computing symbolic presentations of given matrices and constructing automata for the languages $\mathcal{L}_{semigr}$, $\mathcal{L}^{\text{vrp}}_{\mathbf{x},\mathbf{y}}$ and $\mathcal{L}^{\text{flt}}_{x,y}$ takes exponential time. The next steps of the algorithm take only polynomial time in the size of these automata. However the PTIME algorithm for computing all mappings from $\mathbf{x}$ to $\mathbf{y}$ could be combined with the result of Gurevich and Schupp [16] to produce a polynomial time algorithm for the vector reachability problem over the modular group.

4 Geometric interpretation and extensions

Consider a semigroup generated by matrices $M_1, \dots, M_n$ from $\mathrm{SL}(2,\mathbb{Z})$. As we showed above, this semigroup can be described by a regular language which we called $\mathcal{L}_{semigr}$. It's not hard to see that the proof of Theorem 14 remains valid if we replace $\mathcal{L}_{semigr}$ by any other regular language, that is, a language defined by a finite automaton or a labelled transition system.

▶ **Proposition 15.** *Suppose that we are given a finite collection of matrices $M_1, \dots, M_n$ from $\mathrm{SL}(2,\mathbb{Z})$ and a regular language $L \subseteq \{1, \dots, n\}^*$. Consider the following generalized reachability problems:*

- **Generalized vector reachability problem.** *Given two vectors* $\mathbf{x}$ *and* $\mathbf{y}$ *with integer coefficients, decide whether there exists a word* $i_1 \ldots i_k$ *from the language* L *such that* $M_{i_1} \cdots M_{i_k}\mathbf{x} = \mathbf{y}$.
- **Generalized reachability problem by fractional linear transformations.** *Given two rational numbers* x *and* y*, decide whether there exists a word* $i_1 \ldots i_k$ *from* L *such that* $f_{M_{i_1} \cdots M_{i_k}}(x) = y$.

Then the above generalized reachability problems are decidable.

Proof. The proof of this proposition is similar to the proof of Theorem 14. Namely, it follows from the fact that a regular language L defines a regular subset in $\mathrm{SL}(2,\mathbb{Z})$ and Proposition 13, where we proved that the emptiness problem for the intersection of two regular subsets in $\mathrm{SL}(2,\mathbb{Z})$ is decidable. ◀

As an application of Proposition 15 let us consider the follow matrix equation

$$M_1^{x_1} \cdots M_k^{x_k}\mathbf{x} = N_1^{y_1} \cdots N_l^{y_l}\mathbf{y}, \tag{2}$$

where $x_1, \ldots, x_k$ and $y_1, \ldots, y_l$ are non-negative integers. In [1] it was proved that if $M_1, \ldots, M_k$ and $N_1, \ldots, N_l$ are commuting $n \times n$ matrices over algebraic numbers and $\mathbf{x}, \mathbf{y}$ are vectors with algebraic coefficients, then it is decidable in polynomial time whether the Equation (2) has a solution. On the other hand, in [2] it was shown that there is no algorithm for solving the equation $M_1^{x_1} \cdots M_k^{x_k} = Z$, where $M_1, \ldots, M_k$ are integer $n \times n$ matrices and Z is the zero matrix. Using the construction of Kronecker (or tensor) product of matrices, it is possible to show that the above-mentioned result implies that Equation (2) is algorithmically undecidable in general for non-commuting integer matrices $M_1, \ldots, M_k$ and $N_1, \ldots, N_l$.

However with the help of Proposition 15 we can algorithmically solve Equation (2) in the case when $M_1, \ldots, M_k$ and $N_1, \ldots, N_l$ are matrices from $\mathrm{SL}(2,\mathbb{Z})$ and the vectors $\mathbf{x}, \mathbf{y}$ have integer coefficients. Indeed, since the matrices from $\mathrm{SL}(2,\mathbb{Z})$ are invertible, we can rewrite (2) as $(N_l^{-1})^{y_l} \cdots (N_1^{-1})^{y_1} M_1^{x_1} \cdots M_k^{x_k}\mathbf{x} = \mathbf{y}$. It is not hard to see that $\{(N_l^{-1})^{y_l} \cdots (N_1^{-1})^{y_1} M_1^{x_1} \cdots M_k^{x_k} : x_1, \ldots, x_k,\, y_1, \ldots, y_l \in \mathbb{N} \cup \{0\}\}$ is a regular subset of $\mathrm{SL}(2,\mathbb{Z})$, and hence the problem is decidable. Using the same idea we can algorithmically solve Equation (2) also in the case when $x_1, \ldots, x_k$ and $y_1, \ldots, y_l$ are arbitrary integers and the matrices are from $\mathrm{SL}(2,\mathbb{Z})$.

In the rest of this section we will give a geometric interpretation of both reachability problems (Figure 1), which we will use later to solve a special case of the scalar reachability problem (Proposition 18).

▶ **Proposition 16.** *According to Theorem 8, the set of matrices* M *from* $\mathrm{SL}(2,\mathbb{Z})$ *that transform a vector* $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ *to a vector* $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ *has the form* $\mathcal{F} = \{BT^{kt}C \ : \ t \in \mathbb{Z}\}$.

Consider the equation $BT^{kt}C\mathbf{x} = \mathbf{y}$ *and let us make the following change of variables:* $\mathbf{u} = C\mathbf{x}$ *and* $\mathbf{v} = B^{-1}\mathbf{y}$: $\mathbf{x} \xrightarrow{C} \mathbf{u} \xrightarrow{T^{kt}} \mathbf{v} \xrightarrow{B} \mathbf{y}$. *Then* $\mathbf{u} = \mathbf{v} = \begin{bmatrix} d \\ 0 \end{bmatrix}$*, where* $|d| = \gcd(x_1, x_2) = \gcd(y_1, y_2)$.

Proof. In the new notations, the equation $BT^{kt}C\mathbf{x} = \mathbf{y}$ can be written as $T^{kt}\mathbf{u} = \mathbf{v}$, and this equality holds for any $t \in \mathbb{Z}$. Now let $\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$. Hence we have $\begin{bmatrix} 1 & kt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, which is equivalent to $u_2 = v_2$ and $u_1 + ktu_2 = v_1$, for any $t \in \mathbb{Z}$. So, we must have $u_2 = v_2 = 0$ and hence $u_1 = v_1$.

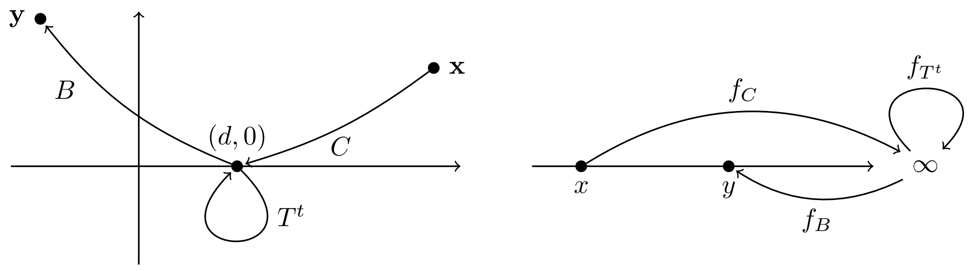


Figure 1 Geometric interpretation of the linear transformation $\mathbf{y} = BT^tC\mathbf{x}$ (left) and of the fractional linear transformation $y = f_{BT^tC}(x)$ (right).

Therefore, the vectors $\mathbf{u}$ and $\mathbf{v}$ have the form $\mathbf{u} = \mathbf{v} = \begin{bmatrix} d \\ 0 \end{bmatrix}$ for some $d \in \mathbb{Z}$. Moreover, since $\mathbf{u} = C\mathbf{x}$, we obtain from Lemma 4 that $|d| = \gcd(x_1, x_2) = \gcd(y_1, y_2)$. ◀

We can give the following geometric interpretation of the transformation $BT^tC\mathbf{x} = \mathbf{y}$: first, we apply C to $\mathbf{x}$ and arrive at $\mathbf{u} = \begin{bmatrix} d \\ 0 \end{bmatrix}$, then we loop at $\mathbf{u}$ for t many times using T, and finally apply B to move from $\mathbf{u}$ to $\mathbf{y}$ (see Figure 1 on the left).

Similarly, we have the following geometric interpretation of the fractional linear transformation $y = f_{BT^tC}(x) = f_B \circ f_{T^t} \circ f_C(x)$: first it maps x to ∞ using f_C, then loops at ∞ for t many times using f_T, and finally maps ∞ to y using f_B (see Figure 1 on the right).

We now show how to apply the geometric interpretation of the vector reachability problem to solve a special case of the scalar reachability problem.

▶ **Definition 17.** The *scalar reachability problem* in $\mathrm{SL}(2,\mathbb{Z})$ is stated as follows: Let $[z_1, z_2]$ and $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ be vectors from $\mathbb{Z} \times \mathbb{Z}$ and let λ be an integer number. We are also given a finite collection of matrices $M_1, \dots, M_n$ from $\mathrm{SL}(2,\mathbb{Z})$. The question is to decide whether there exists a matrix $M \in \langle M_1, \dots, M_n \rangle$ which satisfies the equation $[z_1, z_2]\, M \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda$.

We will consider a special case of this problem when $z_2 = 1$ and $\lambda = 1$. Our proof relies on the characterization from Theorem 8 and Corollary 9 and on Proposition 13 in which we showed that the emptiness problem for the intersection of two regular subsets in $\mathrm{SL}(2,\mathbb{Z})$ is decidable.

▶ **Proposition 18.** *Suppose that the above equation has the form*

$$[a, 1]\, M \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 1, \tag{3}$$

where a, x_1 and x_2 are some integer numbers. Then this special case of the scalar reachability problem is decidable.

Proof. The general idea of the proof is the same as in Theorem 14, that is, we will show that the set of matrices $M \in \mathrm{SL}(2,\mathbb{Z})$ that satisfy Equation (3) can be described by a regular language. First, let us consider a geometric interpretation of this problem. We can rewrite Equation (3) as a system of two equations: $M \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ and $ay_1 + y_2 = 1$. So, M satisfies Equation (3) if and only if it maps a fixed vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ to some vector $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ that lies

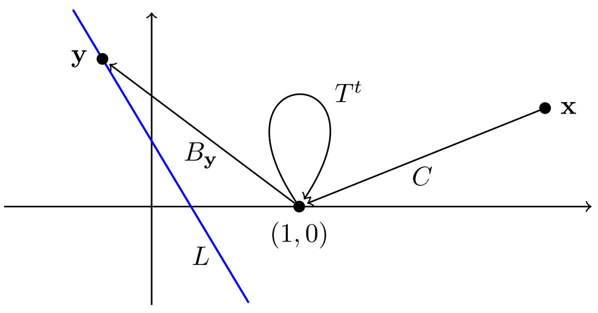


Figure 2 Geometric interpretation of the scalar reachability problem.

on the line L described by the equation $ay_1 + y_2 = 1$. In other words, we have a *vector to line reachability problem* for the line L that is defined by the equation $ay_1 + y_2 = 1$.

Note that if a vector $\mathbf{y}$ lies of the line $ay_1 + y_2 = 1$, then $\gcd(y_1, y_2) = 1$. Hence by Lemma 4, Equation (3) has a solution only if $\gcd(x_1, x_2) = 1$. So, from now on we assume that $\gcd(x_1, x_2) = 1$.

By Corollary 9, any $M \in \mathrm{SL}(2,\mathbb{Z})$ that maps $\mathbf{x}$ to a vector $\mathbf{y}$ on the line L has the form $M = BT^tC$, where B and C are some matrices from $\mathrm{SL}(2,\mathbb{Z})$ and $t \in \mathbb{Z}$. Geometrically, the transformation $\mathbf{y} = BT^tC\mathbf{x}$ goes via the point $(1,0)$ as shown in Figure 2.

Note that the matrices B and C above depend on the vector $\mathbf{y}$ as a parameter. Here we prove a useful lemma which will imply that we can choose only one matrix C that maps $\mathbf{x}$ to $\begin{bmatrix}1\\0\end{bmatrix}$ independently of the vector $\mathbf{y}$.

▶ **Lemma 19.** *Let* $\mathbf{x} = \begin{bmatrix}x_1\\x_2\end{bmatrix}$ *and* $\mathbf{y} = \begin{bmatrix}y_1\\y_2\end{bmatrix}$ *be any vectors from* $\mathbb{Z}\times\mathbb{Z}$ *such that* $\gcd(x_1, x_2) = \gcd(y_1, y_2) = d$. *Let* d_1 *and* d_2 *be any integer numbers with* $|d_1| = |d_2| = d$ *and let* A_1, B_1 *and* A_2, B_2 *by any matrices from* $\mathrm{SL}(2,\mathbb{Z})$ *such that* $B_i\mathbf{x} = \begin{bmatrix}d_i\\0\end{bmatrix}$ *and* $A_i\begin{bmatrix}d_i\\0\end{bmatrix} = \mathbf{y}$, *for* $i = 1, 2$. *Then* $\{A_1T^tB_1 \ : \ t \in \mathbb{Z}\} = \{A_2T^tB_2 \ : \ t \in \mathbb{Z}\}$. *In other words, the following diagrams define the same set of matrices that map* $\mathbf{x}$ *to* $\mathbf{y}$.

$$\begin{tikzcd} \mathbf{x} \arrow[r, "B_1"] & \left[\begin{smallmatrix}d_1\\0\end{smallmatrix}\right] \arrow[r, "A_1"] \arrow[loop below, "T^t"] & \mathbf{y} \end{tikzcd} \qquad \begin{tikzcd} \mathbf{x} \arrow[r, "B_2"] & \left[\begin{smallmatrix}d_2\\0\end{smallmatrix}\right] \arrow[r, "A_2"] \arrow[loop below, "T^t"] & \mathbf{y} \end{tikzcd}$$

Proof. See Section C of the Appendix in [31]. ◀

By Lemma 19, we can choose any matrix C from $\mathrm{SL}(2,\mathbb{Z})$ that maps a vector $\mathbf{x}$ to the vector $\begin{bmatrix}1\\0\end{bmatrix}$, and for each $\mathbf{y}$ on the line L we can choose any matrix $B_{\mathbf{y}}$ that maps $\begin{bmatrix}1\\0\end{bmatrix}$ to the vector $\mathbf{y}$. Then the solution of Equation (3) will be described by the following set $\mathcal{F} = \{B_{\mathbf{y}}T^tC \ : \ \mathbf{y} \in L \text{ and } t \in \mathbb{Z}\}$. Figure 2 gives geometric interpretation of this solution.

We need to choose $B_{\mathbf{y}}$ in such a way that $\mathcal{F}$ becomes a regular set. Let $\mathbf{y} = \begin{bmatrix}y_1\\y_2\end{bmatrix} \in L$, then we have $ay_1 + y_2 = 1$. As one can check, if we let $B_{\mathbf{y}} = \begin{bmatrix}y_1 & -1\\-ay_1+1 & a\end{bmatrix}$ then $B_{\mathbf{y}} \in \mathrm{SL}(2,\mathbb{Z})$ and $B_{\mathbf{y}}\begin{bmatrix}1\\0\end{bmatrix} = \mathbf{y}$.

Since every entry of $B_{\mathbf{y}}$ is a linear function of y_1, we obtain by Proposition 7 that $B_{\mathbf{y}} = AT^{ky_1}D$, where A and D are some matrices from $\mathrm{SL}(2,\mathbb{Z})$ and k is some integer number (in fact, one can show that $k = 1$). Finally, we can write all solutions of Equation (3) as $\mathcal{F} = \{AT^{ky_1}DT^tC \ : \ y_1 \in \mathbb{Z} \text{ and } t \in \mathbb{Z}\}$. This is clearly a regular set and, therefore, the scalar reachability problem is decidable. ◀

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*, SODA'96, pages 498–507, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

2 Paul Bell, Vesa Halava, Tero Harju, Juhani Karhumäki, and Igor Potapov. Matrix equations and Hilbert's tenth problem. *Internat. J. Algebra Comput.*, 18(8):1231–1241, 2008.

3 Paul Bell and Igor Potapov. On undecidability bounds for matrix decision problems. *Theoretical Computer Science*, 391(1-2):3–13, 2008.

4 Paul Bell and Igor Potapov. Reachability problems in quaternion matrix and rotation semigroups. *Information and Computation*, 206(11):1353–1361, 2008.

5 Paul C. Bell, Mika Hirvensalo, and Igor Potapov. Mortality for 2x2 matrices is NP-hard. In Branislav Rovan, Vladimiro Sassone, and Peter Widmayer, editors, *Mathematical Foundations of Computer Science 2012*, volume 7464 of *Lecture Notes in Computer Science*, pages 148–159. Springer Berlin Heidelberg, 2012.

6 Paul C. Bell and Igor Potapov. On the computational complexity of matrix semigroup problems. *Fundam. Inf.*, 116(1-4):1–13, January 2012.

7 Vincent D. Blondel, Emmanuel Jeandel, Pascal Koiran, and Natacha Portier. Decidable and undecidable problems about quantum automata. *SIAM J. Comput.*, 34(6):1464–1473, June 2005.

8 Julien Cassaigne, Tero Harju, and Juhani Karhumaki. On the undecidability of freeness of matrix semigroups. *International Journal of Algebra and Computation*, 09(03n04):295–305, 1999. `doi:10.1142/S0218196799000199`.

9 Julien Cassaigne and François Nicolas. On the decidability of semigroup freeness. *RAIRO – Theor. Inf. and Applic.*, 46(3):355–399, 2012.

10 Fernando Chamizo. Non-euclidean visibility problems. *Proceedings of the Indian Academy of Sciences – Mathematical Sciences*, 116(2):147–160, 2006.

11 Christian Choffrut and Juhani Karhumaki. Some decision problems on integer matrices. *RAIRO-Theor. Inf. Appl.*, 39(1):125–131, 2005.

12 J. Elstrodt, F. Grunewald, and J. Mennicke. Arithmetic applications of the hyperbolic lattice point theorem. *Proc. London Math. Soc.*, s3-57:pp.239–283, 1988.

13 J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Logic in Computer Science, 1999. Proceedings. 14th Symposium on*, pages 352–359, 1999.

14 Esther Galby, Joël Ouaknine, and James Worrell. On Matrix Powering in Low Dimensions. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 329–340, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

15 M. P. García del Moral, I. Martín, J. M. Peña, and A. Restuccia. Sl(2,z) symmetries, supermembranes and symplectic torus bundles. *Journal of High Energy Physics*, 2011(9):1–12, 2011.

16 Yuri Gurevich and Paul Schupp. Membership problem for the modular group. *SIAM J. Comput.*, 37(2):425–459, May 2007.

17 Vesa Halava, Tero Harju, and Mika Hirvensalo. Undecidability bounds for integer matrices using Claus instances. *International Journal of Foundations of Computer Science*, 18(05):931–948, 2007. `doi:10.1142/S0129054107005066`.

18 Vesa Halava, Tero Harju, Mika Hirvensalo, and Juhani Karhumaki. Skolem's problem – on the border between decidability and undecidability. Technical Report 683, Turku Centre for Computer Science, 2005.

19 Ravindran Kannan and Achim Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM J. Comput.*, 8(4):499–507, 1979.

20 Alexei Lisitsa and Igor Potapov. Membership and reachability problems for row-monomial transformations. In *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, pages 623–634, 2004.

21 Roger C. Lyndon and Paul E. Schupp. *Combinatorial group theory*. Springer-Verlag, Berlin-New York, 1977. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 89.

22 Dana Mackenzie. A new twist in knot theory. *What's Happening in the Mathematical Sciences*, Volume 7, 2009.

23 Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial group theory*. Dover Publications, Inc., New York, revised edition, 1976. Presentations of groups in terms of generators and relations.

24 A. Markov. On certain insoluble problems concerning matrices. *Doklady Akad. Nauk SSSR*, 57(6):539–542, June 1947.

25 Thomas Noll. Musical intervals and special linear transformations. *Journal of Mathematics and Music: Mathematical and Computational Approaches to Music Theory, Analysis, Composition and Performance*, vol.1, 2007.

26 Joël Ouaknine, João Sousa Pinto, and James Worrell. On termination of integer linear loops. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'15, pages 957–969. SIAM, 2015.

27 Joël Ouaknine and James Worrell. On the positivity problem for simple linear recurrence sequences,. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 318–329, 2014.

28 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 330–341, 2014.

29 Leonid Polterovich and Zeev Rudnick. Stable mixing for cat maps and quasi-morphisms of the modular group. *Ergodic Theory and Dynamical Systems*, 24:609–619, 4 2004.

30 Igor Potapov. Composition problems for braids. In *33nd International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 24 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 175–187. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2013.

31 Igor Potapov and Pavel Semukhin. Vector reachability problem in $\mathrm{SL}(2,\mathbb{Z})$. *CoRR*, abs/1510.03227, 2015. URL: `http://arxiv.org/abs/1510.03227`.

32 Robert A. Rankin. *Modular forms and functions*. Cambridge University Press, Cambridge-New York-Melbourne, 1977.

33 Edward Witten. $\mathrm{SL}(2,\mathbb{Z})$ action on three-dimensional conformal field theories with abelian symmetry. In *From fields to strings: circumnavigating theoretical physics. Vol. 2*, pages 1173–1200. World Sci. Publ., Singapore, 2005.

34 Don Zagier. Elliptic modular forms and their applications. In Kristian Ranestad, editor, *The 1-2-3 of Modular Forms*, Universitext, pages 1–103. Springer Berlin Heidelberg, 2008.

The Generalised Colouring Numbers on Classes of Bounded Expansion[*]

Stephan Kreutzer[1], Michał Pilipczuk[2], Roman Rabinovich[3], and Sebastian Siebertz[4]

1 **Technische Universität Berlin, Berlin, Germany**
`stephan.kreutzer@tu-berlin.de`
2 **Institute of Informatics, University of Warsaw, Warsaw, Poland**
`michal.pilipczuk@mimuw.edu.pl`
3 **Technische Universität Berlin, Berlin, Germany**
`roman.rabinovich@tu-berlin.de`
4 **Technische Universität Berlin, Berlin, Germany**
`sebastian.siebertz@tu-berlin.de`

Abstract

The generalised colouring numbers $\mathrm{adm}_r(G)$, $\mathrm{col}_r(G)$, and $\mathrm{wcol}_r(G)$ were introduced by Kierstead and Yang as generalisations of the usual colouring number, also known as the degeneracy of a graph, and have since then found important applications in the theory of bounded expansion and nowhere dense classes of graphs, introduced by Nešetřil and Ossona de Mendez. In this paper, we study the relation of the colouring numbers with two other measures that characterise nowhere dense classes of graphs, namely with uniform quasi-wideness, studied first by Dawar et al. in the context of preservation theorems for first-order logic, and with the splitter game, introduced by Grohe et al. We show that every graph excluding a fixed topological minor admits a universal order, that is, one order witnessing that the colouring numbers are small for every value of r. Finally, we use our construction of such orders to give a new proof of a result of Eickmeyer and Kawarabayashi, showing that the model-checking problem for successor-invariant first-order formulas is fixed parameter tractable on classes of graphs with excluded topological minors.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory, F.4.1 Mathematical Logic (Model Theory)

Keywords and phrases Graph Structure Theory, Nowhere Dense Graphs, Generalised Colouring Numbers, Splitter Game, First-Order Model-Checking

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.85

1 Introduction

The *colouring number* $\mathrm{col}(G)$ of a graph G is the minimum k for which there is a linear order $<_L$ on the vertices of G such that each vertex v has *back-degree* at most $k-1$, that is, v has at most $k-1$ neighbours u with $u <_L v$. The colouring number is a measure for uniform sparseness in graphs: we have $\mathrm{col}(G) = k$ if and only if every subgraph H of G has a vertex

[*] This work was initiated during Sebastian Siebertz's visit at the Institute of Informatics of the University of Warsaw, which was supported by the Warsaw Centre of Mathematics and Computer Science. Michał Pilipczuk is supported by the Foundation for Polish Science (FNP) via the START stipend programme. Stephan Kreutzer, Roman Rabinovich and Sebastian Siebertz's research has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC Consolidator Grant DISTRUCT, grant agreement No 648527).

of degree at most $k-1$. Hence, provided $\mathrm{col}(G)=k$, not only G is sparse, but also every subgraph of G is sparse. The colouring number minus one is also known as the *degeneracy*.

Recently, Nešetřil and Ossona de Mendez introduced the notions of *bounded expansion* [12] and *nowhere density* [14] as very general formalisations of uniform sparseness in graphs. Since then, several independent and seemingly unrelated characterisations of these notions have been found, showing that these concepts behave robustly. For example, nowhere dense classes of graphs can be defined in terms of excluded shallow minors [14], in terms of uniform quasi-wideness [2], a notion studied in model theory, or in terms of a game [8] with direct algorithmic applications. The *generalised colouring numbers* adm_r, col_r, and wcol_r were introduced by Kierstead and Yang [11] in the context of colouring and marking games on graphs. As proved by Zhu [17], they can be used to characterise both bounded expansion and nowhere dense classes of graphs.

The invariants adm_r, col_r, and wcol_r are defined similarly to the classic colouring number: for example, the *weak r-colouring* number $\mathrm{wcol}_r(G)$ of a graph G is the minimum integer k for which there is a linear order of the vertices such that each vertex v can reach at most $k-1$ vertices w by a path of length at most r in which w is the smallest vertex on the path.

The generalised colouring numbers found important applications in the context of algorithmic theory of sparse graphs. For example, they play a key role in Dvořák's approximation algorithm for minimum dominating sets [4], or in the construction of sparse neighbourhood covers on nowhere dense classes, a fundamental step in the almost linear time model-checking algorithm for first-order formulas of Grohe et al. [8].

In this paper we study the relation between the colouring numbers and the above mentioned characterisations of nowhere dense classes of graphs, namely with uniform quasi-wideness and the splitter game. We use the generalised colouring numbers to give a new proof that every bounded expansion class is uniformly quasi-wide. This was first proved by Nešetřil and Ossona de Mendez in [13]; however, the constants appearing in the proof of [13] are huge. We present a very simple proof which also improves the appearing constants. Furthermore, for the splitter game introduced in [8], we show that splitter has a very simple strategy to win on any class of bounded expansion, which leads to victory much faster than in general nowhere dense classes of graphs.

Every graph G from a fixed class $\mathcal{C}$ of bounded expansion satisfies $\mathrm{wcol}_r(G) \leq f(r)$ for some function f and all positive integers r. However, the order that witnesses this inequality for G may depend on the value r. We say that a class $\mathcal{C}$ admits *uniform orders* if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for each $G \in \mathcal{C}$ there is one linear order that witnesses $\mathrm{wcol}_r(G) \leq f(r)$ for every value of r. We show that every class that excludes a fixed topological minor admits uniform orders that can be computed efficiently.

Finally, based on our construction of uniform orders for graphs that exclude a fixed topological minor, we provide an alternative proof of a very recent result of Eickmeyer and Kawarabayashi [6], that the model-checking problem for successor-invariant first-order (FO) formulas is fixed-parameter tractable on such classes (we obtained this result independently of, but later than, [6]). Successor-invariant logics have been studied in database theory and finite model theory, and successor-invariant FO is known to be more expressive than plain FO [15]. The model-checking problem for successor-invariant FO is known to be fixed-parameter tractable parameterized by the size of the formula on any graph class that excludes a fixed minor [7]. Very recently, this result was lifted to classes that exclude a fixed topological minor by Eickmeyer and Kawarabayashi [6]. The key point of their proof is to use the decomposition theorem for graphs excluding a fixed topological minor, due to Grohe and Marx [9]. Our approach is similar to that of [6]. However, we employ new constructions based on the generalised colouring numbers and use the decomposition theorem of [9] only implicitly. In particular, we do not construct a graph decomposition in order to solve the model-checking problem. Therefore, we believe that our approach may be easier to extend further to classes of bounded expansion, or even to nowhere dense classes of graphs.

2 Preliminaries

Notation. We use standard graph-theoretical notation; see e.g. [3] for reference. All graphs considered in this paper are finite, simple, and undirected. For a graph G, by $V(G)$ and $E(G)$ we denote the vertex and edge sets of G, respectively. A graph H is a *subgraph* of G, denoted $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For any $M \subseteq V(G)$, by $G[M]$ we denote the subgraph induced by M. We write $G - M$ for the graph $G[V(G) \setminus M]$ and if $M = \{v\}$, we write $G - v$ for $G - M$. For a non-negative integer ℓ, a *path of length* ℓ in G is a sequence $P = (v_1, \ldots, v_{\ell+1})$ of pairwise different vertices such that $v_iv_{i+1} \in E(G)$ for all $1 \leq i \leq \ell$. We write $V(P)$ for the vertex set $\{v_1, \ldots, v_{\ell+1}\}$ of P and $E(P)$ for the edge set $\{v_iv_{i+1} : 1 \leq i \leq \ell\}$ of P and identify P with the subgraph of G with vertex set $V(P)$ and edge set $E(P)$. We say that the path P *connects* its *endpoints* $v_1, v_{\ell+1}$, whereas $v_2, \ldots, v_\ell$ are the *internal vertices* of P. The *length* of a path is the number of its edges. Two vertices $u, v \in V(G)$ are *connected* if there is a path in G with endpoints u, v. The *distance* $\mathrm{dist}(u, v)$ between two connected vertices u, v is the minimum length of a path connecting u and v; if u, v are not connected, we put $\mathrm{dist}(u, v) = \infty$. The *radius* of G is $\min_{u\in V(G)} \max_{v\in V(G)} \mathrm{dist}(u, v)$. The set of all neighbours of a vertex v in G is denoted by $N^G(v)$, and the set of all vertices at distance at most r from v is denoted by $N_r^G(v)$. A graph G is c-*degenerate* if every subgraph $H \subseteq G$ has a vertex of degree at most c. A c-degenerate graph of order n contains an independent set of order at least $n/(c + 1)$.

A graph H with $V(H) = \{v_1, \ldots, v_n\}$ is a *minor* of G, written $H \preccurlyeq G$, if there are pairwise disjoint connected subgraphs $H_1, \ldots, H_n$ of G, called *branch sets*, such that whenever $v_iv_j \in E(H)$, then there are $u_i \in H_i$ and $u_j \in H_j$ with $u_iu_j \in E(G)$. We call $(H_1, \ldots, H_n)$ a *minor model* of H in G. The graph H is a *topological minor* of G, written $H \preccurlyeq^t G$, if there are pairwise different vertices $u_1, \ldots, u_n \in V(G)$ and a family of paths $\{P_{ij} : v_iv_j \in E(H)\}$, such that each P_{ij} connects u_i and u_j, and paths P_{ij} are pairwise internally vertex-disjoint.

Generalised colouring numbers. Let us fix a graph G. By $\Pi(G)$ we denote the set of all linear orders of $V(G)$. For $L \in \Pi(G)$, we write $u <_L v$ if u is smaller than v in L, and $u \leq_L v$ if $u <_L v$ or $u = v$. Let $u, v \in V(G)$. For a non-negative integer r, we say that u is *weakly* r-*reachable* from v with respect to L, if there is a path P of length ℓ, $0 \leq \ell \leq r$, connecting u and v such that u is minimum among the vertices of P (with respect to L). By $\mathrm{WReach}_r[G, L, v]$ we denote the set of vertices that are weakly r-reachable from v w.r.t. L.

Vertex u is *strongly* r-*reachable* from v with respect to L, if there is a path P of length ℓ, $0 \leq \ell \leq r$, connecting u and v such that $u \leq_L v$ and such that all internal vertices w of P satisfy $v <_L w$. Let $\mathrm{SReach}_r[G, L, v]$ be the set of vertices that are strongly r-reachable from v w.r.t. L. Note that we have $v \in \mathrm{SReach}_r[G, L, v] \subseteq \mathrm{WReach}_r[G, L, v]$.

For a non-negative integer r, we define the *weak* r-*colouring number* $\mathrm{wcol}_r(G)$ of G and the r-*colouring number* $\mathrm{col}_r(G)$ of G respectively as follows:

$$\begin{aligned} \mathrm{wcol}_r(G) &:= \min_{L\in\Pi(G)} \max_{v\in V(G)} \big|\mathrm{WReach}_r[G, L, v]\big|, \\ \mathrm{col}_r(G) &:= \min_{L\in\Pi(G)} \max_{v\in V(G)} \big|\mathrm{SReach}_r[G, L, v]\big|. \end{aligned}$$

For a non-negative integer r, the *r-admissibility* $\mathrm{adm}_r[G, L, v]$ of v w.r.t. L is the maximum size k of a family $\{P_1, \ldots, P_k\}$ of paths of length at most r that start in v, end at a vertex w with $w \leq_L v$, and satisfy $V(P_i) \cap V(P_j) = \{v\}$ for all $1 \leq i < j \leq k$. As for $r > 0$ we can always let the paths end in the first vertex smaller than v, we can assume that the internal vertices of the paths are larger than v. Note that $\mathrm{adm}_r[G, L, v]$ is an integer, whereas $\mathrm{WReach}_r[G, L, v]$ and $\mathrm{SReach}_r[G, L, v]$ are vertex sets. The *r-admissibility* $\mathrm{adm}_r(G)$ of G is

$$\mathrm{adm}_r(G) \quad = \quad \min_{L \in \Pi(G)} \max_{v \in V(G)} \mathrm{adm}_r[G, L, v].$$

The generalised colouring numbers were introduced by Kierstead and Yang [11] in the context of colouring and marking games on graphs. The authors also proved that the generalised colouring numbers are related by the following inequalities:

$$\mathrm{adm}_r(G) \leq \mathrm{col}_r(G) \leq \mathrm{wcol}_r(G) \leq (\mathrm{adm}_r(G))^r. \tag{1}$$

Shallow minors, bounded expansion, and nowhere denseness. A graph H with $V(H) = \{v_1, \ldots, v_n\}$ is a *depth-r minor* of G, denoted $H \preccurlyeq_r G$, if there is a minor model $(H_1, \ldots, H_n)$ of H in G such that each H_i has radius at most r. We write $d(H)$ for the *average degree* of H, that is, for the number $2|E(H)|/|V(H)|$. A class $\mathcal{C}$ of graphs has *bounded expansion* if there is a function $f\colon \mathbb{N} \to \mathbb{N}$ such that for all non-negative integers r we have $d(H) \leq f(r)$ for every $H \preccurlyeq_r G$ with $G \in \mathcal{C}$. A class $\mathcal{C}$ of graphs is *nowhere dense* if for every real $\epsilon > 0$ and every non-negative integer r, there is an integer n_0 such that if H is an n-vertex graph with $n \geq n_0$ and $H \preccurlyeq_r G$ for some $G \in \mathcal{C}$, then $d(H) \leq n^\epsilon$.

Bounded expansion and nowhere dense classes of graphs were introduced by Nešetřil and Ossona de Mendez as models for uniform sparseness of graphs [12, 14]. As proved by Zhu [17], the generalised colouring numbers are tightly related to densities of low-depth minors, and hence they can be used to characterise bounded expansion and nowhere dense classes.

▶ **Theorem 1** (Zhu [17]). *A class $\mathcal{C}$ of graphs has bounded expansion if and only if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{wcol}_r(G) \leq f(r)$ for all $r \in \mathbb{N}$ and all $G \in \mathcal{C}$.*

Due to (1), we may equivalently demand that there is a function $f : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{adm}_r(G) \leq f(r)$ or $\mathrm{col}_r(G) \leq f(r)$ for all non-negative integers r and all $G \in \mathcal{C}$.

Similarly, from Zhu's result one can derive a characterisation of nowhere dense classes of graphs, as presented in [14]. A class $\mathcal{C}$ of graphs is called *hereditary* if it is closed under induced subgraphs, that is, if H is an induced subgraph of $G \in \mathcal{C}$, then $H \in \mathcal{C}$.

▶ **Theorem 2** (Nešetřil and Ossona de Mendez [14]). *A hereditary class $\mathcal{C}$ of graphs is nowhere dense if and only if for every real $\epsilon > 0$ and every non-negative integer r, there is a positive integer n_0 such that if $G \in \mathcal{C}$ is an n-vertex graph with $n \geq n_0$, then $\mathrm{wcol}_r(G) \leq n^\epsilon$.*

As shown in [4], for every non-negative integer r, computing $\mathrm{adm}_r(G)$ is fixed-parameter tractable on any class of bounded expansion (parameterized by $\mathrm{adm}_r(G)$). For $\mathrm{col}_r(G)$ and $\mathrm{wcol}_r(G)$ this is not known; however, by (1) we can use admissibility to obtain approximations of these numbers. On nowhere dense classes of graphs, for every $\epsilon > 0$ and every non-negative integer r, we can compute an order that witnesses $\mathrm{wcol}_r(G) \leq n^\epsilon$ in time $\mathcal{O}(n^{1+\epsilon})$ if G is sufficiently large [8], based on Nešetřil and Ossona de Mendez's augmentation technique [12].

3 Uniform quasi-wideness and the splitter game

In this section we discuss the relation between weak r-colouring numbers and two notions that characterise nowhere dense classes: uniform quasi-wideness and the splitter game.

For a graph G, a vertex subset $A \subseteq V(G)$ is called *r-independent* in G, if $\mathrm{dist}_G(a,b) > r$ for all different $a, b \in V(G)$. A vertex subset is called *r-scattered*, if it is $2r$-independent, that is, if the r-neighbourhoods of different elements of A do not intersect.

Informally, uniform quasi-wideness means the following: in any large enough subset of vertices of a graph from $\mathcal{C}$, one can find a large subset that is r-scattered in G, possibly after removing from G a small number of vertices. Formally, a class $\mathcal{C}$ of graphs is *uniformly quasi-wide* if there are functions $N : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $s : \mathbb{N} \to \mathbb{N}$ such that for all $m, r \in \mathbb{N}$, if $W \subseteq V(G)$ for a graph $G \in \mathcal{C}$ with $|W| > N(m,r)$, then there is a set $S \subseteq V(G)$ of size at most $s(r)$ such that W contains a subset of size at least m that is r-scattered in $G - S$.

The notion of quasi-wideness was introduced by Dawar [2] in the context of homomorphism preservation theorems. It was shown in [13] that classes of bounded expansion are uniformly quasi-wide and that uniform quasi-wideness characterises nowhere dense classes of graphs.

▶ **Theorem 3** (Nešetřil and Ossona de Mendez [13])**.** *A hereditary class $\mathcal{C}$ of graphs is nowhere dense if and only if it is uniformly quasi-wide.*

It was shown by Atserias et al. in [1] that classes that exclude K_k as a minor are uniformly quasi-wide. In fact, in this case we can choose $s(r) = k - 1$, independent of r (if such a constant function for a class $\mathcal{C}$ exists, the class is called *uniformly almost wide*). However, the function $N(m,r)$ that was used in the proof is huge: it comes from an iterated Ramsey argument. The same approach was used in [13] to show that every nowhere dense class, and in particular, every class of bounded expansion, is uniformly quasi-wide. We present a new proof that every bounded expansion class is uniformly quasi-wide, which gives us a much better bound on $N(m,r)$ and which is much simpler than the previously known proof.

▶ **Theorem 4.** *Let G be a graph and let $r, m \in \mathbb{N}$. Let $c \in \mathbb{N}$ be such that $\mathrm{wcol}_r(G) \le c$ and let $A \subseteq V(G)$ be a set of size at least $(c+1) \cdot 2^m$. Then there exists a set S of size at most $c(c-1)$ and a set $B \subseteq A$ of size at least m which is r-independent in $G - S$.*

Proof. Let $L \in \Pi(G)$ be such that $|\mathrm{WReach}_r[G,L,v]| \le c$ for every $v \in V(G)$. Let H be the graph with vertex set $V(G)$, where we put an edge $uv \in E(H)$ if and only if $u \in \mathrm{WReach}_r[G,L,v]$ or $v \in \mathrm{WReach}_r[G,L,u]$. Then L certifies that H is c-degenerate, and hence we can greedily find an independent set $I \subseteq A$ of size 2^m in H. By the definition of the graph H, we have that $\mathrm{WReach}_r[G,L,v] \cap I = \{v\}$ for each $v \in I$.

▶ **Claim 5.** *Let $v \in I$. Then deleting $\mathrm{WReach}_r[G,L,v] \setminus \{v\}$ from G leaves v at a distance greater than r (in $G - (\mathrm{WReach}_r[G,L,v] \setminus \{v\})$) from all the other vertices of I.*

Proof. Let $u \in I$ and let P be a path in G that has length at most r and connects u and v. Let $z \in V(P)$ be minimal with respect to L. Then $z <_L v$ or $z = v$. If $z <_L v$, then $z \in \mathrm{WReach}_r[G,L,v]$ and hence the path P no longer exists after the deletion of $\mathrm{WReach}_r[G,L,v] \setminus \{v\}$ from G. On the other hand, if $z = v$, then $v \in \mathrm{WReach}_r[G,L,u]$, contradicting the fact that both $u, v \in I$. ⌟

We iteratively find sets $B_0 \subseteq \ldots \subseteq B_m \subseteq I$, sets $I_0 \supseteq \ldots \supseteq I_m$, and sets $S_0 \subseteq \ldots \subseteq S_m$ such that B is r-independent in $G - S$, where $B := B_m$ and $S := S_m$. We maintain the invariant that sets B_i, I_i, and S_i are pairwise disjoint for each i. Let $I_0 = I$, $B_0 = \emptyset$ and $S_0 = \emptyset$. In one step $i = 1, 2, \ldots, m$, we delete some vertices from I_i (thus obtaining I_{i+1}), shift one vertex from I_i to B_i (obtaining B_{i+1}) and, possibly, add some vertices from $V(G) \setminus I_i$ to S_i (obtaining S_{i+1}). More precisely, let v be the vertex of I_i that is the largest in the order L. We set $B_{i+1} = B_i \cup \{v\}$, and now we discuss how I_{i+1} and S_{i+1} are constructed.

We distinguish two cases. First, suppose v is connected by a path of length at most r in $G - S_i$ to at most half of the vertices of I_i (including v). Then we remove these reachable vertices from I_i, and set I_{i+1} to be the result. We also set $S_{i+1} = S_i$. Note that $|I_{i+1}| \geq |I_i|/2$.

Second, suppose v is connected by a path of length at most r in $G - S_i$ to more than half of the vertices of I_i (including v). We proceed in two steps. First, we add the at most $c - 1$ vertices of $\mathrm{WReach}_r[G, L, v] \setminus \{v\}$ to S_{i+1}, that is, we let $S_{i+1} = S_i \cup (\mathrm{WReach}_r[G, L, v] \setminus \{v\})$. (Recall here that $\mathrm{WReach}_r[G, L, v] \cap I = \{v\}$.) By Claim 5, this leaves v at a distance greater than r from every other vertex of I_i in $G - S_{i+1}$. Second, we construct I_{i+1} from I_i by removing the vertex v and all the vertices of I_i that are not connected to v by a path of length at most r in $G - S_i$, hence we have $|I_{i+1}| \geq \lfloor |I_i|/2 \rfloor$.

Observe the construction above can be carried out for m steps, because in each step, we remove at most half of the vertices of I_i (rounded up) when constructing I_{i+1}. As $|I_0| = |I| = 2^m$, it is easy to see that the set I_i cannot become empty within m iterations. Moreover, it is clear from the construction that we end up with a set $B = B_m$ that has size m and is r-scattered in $G - S$, where $S = S_m$. It remains to argue that $|S_m| \leq c(c - 1)$. For this, it suffices to show that the second case cannot apply more than c times in total.

Suppose the second case was applied in the ith iteration, when considering a vertex v. Every vertex $u \in I_i$ with $u <_L v$ that was connected to v by a path of length at most r in $G - S_i$ satisfies $\mathrm{WReach}_r[G, L, v] \cap \mathrm{WReach}_r[G, L, u] \neq \emptyset$. Thus, every remaining vertex $u \in I_{i+1}$ has at least one of its weakly r-reachable vertices deleted (that is, included in S_{i+1}). As the number of such vertices is at most $c - 1$ at the beginning, and it can only decrease during the construction, this implies that the second case can occur at most c times. ◀

As shown in [16], if $K_k \not\preccurlyeq G$, then $\mathrm{wcol}_r(G) \in \mathcal{O}(r^{k-1})$. Hence, for such graphs we have to delete only a polynomial (in r) number of vertices in order to find an r-independent set of size m in a set of vertices of size single exponential in m.

We now implement the same idea to find a very simple strategy for splitter in the splitter game, introduced by Grohe et al. [8] to characterise nowhere dense classes of graphs. Let $\ell, r \in \mathbb{N}$. The *simple ℓ-round radius-r splitter game* on G is played by two players, *connector* and *splitter*, as follows. We let $G_0 := G$. In round $i + 1$ of the game, connector chooses a vertex $v_{i+1} \in V(G_i)$. Then splitter picks a vertex $w_{i+1} \in N_r^{G_i}(v_{i+1})$. We let $G_{i+1} := G_i[N_r^{G_i}(v_{i+1}) \setminus \{w_{i+1}\}]$. Splitter wins if $G_{i+1} = \emptyset$. Otherwise the game continues at G_{i+1}. If splitter has not won after ℓ rounds, then connector wins.

A *strategy* for splitter is a function σ that maps every partial play $(v_1, w_1, \ldots, v_s, w_s)$, with associated sequence $G_0, \ldots, G_s$ of graphs, and the next move $v_{s+1} \in V(G_s)$ of connector, to a vertex $w_{s+1} \in N_r^{G_s}(v_{s+1})$ that is the next move of splitter. A strategy σ is a *winning strategy* for splitter if splitter wins every play in which she follows the strategy f. We say that splitter *wins* the simple ℓ-round radius-r splitter game on G if she has a winning strategy.

▶ **Theorem 6** (Grohe et al. [8]). *A class $\mathcal{C}$ of graphs is nowhere dense if and only if there is a function $\ell : \mathbb{N} \to \mathbb{N}$ such that splitter wins the simple $\ell(r)$-round radius-r splitter game on every graph $G \in \mathcal{C}$.*

More precisely, it was shown in [8] that $\ell(r)$ can be chosen as $N(2s(r), r)$, where N and s are the functions that characterise $\mathcal{C}$ as a uniformly quasi-wide class of graphs. We present a proof that on bounded expansion classes, splitter can win much faster.

▶ **Theorem 7.** *Let G be a graph, let $r \in \mathbb{N}$ and let $\ell = \mathrm{wcol}_{2r}(G)$. Then splitter wins the ℓ-round radius-r splitter game.*

Proof. Let L be a linear order that witnesses $\mathrm{wcol}_{2r}(G) = \ell$. Suppose in round $i+1 \leq \ell$, connector chooses a vertex $v_{i+1} \in V(G_i)$. Let w_{i+1} (splitter's choice) be the minimum vertex of $N_r^{G_i}(v_{i+1})$ with respect to L. Then for each $u \in N_r^{G_i}(v_{i+1})$ there is a path between u and w_{i+1} of length at most $2r$ that uses only vertices of $N_r^{G_i}(v_{i+1})$. As w_i is minimum in $N_r^{G_i}(v_{i+1})$, w_{i+1} is weakly $2r$-reachable from each $u \in N_r^{G_i}(v_{i+1})$. Now let $G_{i+1} := G_i[N_r^{G_i}(v_{i+1}) \setminus \{w_{i+1}\}]$. As w_{i+1} is not part of G_{i+1}, in the next round splitter will choose another vertex which is weakly $2r$-reachable from every vertex of the remaining r-neighbourhood. As $\mathrm{wcol}_{2r}(G) = \ell$, the game must stop after at most ℓ rounds. ◀

4 Uniform orders for graphs excluding a topological minor

If $\mathcal{C}$ is a class of bounded expansion such that $\mathrm{wcol}_r(G) \leq f(r)$ for all $G \in \mathcal{C}$ and all $r \in \mathbb{N}$, the order L that witnesses this inequality for G may depend on the value r. We say that a class $\mathcal{C}$ *admits uniform orders* if there is a function $f : \mathbb{N} \to \mathbb{N}$ such that for each $G \in \mathcal{C}$, there is a linear order $L \in \Pi(G)$ such that $|\mathrm{WReach}_r[G, L, v]| \leq f(r)$ for all $v \in V(G)$ and all $r \in \mathbb{N}$. In other words, there is one order that simultaneously certifies the inequality $\mathrm{wcol}_r(G) \leq f(r)$ for all r.

It is implicit in [16] that every class that excludes a fixed minor admits uniform orders, which can be efficiently computed. We are going to show that the same holds for classes that exclude a fixed topological minor. Our construction is similar to the construction of [16], in particular, our orders can be computed quickly in a greedy fashion. The proof that we find an order of high quality is based on the decomposition theorem for graphs with excluded topological minors, due to Grohe and Marx [9]. Note however, that for the construction of the order we do not have to construct a tree decomposition according to Grohe and Marx [9].

Construction. Let G be a graph. We present a construction of an order of $V(G)$ of high quality. We iteratively construct a sequence $H_1, \ldots, H_\ell$ of pairwise disjoint and connected subgraphs of G such that $\bigcup_{1 \leq i \leq \ell} V(H_i) = V(G)$. For $0 \leq i < \ell$, let $G_i := G - \bigcup_{1 \leq j \leq i} V(H_j)$. We say that a component C of G_i is *connected* to a subgraph H_j, $j \leq i$, if there is a vertex $u \in V(H_j)$ and a vertex $v \in V(C)$ such that $uv \in E(G)$. For all i, $1 \leq i < \ell$, we will maintain the following invariant. If C is a component of G_i, then the subgraphs $H_{i_1}, \ldots, H_{i_s} \in \{H_1, \ldots, H_i\}$ that are connected to C form a minor model of the complete graph K_s, where s is their number.

To start, we choose an arbitrary vertex $v \in V(G)$ and let H_1 be the connected subgraph $G[\{v\}]$. Clearly, H_1 satisfies the above invariant. Now assume that for some i, $1 \leq i < \ell$, the sequence $H_1, \ldots, H_i$ has already been constructed. Fix some component C of G_i and, by the invariant, assume that the subgraphs $H_{i_1}, \ldots, H_{i_s} \in \{H_1, \ldots, H_i\}$ with $1 \leq i_1 < \ldots < i_s \leq i$ that have a connection to C form a minor model of K_s. For a vertex $v \in V(C)$, let $m(v)$ be the maximum cardinality of a family $\mathcal{P}$ of paths with the following properties: each path of $\mathcal{P}$ connects v with a different subgraph H_{i_j}, the internal vertices of each path from $\mathcal{P}$ belong to G_i, and the paths of $\mathcal{P}$ are pairwise disjoint apart from sharing v. Note that $m(v)$ can be computed in polynomial time using any maximum flow algorithm. Pick v to be a vertex of C with maximum $m(v)$. Let T be the tree of the breadth-first search in $G[C]$ that starts in v; thus, T is rooted at v. We choose H_{i+1} to be a minimal connected subtree of T that contains v and, for each j with $1 \leq j \leq s$, at least one neighbour of H_{i_j} in C.

From the construction it is easy to see that for every component C' of G_{i+1}, the subgraphs $H'_{i_1}, \ldots, H'_{i_{s'}} \in \{H_1, \ldots, H_{i+1}\}$ that are connected to C' form the minor model of a complete graph, hence the invariant is again established. Having chosen H_{i+1}, we proceed to the next iteration. The construction stops when all vertices are part of some H_i, $1 \leq i \leq \ell$.

We construct an order L of $V(G)$ as follows. Let $v <_L u$ if $v \in V(H_i)$ and $u \in V(H_j)$ for some $i < j$. Furthermore, we order the vertices within each H_i arbitrarily. Obviously, the construction does not depend on r, hence the produced order is uniform for G.

Analysis. From now on we assume that G excludes K_k as a topological minor, for some constant k. Furthermore, assume that the graphs $H_1, \ldots, H_\ell$ and a corresponding order L have been constructed, as described above. We now show that the constructed order has good qualities. Our proof is based on the following two key lemmas. The first lemma states that for every component C of G_i arising after the construction of $H_1, \ldots, H_i$, every vertex v of C can reach only a bounded number of subgraphs among $H_1, \ldots, H_i$ by disjoint paths.

► **Lemma 8.** *There is a constant α (depending only on k) such that for all integers i, $1 \le i < \ell$, if C is a component of G_i, then for every vertex $v \in V(C)$, we have $m(v) \le \alpha$, where $m(v)$ is defined as in the construction.*

The second lemma states that from a vertex of H_{i+1}, we can reach only a bounded number of vertices of each H_j, $1 \le j \le i+1$, by short disjoint paths in G_i.

► **Lemma 9.** *There is a constant β (depending only on k) such that for all integers i, j, where $1 \le j \le i \le \ell$, and all positive integers r, the following holds. Suppose $v \in V(H_i)$, and let $\mathcal{P}$ be any family of paths of length at most r with the following properties: each path from $\mathcal{P}$ connects v with a different vertex of H_j, the internal vertices of $\mathcal{P}$ belong to G_j, and paths from $\mathcal{P}$ are internally vertex disjoint. Then $\mathcal{P}$ has size not larger than $\beta \cdot r$.*

It is easy to show that the above two lemmas guarantee that L has the required properties. The proof of this fact, as well as all the other facts marked with $*$, is in the appendix.

► **Corollary 10** ($*$). *If $K_k \not\preccurlyeq^t G$, then there exists a constant γ (depending only on k) and a uniform order L that witnesses $\mathrm{adm}_r(G) \le \gamma \cdot r$ for all non-negative integers r.*

The proof of Lemma 8 is based on the decomposition theorem for graphs with excluded topological minors of Grohe and Marx [9]. Recall that a *tree decomposition* of a graph G is a pair (T, β), where T is a tree and $\beta : V(T) \to 2^{V(G)}$, such that for every vertex $v \in V(G)$ the set $\beta^{-1}(v) = \{t \in V(T) : v \in \beta(t)\}$ is non-empty and connected in T, and for every edge $e \in E(G)$ there is a node $t \in V(T)$ such that $e \subseteq \beta(t)$. The *width* of (T, β) is $\max\{|\beta(t)| - 1 : t \in V(T)\}$ and the *adhesion* of (T, β) is $\max\{|\beta(s) \cap \beta(t)| : st \in E(T)\}$.

For a node $t \in T$, we call $\beta(t)$ the *bag at t*. If $T' \subseteq T$, we write $\beta(T')$ for $\bigcup_{t' \in V(T')} \beta(t')$ and if $M \subseteq V(G)$, we write $\beta^{-1}(M)$ for $\bigcup_{v \in M} \beta^{-1}(v)$. Denote by $K[X]$ the complete graph on a vertex set X. The *torso at t* is the graph $\tau(t) := G[\beta(t)] \cup \bigcup_{st \in E(T)} K[\beta(s) \cap \beta(t)]$.

► **Theorem 11** ([9]). *For every $k \in \mathbb{N}$, there exist constants $a(k), c(k), d(k)$ and $e(k)$ such that the following holds. Let H be a graph on k vertices. Then for every graph G with $H \not\preccurlyeq^t G$ there is a tree decomposition (T, β) of adhesion at most $a(k)$ such that for all $t \in V(T)$ one of the following two alternatives hold.*

1. *The torso $\tau(t)$ has at most $c(k)$ vertices of degree larger than $d(k)$, which we call the* apex vertices *of $\tau(t)$. Such a node t will be called a* bounded degree node*.*
2. *The torso $\tau(t)$ excludes the complete graph $K_{e(k)}$ as a minor. Such a node t will be called an* excluded minor node*.*

We will need the following well-known properties of trees and tree decompositions.

► **Lemma 12** (Helly-property for trees). *Let T be a tree and let $(T_i)_{i \in I}$ be a family of subtrees of T. If $V(T_i) \cap V(T_j) \neq \emptyset$, for all $i, j \in I$, then $\bigcap_{i \in I} V(T_i) \neq \emptyset$.*

▶ **Lemma 13.** *Let (T, β) be a tree decomposition of a graph G. Let $e = st$ be an edge of T and let T_1, T_2 be the components of $T - e$. Then $\beta(s) \cap \beta(t)$ separates $\beta(T_1)$ from $\beta(T_2)$, that is, every path from a vertex of $\beta(T_1)$ to a vertex of $\beta(T_2)$ traverses a vertex of $\beta(s) \cap \beta(t)$.*

▶ **Lemma 14.** *If $H \subseteq G$ is a connected subgraph of G, then $\beta^{-1}(V(H))$ is connected in T.*

For the proof of Lemma 8, assume that G is decomposed as described by Theorem 11. Assume that $H_1, \ldots, H_i$ have been constructed and let C be a component of G_i that has a connection to the subgraphs $H_{i_1}, \ldots, H_{i_s}$. Recall that throughout the construction we guarantee that the subgraphs $H_{i_1}, \ldots, H_{i_s}$ form the minor model of a complete graph K_s. We first identify one bag of the decomposition as a bag which intersects many distinct branch sets of this minor model. The following lemma follows easily from the separator properties of tree decompositions, in particular Lemma 13.

▶ **Lemma 15 (∗).** *There can be at most one node t such that $\beta(t)$ intersects strictly more than $a(k)$ of the branch sets H_{i_j}, for $1 \leq j \leq s$.*

We now show that there is a bag that intersects every branch set. The proof is a simple application of the Helly property of trees (Lemma 12) and Lemma 14.

▶ **Lemma 16 (∗).** *There is a node t such that $\beta(t)$ intersects each H_{i_j}, for $1 \leq j \leq s$.*

Hence, provided $s > a(k)$, there is a node t with $\beta(t)$ intersecting at least $a(k)+1$ branch sets H_{i_j}. By Lemma 15, this node is unique. We call it the *core node* of the minor model. Next we show that if the model is large, then its core node must be a bounded degree node. Shortly speaking, this is because the model $H_{i_1}, \ldots, H_{i_s}$ trimmed to the torso of the core node is already a minor model of K_s in this torso.

▶ **Lemma 17 (∗).** *If $s > \max\{a(k), e(k)\}$, then the core node of the minor model is a bounded degree node.*

For vertices outside the bag of the core node, the bound promised in Lemma 8 can be proved similarly as Lemma 15.

▶ **Lemma 18 (∗).** *Let C be a component of G_i that has a connection to the subgraphs $H_{i_1}, \ldots, H_{i_s}$. If $s > a(k)$, then for every vertex $v \in V(C) \setminus \beta(t)$, where t is the core node of the model, we have that $m(v) \leq a(k)$.*

We now complete the proof of Lemma 8 by looking at the vertices inside the core bag.

Proof of Lemma 8. We set $\alpha := a(k) + c(k) + d(k) + e(k)$. Assume towards a contradiction that for some i, $1 \leq i < \ell$, we have that some component C of G_i contains a vertex v_1 with $m(v_1) > \alpha$. Denote the branch sets that have a connection to C by $H_{i_1}, \ldots, H_{i_s}$, where $i_1 < i_2 < \ldots < i_s$. Let $\mathcal{P}$ be a maximum-size family of paths that pairwise share only v_1 and connect v_1 with different branch sets H_{i_j}. As $m(v_1) > \alpha$, we have that $|\mathcal{P}| > \alpha$, and in particular $s > \alpha$. As $\alpha > a(k)$, by Lemmas 15 and 16 we can identify the unique core node t of the minor model. As $s > \max\{a(k), e(k)\}$, by Lemma 17 the core node is a bounded degree node. As $m(v_1) > a(k)$, by Lemma 18 we have $v_1 \in \beta(t)$. As $\mathcal{P}$ contains more than $d(k)$ disjoint paths from v to distinct branch sets, the degree of v_1 in G must be greater than $d(k)$, hence v_1 is an apex vertex of $\tau(t)$.

Since $i_1 < i_2 < \ldots < i_s$, we have that the component C was created when H_{i_s} was removed from G_{i_s-1}. Let C' be the component of G_{i_s-1} that contains C and H_{i_s} (and thus v_1). Observe that C' is still connected to $H_1, \ldots, H_{i_{s-1}}$, and possibly to some other branch sets. Recall that H_{i_s} was constructed as a subtree of the breadth-first search tree in G_{i_s} that started in a vertex $v_2 \in V(C')$ which, at this point of the construction, had maximum $m(v_2)$ among vertices in C'. However, at this point vertex v_1 was also present in C', and $\mathcal{P}$ certifies that it could send at least $\alpha - 1$ disjoint paths to different branch sets among $H_1, \ldots, H_{i_{s-1}}$ (in $\mathcal{P}$, at most one path leads to H_{i_s}, and all the other paths are also present in C'). We infer that it held that $m(v_2) \geq \alpha - 1$ at the moment v_2 was taken. Since $\alpha > a(k) + c(k) + d(k) + e(k) \geq a(k) + d(k) + e(k) + 1$, the same reasoning as above shows that t is also the core vertex of the minor model formed by branch sets connected to C'. Thus, by exactly the same reasoning we obtain that v_2 is also an apex vertex of $\tau(t)$.

Since $\alpha > a(k)+c(k)+d(k)+e(k)$, we can repeat this reasoning $c(k)+1$ times, obtaining vertices $v_1, \ldots, v_{c(k)+1}$, which are all apex vertices of $\tau(t)$. This contradicts the fact that $\tau(t)$ contains at most $c(k)$ apex vertices. ◀

At last, we come to the proof of Lemma 9

Proof of Lemma 9. We set β so that $\beta \cdot r \geq (2r+1) \cdot \alpha$, where α is the constant given by Lemma 8. For the sake of contradiction, suppose there is a family of paths $\mathcal{P}$ as in the statement, whose size is larger than $(2r+1) \cdot \alpha$.

Recall that H_j was chosen as a subtree of a breadth-first search tree in G_{j-1}; throughout the proof, we treat H_j as a rooted tree. As H_j is a subtree of a BFS tree, every path from a vertex w of the tree to the root v' of the tree is an isometric path in G_{j-1}, that is, a shortest path between w and v' in the graph G_{j-1}. If P is an isometric path in a graph H, then $|N_r^H(v) \cap V(P)| \leq 2r+1$ for all $v \in V(H)$ and all $r \in \mathbb{N}$. As the paths from $\mathcal{P}$ are all contained in G_{j-1}, and they have lengths at most r, this implies that the path family $\mathcal{P}$ cannot connect v with more than $2r+1$ vertices of H_j which lie on the same root-to-leaf path in H_j. Since $|\mathcal{P}| > (2r+1) \cdot \alpha$, we can find a set $X \subseteq V(H_j)$ such that $|X| > \alpha$, each vertex of X is connected to v by some path from $\mathcal{P}$, and no two vertices of X lie on the same root-to-leaf path in H_j. Recall that, by the construction, each leaf of H_j is connected to a different branch set $H_{j'}$ for some $j' < j$. Consequently, we can take the paths of $\mathcal{P}$ leading to X and extend them within H_j to obtain a family of more than α disjoint paths in G_{j-1} that connect v with different branch sets $H_{j'}$ for $j' < j$. This contradicts Lemma 8. ◀

Observe that the order can be computed in time $\mathcal{O}(n^5)$: for each vertex, we compute by a standard flow algorithm in time $\mathcal{O}(n^3)$ whether it should be chosen as the next tree root to form a subgraph H_{i_j}. This choice has to be made at most n times.

Finally, we state one property of the construction that follows immediately from Lemma 8.

▶ **Lemma 19.** *Each constructed subgraph H_i has maximum degree at most $\alpha+1$, where α is the constant given by Lemma 8.*

5 Model-checking for successor-invariant first-order formulas

A *finite and purely relational signature* τ is a finite set $\{R_1, \ldots, R_k\}$ of relation symbols, where each relation symbol R_i has an associated arity a_i. A finite τ-structure $\mathfrak{A}$ consists of a finite set A, the universe of $\mathfrak{A}$, and a relation $R_i(\mathfrak{A}) \subseteq A^{a_i}$ for each relation symbol $R_i \in \tau$. If $\mathfrak{A}$ is a finite τ-structure, then the *Gaifman graph* of $\mathfrak{A}$, denoted $G(\mathfrak{A})$, is the graph with $V(G(\mathfrak{A})) = A$ and there is an edge $uv \in E(G(\mathfrak{A}))$ if and only if $u \neq v$ and u and v appear together in some relation $R_i(\mathfrak{A})$ of $\mathfrak{A}$. We say that a class $\mathcal{C}$ of finite τ-structures has *bounded expansion* if the graph class $G(\mathcal{C}) := \{G(\mathfrak{A}) : \mathfrak{A} \in \mathcal{C}\}$ has bounded expansion. Similarly, for $r \in \mathbb{N}$, we write $\mathrm{adm}_r(\mathfrak{A})$ for $\mathrm{adm}_r(G(\mathfrak{A}))$ etc.

Let V be a set. A successor relation on V is a binary relation $S \subseteq V \times V$ such that (V, S) is a directed path of length $|V| - 1$. Let τ be a finite relational signature. A formula $\varphi \in \mathrm{FO}[\sigma \cup \{S\}]$ is *successor-invariant* if for all τ-structures $\mathfrak{A}$ and for all successor relations S_1, S_2 on $V(\mathfrak{A})$ it holds that $(\mathfrak{A}, S_1) \models \varphi \iff (\mathfrak{A}, S_2) \models \varphi$.

Successor-invariant logics have been studied in database theory and finite model theory in the past. It was shown by Rossman [15] that successor-invariant FO is more expressive than FO without access to a successor relation. It is known that successor-invariant FO (in fact even order-invariant FO) can express only local queries [10], however, the proof does not translate formulas into local FO-formulas which could be evaluated algorithmically. It was shown in [7] that the model-checking problem for successor-invariant first-order formulas is fixed-parameter tractable on any proper minor closed class of graphs. Very recently, the same result was shown for classes with excluded topological minors [6]. We give a new proof of the model-checking result of [6] which is based on the nice properties of the order we have constructed for graphs that exclude a topological minor.

Eickmeyer et al. [7] showed that on well-behaved classes of graphs one can apply the following reduction from the model-checking problem for successor-invariant formulas to the model-checking problem for plain first-order formulas.

▶ **Lemma 20** (Eickmeyer et al. [7]). *Let $\mathcal{C}$ be a class of τ-structures such that for each $\mathfrak{A} \in \mathcal{C}$ one can compute in polynomial time a graph $H(\mathfrak{A})$ such that*

1. *$V(H(\mathfrak{A})) = V(G(\mathfrak{A}))$ and $E(H(\mathfrak{A})) \supseteq E(G(\mathfrak{A}))$.*
2. *H contains a spanning tree T which can be computed in polynomial time and which is of maximum degree d for some fixed integer d depending on $\mathcal{C}$ only.*
3. *The model-checking problem for first-order formulas on the graph class $\{H(\mathfrak{A}) : \mathfrak{A} \in \mathcal{C}\}$ is fixed-parameter tractable.*

Then the model-checking problem for successor-invariant first-order formulas is fixed-parameter tractable on $\mathcal{C}$.

We remark that the original lemma from [7] refers to k-walks in H, which are easily seen to be equivalent to spanning trees of maximum degree k. In our view, spanning trees are more intuitive to handle in our graph theoretic context.

▶ **Lemma 21.** *Let $k \in \mathbb{N}$. There is a constant δ, depending only on k, and a function $f : \mathbb{N} \to \mathbb{N}$ such that the following holds. For every graph G with $K_k \not\preccurlyeq^t G$ we can compute in polynomial time a supergraph H with $V(H) = V(G)$ and $E(H) \supseteq E(G)$ such that $\mathrm{adm}_r(H) \le f(r)$ for all $r \in \mathbb{N}$ and such that H contains a spanning tree T with maximum degree at most δ; furthermore, such a spanning tree T can be also computed in polynomial time.*

Proof. Without loss of generality, we assume that G is connected. Otherwise, we may apply the construction in each connected component separately, and then connect the components arbitrarily using single edges (added to H) in a path-like manner. It is easy to see that including the additional edges to the spanning tree increases its maximum degree by at most 2, while the admissibility of the graph also increases by at most 2.

We perform the construction of the subgraphs $H_1, \ldots, H_\ell$ almost exactly as in Section 4. However, when constructing the H_i's and the order L, we put some additional restrictions that do not change the quality of L. First, recall that when we defined H_{i+1}, for some $0 \le i < \ell$, we considered a tree of breadth-first search starting at v_{i+1} in a connected component C of G_i. Suppose that the subgraphs that C is connected to are $H_{i_1}, \ldots, H_{i_s}$, where $1 \le i_1 < \ldots < i_s \le i$. Then H_{i+1} was defined as a minimal subtree of the considered BFS tree that contained, for each $1 \le j \le s$, some vertex of H_{i_j} that is adjacent to C.

Observe that in the construction we were free to choose which neighbour of H_{i_j} will be picked to be included in H_{i+1}. For $j < s$ we make an arbitrary choice as before, but the neighbour of H_{i_s} (if exists; note that this is the case for $i > 0$) is chosen as follows. We first select the vertex $w'_{i+1} \in V(H_{i_s})$ that is the largest in the order L among those vertices of H_{i_s} that are adjacent to C (the vertices of H_j for $j \leq i$ are already ordered by L at this point). Then, we select any its neighbour w_{i+1} in C as the vertex that is going to be included in H_{i+1} in its construction. Finally, recall that in the construction of L, we could order the vertices of H_{i+1} arbitrarily. Hence, we fix an order of H_{i+1} so that w_{i+1} is the smallest among $V(H_{i+1})$. This concludes the description of the restrictions applied to the construction.

We now construct H by taking G and adding some edges. During the construction, we will mark some edges of H as *spanning edges*. We start by marking all the edges of all the trees H_i, for $1 \leq i \leq \ell$, as spanning edges. At the end, we will argue that the spanning edges form a spanning tree of H with maximum degree at most δ.

For each i with $1 \leq i < \ell$, let us examine the vertex w_{i+1}, and let us *charge* it to w'_{i+1}. Note that in this manner every vertex w_{i+1} is charged to its neighbour that lies before it in the order L. For any $w \in V(G)$, let $D(w)$ be the set of vertices charged to w. Now examine the vertices of G one by one, and for each $w \in V(G)$ do the following. If $D(w) = \emptyset$, do nothing. Otherwise, if $D(w) = \{u_1, u_2, \ldots, u_h\}$, mark the edge wu_1 as a spanning edge, and add edges $u_1u_2, u_2u_3, \ldots, u_{h-1}u_h$ to H, marking them as spanning edges as well.

▶ **Claim 22** (∗). *The spanning edges form a spanning tree of H of maximum degree at most $\alpha + 4$, where α is the constant given by Lemma 8.*

It remains to argue that H has small admissibility. For this, it suffices to prove the following claim. The proof uses the additional restrictions we introduced in the construction.

▶ **Claim 23** (∗). *Let r be a positive integer. If the order L certifies that $\mathrm{col}_{2r}(G) \leq m$, that is, $\max_{v \in V(G)} |\mathrm{SReach}_{2r}[G, L, v]| \leq m$, then $\mathrm{adm}_r(H) \leq m + 2$.*

The statement of the lemma now directly follows from Claims 22 and 23. ◀

Given a graph G that excludes K_k as a topological minor, let us write $H(G)$ for a graph constructed according to Lemma 21.

▶ **Corollary 24.** *The class $\{H(G) : K_k \not\preccurlyeq^t G\}$ has bounded expansion.*

We can now use Theorem 20 to combine the following result of Dvořak et al. [5] with Lemma 21, to prove fixed-parameter tractability of successor-invariant FO on classes that exclude a fixed topological minor.

▶ **Lemma 25** (Dvořák et al. [5]). *The model-checking problem for first-order formulas is fixed-parameter tractable on any class of bounded expansion.*

▶ **Corollary 26.** *The model-checking problem for successor-invariant first-order formulas is fixed parameter tractable on any class of graphs that excludes a fixed topological minor.*

6 Conclusions

In this work we gave several new applications of the generalised colouring numbers on classes of bounded expansion. In particular, we have shown that whenever a graph class $\mathcal{C}$ excludes some fixed topological minor, then any graph from $\mathcal{C}$ admits one ordering of vertices that certifies the boundedness of the generalised colouring numbers for all radii r at once. It is tempting to conjecture that such an ordering exists for any graph class of bounded expansion.

Our construction of the uniform ordering proved to be useful in showing that model-checking successor-invariant FO is FPT on any graph class that excludes a fixed topological minor. We believe that our construction may be helpful in extending this result to any graph class of bounded expansion, since both the construction of the order, and the reasoning of Section 5, are oblivious to the fact that the graph class excludes some topological minor. The only place where we used this assumption is the analysis of the constructed order.

References

1 Albert Atserias, Anuj Dawar, and Phokion G Kolaitis. On preservation under homomorphisms and unions of conjunctive queries. *Journal of the ACM (JACM)*, 53(2):208–237, 2006.

2 Anuj Dawar. Homomorphism preservation on quasi-wide classes. *Journal of Computer and System Sciences*, 76(5):324–332, 2010.

3 Reinhard Diestel. *Graph Theory: Springer Graduate Text GTM 173*, volume 173. Reinhard Diestel, 2012.

4 Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34:833–840, 2013.

5 Zdeněk Dvořák, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *Journal of the ACM (JACM)*, 60(5):36, 2013.

6 Kord Eickmeyer and K. Kawarabayashi. Personal communication, 2016.

7 Kord Eickmeyer, K. Kawarabayashi, and Stephan Kreutzer. Model checking for successor-invariant first-order logic on minor-closed graph classes. In *Proceedings of the 28th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS), 2013*, pages 134–142. IEEE, 2013.

8 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pages 89–98. ACM, 2014.

9 Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. *SIAM Journal on Computing*, 44(1):114–159, 2015.

10 Martin Grohe and Thomas Schwentick. Locality of order-invariant first-order formulas. *ACM Transactions on Computational Logic (TOCL)*, 1(1):112–130, 2000.

11 Hal A Kierstead and Daqing Yang. Orderings on graphs and game coloring number. *Order*, 20(3):255–264, 2003.

12 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.

13 Jaroslav Nešetřil and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *The Journal of Symbolic Logic*, 75(03):868–887, 2010.

14 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.

15 Benjamin Rossman. Successor-invariant first-order logic on finite structures. *The Journal of Symbolic Logic*, 72(02):601–618, 2007.

16 Jan van den Heuvel, Patrice Ossona de Mendez, Daniel Quiroz, Roman Rabinovich, and Sebastian Siebertz. On the generalised colouring numbers of graphs that exclude a fixed minor. *CoRR*, abs/1602.09052, 2016. URL: `http://arxiv.org/abs/1602.09052`.

17 Xuding Zhu. Colouring graphs with bounded generalized colouring number. *Discrete Mathematics*, 309(18):5562–5568, 2009.

Polynomial Space Randomness in Analysis*

Xiang Huang[1] **and Donald M. Stull**[2]

1 **Iowa State University, Department of Computer Science, Ames, IA 50011 USA**
huangx@iastate.edu

2 **Iowa State University, Department of Computer Science, Ames, IA 50011 USA**
dstull@iastate.edu

Abstract

We study the interaction between polynomial space randomness and a fundamental result of analysis, the Lebesgue differentiation theorem. We generalize Ko's framework for polynomial space computability in $\mathbb{R}^n$ to define *weakly pspace-random* points, a new variant of polynomial space randomness. We show that the Lebesgue differentiation theorem characterizes weakly pspace random points. That is, a point x is weakly pspace random if and only if the Lebesgue differentiation theorem holds for a point x for every pspace L_1-computable function.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases algorithmic randomness, computable analysis, resource-bounded randomness, complexity theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.86

1 Introduction

The theory of computing allows for a meaningful definition of an individual point of Euclidean space being “random". Classically, such a notion would seem paradoxical, as any singleton set (indeed, any countably infinite set) has measure zero. Martin-Löf used computability to give the first mathematically robust definition of a point being random [10]. Since Martin-Löf's original definition, many notions of randomness have been introduced. In addition to Martin-Löf randomness, two of the most prominent variants are Schnorr randomness and computable randomness [4]. By developing a theory of resource-bounded measure, Lutz initiated the study of resource-bounded randomness [12, 13]. This allowed for research in algorithmic randomness to extend to resource-bounded computation [21].

Recently, research in algorithmic randomness has used computable analysis to study the connection between randomness and classical analysis [1, 5, 6, 7, 14, 15, 20]. With the rise of measure theory, many fundamental theorems of analysis have been “almost everywhere" results. Theorems of this type state that a certain property holds for almost every point; i.e., the set of points that does not satisfy the property is of measure zero. However, almost everywhere theorems typically give no information about which points satisfy the stated property. By adding computability restrictions, tools from algorithmic randomness are able to strengthen a theorem from a property simply holding almost everywhere, to one that holds for all random points. For example, an important classical result of analysis is Lebesgue's theorem on nondecreasing functions. Lebesgue showed that every nondecreasing continuous function $f : [0, 1] \to \mathbb{R}$ is differentiable almost everywhere. Brattka, Miller and

* This research was supported in part by National Science Foundation Grants 1247051 and 1545028.

Nies characterized computable randomness using Lebesgue's theorem by proving the following result [2].

▶ **Theorem.** *Let $z \in [0,1]$. Then z is computably random if and only if $f'(z)$ exists for every nondecreasing computable function $f : [0,1] \to \mathbb{R}$.*

This paper concerns a related theorem, also due to Lebesgue [9].

▶ **Lebesgue Differentiation Theorem.** *For each $f \in L_1([0,1]^n)$,*

$$f(x) = \lim_{Q \to x} \frac{\int_Q f d\mu}{\mu(Q)}$$

for almost every $x \in [0,1]^n$. The limit is taken over all open cubes Q containing x as the diameter of Q tends to 0.

Pathak first studied the Lebesgue differentiation theorem in the context of Martin-Löf randomness [18]. Under the assumption that the function is L_1-computable, Pathak showed that the Lebesgue differentiation theorem holds for every Martin-Löf random point. Subsequently, Pathak, Rojas and Simpson improved this theorem [19]. They showed that the Lebesgue differentiation theorem holds at a point z for every L_1 computable function if and only if z is Schnorr random [19]. Independently, and using very different techniques, Rute also showed that the Lebesgue differentiation theorem holds for Schnorr random points [20].

This paper concerns the connection between *resource-bounded* randomness and analysis. While there has been work on this interaction [3, 11, 17], resource-bounded randomness in analysis is still poorly understood. Recently, Nies extended the result of Brattka, Miller and Nies to the polynomial time domain [17]. Specifically, Nies characterized polynomial time randomness using the differentiability of nondecreasing polynomial time computable functions. In this paper, we extend this research of the Lebesgue differentiation theorem to the context of resource-bounded randomness. We show that the Lebesgue differentiation theorem characterizes *weakly polynomial space randomness.* We note that the polynomial space variant of Nies' result implies our result in one dimension. However, as in classical analysis, the proof for arbitrary dimension requires significantly different tools.

In order to work with resource bounded computability over continuous domains, we use the framework for polynomial space computability in $\mathbb{R}^n$ developed by Ko [8]. Using generalizations of Ko's *polynomial space approximable* sets, we define weakly polynomial space randomness, a new variant of polynomial space randomness. We prove that Lutz's notion of polynomial space randomness implies weakly polynomial randomness. Weakly polynomial space randomness uses open covers, similar to Martin-Löf's original definition, unlike the martingale definitions commonly used in resource-bounded randomness. The use of open covers lends itself better to adapting many theorems of classical analysis. We believe that the notion of weakly polynomial space randomness will be useful in further investigations of resource-bounded randomness in analysis.

Using this definition of randomness, we extend the result of Pathak, et al, and Rute to polynomial space randomness. Specifically, we prove that a point x is weakly polynomial space random if and only if the Lebesgue differentiation theorem holds at x for every polynomial space L_1-computable function. Structurally, the proof of this theorem largely follows that of Pathak, et al. However, the restriction to polynomial space forces significant changes to the internal methods. To prove the converse of our theorem, we introduce *dyadic tree decompositions.* Intuitively, a dyadic tree decomposition partitions an open cover randomness test into a tree structure. This allows for the construction of a polynomial space

L_1-computable function so that the Lebesgue differentiation theorem fails for any point covered by the test. We believe that dyadic tree decompositions will be useful in further research.

2 Preliminaries

Throughout the paper, μ will always denote the Lebesgue measure on $\mathbb{R}^n$. We denote the set of all Lebesgue integrable functions $f : [0,1]^n \to \mathbb{R}$ by $L_1([0,1]^n)$. A *dyadic rational number* d is a rational number that has a finite binary expansion; that is $d = \frac{m}{2^r}$ for some integers m, r with $r \geq 0$. We denote the set of all dyadic rational numbers by $\mathbf{D}$. We denote the set of all dyadic rationals d of precision r by $\mathbf{D}_r$. Formally,

$$\mathbf{D}_r = \{\frac{m}{2^r} \,|\, m \in \mathbb{Z}\}.$$

We denote the set of dyadic rationals in the interval $[0,1]$ by $\mathbf{D}[0,1]$. We denote the set of dyadic rationals of precision r in the interval $[0,1]$ by $\mathbf{D}_r[0,1]$. An *open dyadic cube of precision* r is a subset $Q \subseteq \mathbb{R}^n$ such that

$$Q = (\frac{a_1}{2^r}, \frac{a_1+1}{2^r}) \times \ldots \times (\frac{a_n}{2^r}, \frac{a_n+1}{2^r}),$$

where $a_i \in \mathbb{Z}$, and $r \in \mathbb{N}$. We say that the points $\{\frac{a_1}{2^r}, \frac{a_1+1}{2^r}, \ldots \frac{a_n}{2^r}, \frac{a_n+1}{2^r}\}$ are the *endpoints of* Q. In the same manner, we define closed dyadic cubes, and half-open dyadic cubes. We denote the set of all open dyadic cubes of precision r by

$$\mathbf{B}_r = \{Q \,|\, Q \text{ is an open dyadic cube of precision } r\}.$$

For an open set $Q \subseteq \mathbb{R}^n$ and $t \in \mathbb{R}^n$, define the translation of Q by t to be the set

$$t + Q = \{t + x \,|\, x \in Q\}.$$

2.1 Resource-Bounded Randomness in Euclidean Space

Lutz and Lutz recently adapted resource-bounded randomness to arbitrary dimension [11]. In this section, we review their definition of polynomial space randomness in $\mathbb{R}^n$.

Let $r \in \mathbb{N}$, $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{Z}^n$. Define the *$r$-dyadic cube at* $\mathbf{u}$ to be the half-open dyadic cube of precision r,

$$Q_r(\mathbf{u}) = [u_1 \cdot 2^{-r}, (u_1+1) \cdot 2^{-r}) \times \ldots \times [u_n \cdot 2^{-r}, (u_n+1) \cdot 2^{-r}).$$

Define the family

$$\mathcal{Q}_r = \{Q_r(\mathbf{u}) \mid \mathbf{u} \in \{0, \ldots, 2^r - 1\}^n\}.$$

So then $\mathcal{Q}_r$ is a partition of the unit cube $[0,1)^n$. The family

$$\mathcal{Q} = \bigcup_{r=0}^{\infty} \mathcal{Q}_r,$$

is the set of all half-open dyadic cubes in $[0,1)^n$.

A *martingale* on $[0,1)^n$ is a function $d : \mathcal{Q} \to [0,\infty)$ satisfying

$$d(Q_r(\mathbf{u})) = 2^{-n} \sum_{\mathbf{a} \in \{0,1\}^n} d(Q_{r+1}(2\mathbf{u} + \mathbf{a})), \tag{1}$$

for all $Q_r(\mathbf{u}) \in \mathcal{Q}$. We may think of a martingale d as a strategy for placing successive bets on which cube contains x. After r bets have been placed, the bettor's capital is

$$d^{(r)}(\mathbf{x}) = d(Q_r(\mathbf{u})),$$

where $\mathbf{u}$ is the unique element of $\{0, \dots, 2^r - 1\}^n$ such that $\mathbf{x} \in Q_r(\mathbf{u})$. A martingale d *succeeds* at a point $\mathbf{x} \in [0,1)^n$ if

$$\limsup_{r \to \infty} d^{(r)}(\mathbf{x}) = \infty.$$

Let

$$J = \{(r, \mathbf{u}) \in \mathbb{N} \times \mathbf{Z}^n \mid \mathbf{u} \in \{0, \dots, 2^r - 1\}^n\}.$$

We say that a martingale $d : \mathcal{Q} \to [0, \infty)$ is *computable* if there is a computable function $\hat{d} : \mathbb{N} \times J \to \mathbb{Q} \cap [0, \infty)$ such that for all $(s, r, \mathbf{u}) \in \mathbb{N} \times J$,

$$|\hat{d}(s, r, \mathbf{u}) - d(Q_r(\mathbf{u}))| \leq 2^{-s}. \tag{2}$$

A martingale $d : \mathcal{Q} \to [0, \infty)$ is *p-computable* (resp. *pspace-computable*) if there is a function $\hat{d} : \mathbb{N} \times J \to \mathbb{Q} \cap [0, \infty)$ that satisfies (2) and is computable in $(s + r)^{O(1)}$ time (resp. space). A point $x \in \mathbb{R}^n$ is *p-random* (resp. *pspace-random*) if no p-computable (resp. pspace-computable) martingale succeeds at x.

2.2 Polynomial Space Computability in Euclidean Space

In this section, we review Ko's framework for complexity theory in $\mathbb{R}^n$ [8]. For the remainder of the paper, we include the write tape when considering polynomial space bounds of Turing machines.

We first introduce the polynomial space L_1-computable functions, the class of functions we will be using in the proof of the Lebesgue differentiation theorem. This definition is equivalent to Ko's notion of *pspace approximable functions*. It is a direct analog of the L_1-computable functions used in computable analysis.

A function $f : [0,1]^n \to \mathbb{R}$ is a *simple step function* if f is a step function such that

1. $f(x) \in \mathbf{D}$ for all $x \in [0,1]^n$ and
2. there exists a finite number of (disjoint) dyadic boxes $Q_1, \dots, Q_k$ and dyadic rationals $d_1, \dots, d_k$ such that $f(x) = \sum_{i=1}^{k} d_i \chi_{Q_i}(x)$, where χ_Q is the characteristic function of a set Q.

A function $f \in L_1([0,1]^n)$ is *polynomial space L_1-computable* if there exists a sequence of simple step functions, $\{f_m\}_{m \in \mathbb{N}}$, and a polynomial p such that for all $d \in \mathbf{D}^n$,

1. $f_m(x) = \sum_{i=1}^{k} d_i \chi_{Q_i}(x)$, such that the endpoints of each Q_i are in $\mathbf{D}^n_{p(m)}$,
2. there is a polynomial space TM M computing f_m in the sense that
$$M(0^m, d) = \begin{cases} f_m(d) & \text{if } d \text{ is not a breakpoint of } f_m \\ \# & \text{otherwise} \end{cases}$$
3. $\|f - f_m\|_1 \leq 2^{-n}$.

Note that we may assume that the polynomial p is increasing. We will frequently use the following nice property of polynomial space L_1-computable functions. If $f \in L_1([0,1]^n)$ is approximated by sequence of simple step function $\{f_m\}$ at precision p, then for every $i > 0$, f_i is a constant function on every $Q \in \mathbf{B}_{p(i)}$.

An infinite sequence $\{S_m\}_{m\in\mathbb{N}}$ of finite unions of open boxes is *polynomial space computable* if there exists a polynomial space TM M such that for all $m > 0$, and all $d \in \mathbf{D}^n$,

$$M(0^m, d) = \begin{cases} 1 & \text{if } d \in S_m \\ -1 & \text{if } d \text{ is a boundary point of } S_m \\ 0 & \text{otherwise} \end{cases}$$

A set $S \subseteq [0,1]^n$ is *polynomial space approximable* if S is measurable and there exists a polynomial space computable sequence of sets $\{S_m\}_{m\in\mathbb{N}}$ such that, for every $m > 0$,

1. there is a polynomial p such that all endpoints of S_m are in $\mathbf{D}^n_{p(m)}$ and
2. $\mu(S\Delta S_m) \leq 2^{-m}$.

Note that we may assume that the polynomial p is increasing; that is $p(i) \leq p(i+1)$, for all $i \in \mathbb{N}$.

3 Uniformly Approximable Sequences

We now generalize Ko's definition of approximable sets to approximable *arrays* of sets. We follow Ko in first defining computability, then leveraging this to define approximability.

▶ **Definition 1.** An infinite array $\{S^k_m\}_{k,m\in\mathbb{N}}$ of finite unions of open boxes is *uniformly polynomial space computable* if there exists a polynomial space TM M such that for all $k, m > 0$, and all $d \in \mathbf{D}^n$,

$$M(0^m, 0^k, d) = \begin{cases} 1 & \text{if } d \in S^k_m \\ -1 & \text{if } d \text{ is an boundary point of } S^k_m \\ 0 & \text{otherwise} \end{cases}$$

If $\{S^k_m\}$ is uniformly polynomial space computable and M is a TM satisfying the definition, we say M *computes* $\{S^k_m\}$.

▶ **Definition 2.** A sequence of sets $\{U_m\}_{m\in\mathbb{N}}$ is *uniformly polynomial space approximable* if there exists a uniformly polynomial space computable array of sets $\{S^k_m\}$ and a polynomial p such that

1. all endpoints of S^k_m are in $\mathbf{D}^n_{p(m+k)}$ and
2. $\mu(U_m \Delta S^k_m) \leq 2^{-k}$.

If a polynomial p and a uniformly polynomial space computable sequence $\{S^k_m\}$ satisfies (1) and (2), we say that $\{S^k_m\}_{k,m\in\mathbb{N}}$ *approximates* $\{U_m\}$ *at precision* p. Note that we may assume that the polynomial p is increasing.

We now show that we can construct *uniformly* pspace computable sequences from pspace computable sequences. This lemma will be useful, as polynomial space computability is an easier property to verify than its uniform counterpart.

▶ **Lemma 3.** *Let $\{T_i\}_{i\in\mathbb{N}}$ be a pspace computable sequence, and q_1, q_2 be polynomials. For every k, $m > 0$, define the set S^k_m by*

$$S^k_m = \bigcup_{i=q_1(m)}^{q_2(k)} T_i.$$

Then the array $\{S^k_m\}$ is uniformly polynomial space computable.

Similarly, we are able to construct uniformly pspace approximable sequences from other uniformly approximable sequences.

▶ **Lemma 4.** *Let q be a polynomial, $j \in \mathbb{N}$, and $\{V_i\}_{i\in\mathbb{N}}$ be a uniformly pspace approximable sequence, such that $\mu(V_i) \leq 2^{-i+j}$. Define the sequence $\{U_m\}_{m\in\mathbb{N}}$ by*

$$U_m = \bigcup_{i=q(m)}^{\infty} V_i.$$

Then $\{U_m\}_{m\in\mathbb{N}}$ is a uniformly pspace approximable sequence.

4 Weakly Polynomial Space Randomness

Using uniformly polynomial space approximable sequences, we give an open-cover definition of polynomial space randomness. This variant is intended to be similar to the open-cover definitions of the various computable randomness notions. However, the resource bounds force us to replace the typical enumerability requirements with approximability.

▶ **Definition 5.** Let $a, b \in \mathbb{Z}$. An infinite sequence of open sets $\{U_m\}_{m\in\mathbb{N}} \subseteq [a,b]^n$ is a *polynomial space $\mathcal{W}$-test (pspace $\mathcal{W}$-test)* if the following hold.

1. For every m, $\mu(U_m) \leq 2^{-m}$.
2. There is a uniformly pspace computable array $\{S_m^k\}$ approximating $\{U_m\}$ such that, for all m,

$$U_m \subseteq \liminf_{k\to\infty} S_m^k,$$

A point x *passes* a polynomial space $\mathcal{W}$-test $\{U_m\}_{m\in\mathbb{N}}$ if $x \notin \bigcap_{m=1}^{\infty} U_m$. We say that x is *weakly pspace random* if x passes every polynomial space $\mathcal{W}$-test.

The approximability of pspace $\mathcal{W}$-tests allows us to estimate the measure of the open covers in polynomial space.

▶ **Lemma 6.** *If $\{U_m\}_{m\in\mathbb{N}}$ is a pspace $\mathcal{W}$-test, then there exists a polynomial space TM M such that for every s, r, $m \in \mathbb{N}$ and $\mathbf{u} \in \{0,\ldots,2^r-1\}^n$*

$$|M(0^s, 0^r, \mathbf{u}, 0^m) - \mu(U_m \cap Q_r(\mathbf{u}))| \leq 2^{-s}.$$

We are now able to relate weakly polynomial space randomness with Lutz's pspace randomness. The following lemma shows that pspace randomness implies weakly pspace randomness.

▶ **Theorem 7.** *Let $\{U_m\}_{m\in\mathbb{N}}$ be a polynomial space $\mathcal{W}$-test. Then there exists a pspace martingale d succeeding on all points $x \in \bigcap_{m=1}^{\infty} U_m \bigcap [0,1]^n$.*

5 Randomness and the Lebesgue Differentiation Theorem

In this section we prove our main theorem, that the Lebesgue differentiation theorem characterizes weakly pspace-randomness. Recall the statement of Lebesgue's theorem.

▶ **Lebesgue Differentiation Theorem.** *For each $f \in L_1([0,1]^n)$,*

$$f(x) = \lim_{Q \to x} \frac{\int_Q f d\mu}{\mu(Q)}$$

for almost every $x \in [0,1]^n$. The limit is taken over all open cubes Q containing x as the diameter of Q tends to 0.

A point x that satisfies the Lebesgue differentiation theorem is called a *Lebesgue point*. We will prove the following theorem,

▶ **Main Theorem.** *A point x is weakly pspace-random if and only if for every polynomial space L_1-computable $f \in L_1([0,1]^n)$, and every polynomial space computable sequence of simple functions $\{f_m\}_{m\in\mathbb{N}}$ approximating f,*

$$\lim_{m\to\infty} f_m(x) = \lim_{Q \to x} \frac{\int_Q f d\mu}{\mu(Q)} \tag{3}$$

where the limit is taken over all cubes Q containing x as the diameter of Q tends to 0.

We first make several remarks regarding the form of our main theorem. The use of polynomial space L_1-computability is not simply for the sake of generality. It is well-known that if a function is continuous, the Lebesgue differentiation theorem holds for *every* point. Thus, to get a non-trivial randomness result, we must allow the function to be discontinuous. Our second remark concerns the limit of the approximating functions. In the statement of the classical theorem, the integral limit is equal to $f(x)$; whereas in our main theorem, it is equal to $\lim_{m\to\infty} f_m(x)$. This concession is necessary. For any point x, it is trivial to construct a polynomial space L_1-computable function f such that

$$f(x) \neq \lim_{Q \to x} \frac{\int_Q f d\mu}{\mu(Q)}.$$

Consider the function f which is 0 for all points, except at the given point x, $f(x) = 1$. Clearly, f is polynomial space L_1-computable, but x does not satisfy the Lebesgue differentiation theorem.

5.1 Random points satisfy the Lebesgue differentiation theorem

The outline of our proof roughly follows that of the classical proof of the Lebesgue differentiation theorem [19, 22]. However, the restriction to polynomial space computation significantly changes the internal methods. We first show that if a point $x \in [0,1]^n$ is weakly pspace-random, then it must be contained in an open dyadic cube. This is a useful property of weakly pspace-random points that we take advantage of in later theorems.

▶ **Lemma 8.** *Let $x = (x_1, \dots, x_n) \in [0,1]^n$ be weakly pspace-random. Then, for every i, x_i is not a dyadic rational.*

Let f be a polynomial space L_1-computable function, approximated by the pspace computable sequence of simple step functions $\{f_m\}_{m\in\mathbb{N}}$. We now show that for every weakly pspace-random point x, the limit $\lim\limits_{m\to\infty} f_m(x)$ exists. We will need the following inequality due to Chebyshev. For every $f \in L_1([0,1]^n)$ and $\epsilon > 0$, define the set

$$S(f, \epsilon) = \{x \mid |f(x)| > \epsilon\}.$$

▶ **Chebyshev's Inequality.** *Let $f \in L_1([0,1]^n)$ and $\epsilon > 0$. Then $\mu(S(f,\epsilon)) \le \frac{\|f\|_1}{\epsilon}$.*

▶ **Lemma 9.** *Let $f \in L_1([0,1]^n)$ be polynomial space L_1 computable, approximated by the polynomial space computable sequence of simple step functions $\{f_m\}_{m\in\mathbb{N}}$. If x is weakly pspace-random, the limit $\lim\limits_{m\to\infty} f_m(x)$ exists.*

We now focus on the limit

$$\lim_{Q\to x} \frac{\int_Q f d\mu}{\mu(Q)}$$

on the right hand side of our main theorem (equation (3)). The restriction to polynomial space computation creates difficulties in considering arbitrary open cubes. Intuitively, we overcome this obstacle through the use of translations of dyadic cubes, which are more amenable to polynomial space computation. Formally, for $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$, define the set

$$\mathbf{B}_r^t = \{I_r^t \mid I_r^t = t + Q, \text{ where } Q \in \mathbf{B}_r\}.$$

That is, $\mathbf{B}_r^t$ is the set of all translations of dyadic cubes of precision r by points $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$. For every $x \in [0,1]^n$, let $I_r^t(x)$ denote the (unique) element of $\mathbf{B}_r^t$ containing x. The following theorem of Rute [20], using results due to Morayne and Solecki [16], shows that it suffices to prove that the right hand limit of equation (3) exists for these translations.

▶ **Theorem 10** ([20])**.** *Let $f \in L_1([0,1]^n)$, and $x \in [0,1]^n$. Then the following are equivalent,*

1. *the limit $\lim\limits_{Q\to x} \frac{\int_Q f d\mu}{\mu(Q)}$ exists, where the limit is taken over all cubes containing x, as the diameter goes to* 0
2. *the limit $\lim\limits_{k\to\infty} \frac{\int_{I_k^t(x)} f d\mu}{\mu(I_k^t(x))}$ exists, for all $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$.*

We now show that the limit

$$\lim_{m\to\infty} \frac{\int_{I_r^t(x)} |f - f_m| d\mu}{\mu(I_r^t(x))}$$

exists, for every $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$ and $r > 0$. We will need the following inequality due to Hardy and Littlewood. For every $f \in L_1([0,1]^n)$ and $\epsilon > 0$, define the set

$$T(f,\epsilon) = \{x \mid \sup_{r,t} \frac{\int_{I_r^t(x)} f d\mu}{\mu(I_r^t)} > \epsilon\},$$

where the supremum is taken over all $r > 0$ and $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$.

▶ **Hardy/Littlewood Inequality.** *There is a constant c such that, for every $f \in L_1([0,1]^n)$ and $\epsilon > 0$, $\mu(T(f,\epsilon)) \le \frac{c\|f\|_1}{\epsilon}$.*

▶ **Lemma 11.** *Let $f \in L_1([0,1]^n)$ be polynomial space L_1 computable, approximated by the polynomial space computable sequence of step functions $\{f_m\}_{m\in\mathbb{N}}$. If x is weakly pspace-random, then*

$$\lim_{m\to\infty} \frac{\int_{I_r^t(x)} |f - f_m| d\mu}{\mu(I_r^t(x))} = 0,$$

for every $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$ and $r > 0$.

We are now able to prove that weakly pspace random points satisfy the Lebesgue differentiation theorem.

▶ **Theorem 12.** *If x is weakly pspace-random, then for every polynomial space L_1-computable $f \in L_1([0,1]^n)$, and every polynomial space computable sequence of simple functions $\{f_m\}_{m\in\mathbb{N}}$ approximating f,*

$$\lim_{m\to\infty} f_m(x) = \lim_{Q\to x} \frac{\int_Q f d\mu}{\mu(Q)}$$

where the limit is taken over all cubes Q containing x as the diameter of Q tends to 0.

Proof. Let x be weakly pspace-random. By Theorem 10, it suffices to show that

$$\lim_{m\to\infty} f_m(x) = \lim_{k\to\infty} \frac{\int_{I_k^t(x)} f d\mu}{\mu(I_k^t(x))}$$

for all $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$.

Let $\epsilon > 0$. By Lemmas 9 and 11, there exists an N such that for all $i > N$,

$$|f_i(x) - \lim_{m\to\infty} f_m(x)| < \frac{\epsilon}{2}, \tag{4}$$

and

$$\frac{\int_{I_k^t(x)} |f - f_i| d\mu}{\mu(I_k^t(x))} < \frac{\epsilon}{2}, \tag{5}$$

for every $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$ and $k > 0$. Let $i > N$. Then, using (4) we obtain

$$|\lim_{m\to\infty} f_m(x) - \lim_{k\to\infty} \frac{\int_{I_k^t(x)} f d\mu}{\mu(I_k^t(x))}| < \frac{\epsilon}{2} + |f_i(x) - \lim_{k\to\infty} \frac{\int_{I_k^t(x)} f d\mu}{\mu(I_k^t(x))}|. \tag{6}$$

By Lemma 8, for every $r > 0$, $x \in Q$ for some $Q \in \mathbf{B}_r$. Since f_i is a simple step function, f_i is constant on every $Q \in \mathbf{B}_{p(i)}$. So there exists an N' so that for all $r > N'$,

$$f_i(x) = \frac{\int_{I_r^t(x)} f_i d\mu}{\mu(I_r^t(x))},$$

for every $t \in \{-\frac{1}{3}, 0, \frac{1}{3}\}^n$. Therefore, by inequality (5), for every $r > N'$,

$$\begin{aligned}
|f_i(x) - \frac{\int_{I_r^t(x)} f d\mu}{\mu(I_r^t(x))}| &= |\frac{\int_{I_r^t(x)} f_i d\mu}{\mu(I_r^t(x))} - \frac{\int_{I_r^t(x)} f d\mu}{\mu(I_r^t(x))}| && (7)\\
&\leq \frac{\int_{I_r^t(x)} |f - f_i| d\mu}{\mu(I_r^t(x))} && (8)\\
&< \frac{\epsilon}{2}. && (9)
\end{aligned}$$

Combining inequalities (6) and (9) we have

$$|\lim_{m\to\infty} f_m(x) - \lim_{k\to\infty} \frac{\int_{I_k^t(x)} f d\mu}{\mu(I_k^t(x))}| < \epsilon.$$

Since ϵ was arbitrary, the proof is complete. ◀

5.2 Non-random points are not Lebesgue points

We now show that converse of our main theorem holds. That is, we show that if a point x is not weakly pspace random, the limit $\lim\limits_{Q\to x} \frac{1}{\mu(Q)} \int_Q f d\mu$ does not exist. Our approach is largely similar from the construction of Pathak, et al [19]. However, due to the restriction of polynomial space computation, the implementation is significantly different. To adapt the construction of Pathak et al, we first introduce a notion that will partition a pspace $\mathcal{W}$-test $\{U_m\}$ into a tree of dyadic cubes.

Recall that the *level* of a node in a rooted tree is the length of the (unique) path from the root to the node. We denote the set of all nodes of a tree $\mathbf{T}$ at level i by $Level_i(\mathbf{T})$.

▶ **Definition 13.** A *dyadic tree decomposition of* $[0,1]^n$ is a tree $\mathbf{T}$ of dyadic cubes rooted at $[0,1]^n$ such that the following hold:

1. For every cube $Q \in \mathbf{T}$, the children of Q, are subsets of Q.
2. For any two cubes $Q_1, Q_2 \in \mathbf{T}$, either Q_1 and Q_2 are disjoint, or one contains the other.
3. For any cube $Q \in \mathbf{T}$,

$$\mu(\bigcup_{B\in Child(Q)} B) \leq \frac{\mu(Q)}{4}.$$

A dyadic tree decomposition $\mathbf{T}$ is *polynomial space approximable* if there exists a polynomial p and uniformly pspace computable array $\{T_m^k\}_{k,m\in\mathbb{N}}$ such that the following hold.

1. For every $k, m \in \mathbb{N}$, T_m^k is a finite union of disjoint dyadic cubes.
2. For every $\mu(Level_m(\mathbf{T})\Delta T_m^k) \leq 2^{-(k+m)}$.

Intuitively, for every k and m, T_m^k is a good approximation of the mth level of the tree $\mathbf{T}$.

We now show that every pspace $\mathcal{W}$-test admits a pspace approximable dyadic tree decomposition. We build the tree inductively, using the uniformly pspace computable sequence of the previous lemma.

▶ **Lemma 14.** *Let $\{U_m\}_{m\in\mathbb{N}}$ be a pspace $\mathcal{W}$-test. Then there exists a pspace approximable dyadic tree decomposition $\mathbf{T}$ such that, for every non-dyadic $x \in \bigcap U_m$, x is contained in an infinite path in $\mathbf{T}$.*

We are now able to prove the converse of Theorem 12, thereby completing the proof of our main theorem. The proof of this theorem involves constructing a function that takes advantage of the dyadic tree decomposition of a pspace W-test succeeding on x. We construct the function so that it assigns different values to alternating levels of the tree. As we are guaranteed that x is in an infinite path of the tree, the function oscillates around x.

▶ **Theorem 15.** *If $x \in [0,1]^n$ is not weakly pspace random, then there exists a pspace L_1 computable function f such that the limit $\lim\limits_{Q\to x} \frac{1}{\mu(Q)} \int_Q f d\mu$ does not exist.*

Proof. We first assume that $x = (x_1, \ldots, x_n)$ so that some component x_i of x is a dyadic rational. Without loss of generality assume that $x_1 = d \in \mathbf{D}$. Define the function $f : [0,1]^n \to \mathbb{R}$ to be

$$f(y) = \begin{cases} 1 & \text{if } y \in [0,d] \times [0,1] \times \ldots \times [0,1] \\ 0 & \text{otherwise} \end{cases}$$

It is clear that f is pspace L_1-computable, and that the limit $\lim\limits_{Q\to x} \frac{1}{\mu(Q)} \int_Q f d\mu$ does not exist.

Assume that $x = (x_1, \ldots, x_n)$ so that x_i is not a dyadic rational for all $i \le n$. Let $\{U_m\}_{m\in\mathbb{N}}$ be a pspace $\mathcal{W}$-test succeeding on x. Let $\mathbf{T}$ be a pspace computable dyadic tree partition of $\{U_m\}_{m\in\mathbb{N}}$ given by Lemma 14. Define $f : [0,1]^n \to \mathbb{R}$ as follows. For every $Q \in \mathbf{T}$,

$$f(Q - \bigcup_{B \in Child(Q)} B) = \begin{cases} 1 & \text{if the level of } Q \text{ in } \mathbf{T} \text{ is even} \\ 0 & \text{if the level of } Q \text{ in } \mathbf{T} \text{ is odd} \end{cases}$$

It is clear that f is integrable and well defined for all points that are not in the intersection $\bigcap U_m$. We now show that f is pspace L_1-computable. Let $\{T_m^k\}_{k,m\in\mathbb{N}}$ be the uniformly pspace computable array approximating $\mathbf{T}$. For every $m \in \mathbb{N}$, define

$$\mathbf{T}_m = \bigcup_{i=1}^{m} T_i^{m+2}.$$

We can consider $\mathbf{T}_m$ as a finite subtree of $\mathbf{T}$ which well approximates $\mathbf{T}$. For every $m \in \mathbf{N}$ and every $Q \in \mathbf{T}_m$, define the set of children of Q in the approximation $\mathbf{T}_m$ by

$$C_m(Q) = Child(Q) \cap \mathbf{T}_m.$$

For every $m \in \mathbb{N}$, define $f_m : [0,1]^n \to \mathbb{R}$ as follows.

$$f_m(Q - \bigcup_{B \in C_m(Q)} B) = \begin{cases} 1 & \text{if the level of } Q \text{ in } \mathbf{T}_m \text{ is even} \\ 0 & \text{if the level of } Q \text{ in } \mathbf{T}_m \text{ is odd} \end{cases}$$

It is clear that f_m is a simple step function. Since the array $\{T_m^k\}$ approximating $\mathbf{T}$ is uniformly pspace computable, on input $(0^m, d)$ we are able to compute the level of the largest dyadic cube in $\mathbf{T}$ containing d in polynomial space. Therefore the sequence of functions $\{f_m\}$ is pspace computable.

We now prove that $\{f_m\}_{m\in\mathbb{N}}$ approximates f. For $m \in \mathbb{N}$, define the set $A = \mathbf{T} - \mathbf{T}_m$, the set of all cubes in $\mathbf{T}$ that are not in the approximation $\mathbf{T}_m$. We now bound the error of our approximation $\mathbf{T}_m$. From the definition of tree decompositions, we have

$$\begin{aligned} \mu(A) &= \mu(\mathbf{T} - \mathbf{T}_m) \\ &= \mu(\bigcup_{i=1}^{m} Level_i(\mathbf{T}) - T_i^{m+2}) + \mu(\bigcup_{i=m+1}^{\infty} Level_i(\mathbf{T})) \\ &\le \sum_{i=1}^{m} \mu(Level_i(\mathbf{T}) - T_i^{m+2}) + \sum_{i=m+1}^{\infty} \mu(Level_i(\mathbf{T})) \\ &\le \sum_{i=1}^{m} 2^{-(i+m+2)} + \sum_{i=m+1}^{\infty} 2^{-2i} \\ &\le 2^{-m}. \end{aligned}$$

Therefore, we have

$$\begin{aligned} \|f - f_m\|_1 &= \int_0^1 |f - f_m| \\ &= \int_A |f - f_m| \\ &\le \mu(A) \\ &\le 2^{-m}. \end{aligned}$$

Hence, f is a pspace L_1 computable function.

Finally, we show that the limit $\lim\limits_{Q\to x} \frac{1}{\mu(Q)} \int_Q f d\mu$ does not exist. We first show that $\limsup\limits_{Q\to x} \frac{1}{\mu(Q)} \int_Q f d\mu \geq \frac{3}{4}$. Let $N \in \mathbb{N}$. By Lemma 14, x is contained in an infinite path of $\mathbf{T}$. Choose a dyadic cube $Q \in \mathbf{T}$ containing x so that $\mu(Q) < 2^{-N}$ and the level of Q in $\mathbf{T}$ is even. Then, by our construction of f,

$$\begin{aligned}\frac{1}{\mu(Q)} \int_Q f d\mu &\geq \frac{1}{\mu(Q)} \int_{Q-Child(Q)} 1 d\mu \\ &= \frac{1}{\mu(Q)} \mu(Q - Child(Q)) \\ &\geq \frac{3}{4}. \end{aligned} \tag{10}$$

Similarly, we show that $\liminf\limits_{Q\to x} \frac{1}{\mu(Q)} \int_Q f d\mu \leq \frac{1}{4}$. Let $N \in \mathbb{N}$. Choose a dyadic cube $Q \in \mathbf{T}$ containing x so that $\mu(Q) < 2^{-N}$ and the level of Q in $\mathbf{T}$ is odd. Then, by our construction of f,

$$\begin{aligned}\frac{1}{\mu(Q)} \int_Q f d\mu &\leq \frac{1}{\mu(Q)} \int_{Child(Q)} 1 d\mu \\ &= \frac{1}{\mu(Q)} \mu(Child(Q)) \\ &\leq \frac{1}{4}. \end{aligned} \tag{11}$$

Combining the equalities (10) and (11), we see that the limit $\lim\limits_{Q\to x} \frac{1}{\mu(Q)} \int_Q f d\mu$ does not exist. ◀

Finally, by Theorems 12 and 15, the Lebesgue differentiation theorem characterizes weakly pspace randomness.

6 Conclusion and Open Problems

In the computable setting, there is a strong connection between randomness and classical theorems of analysis. However, this interaction is not as well understood in the context of resource-bounded randomness. An interesting direction is to characterize randomness for different computational resource bounds using the Lebesgue differentiation theorem. For example, what notion of polynomial time randomness is characterized by the Lebesgue differentiation theorem?

We believe the notion of weakly polynomial space randomness will be useful in further investigations into resource-bounded randomness in analysis. An interesting avenue of future research is to relate weakly pspace-randomness with other notions of polynomial space randomness. We showed that Lutz's definition of pspace-randomness implies weakly pspace randomness, but the converse is not known. We conjecture that weakly pspace randomness is strictly weaker than Lutz's notion of pspace-randomness.

Acknowledgments. We thank Adam Case and Jack Lutz for useful discussions. We also thank the anonymous referees for many valuable suggestions.

References

1 Laurent Bienvenu, Adam R. Day, Mathieu Hoyrup, Ilya Mezhirov, and Alexander Shen, A constructive version of Birkhoff's Ergodic Theorem for Martin-Löf random points, *Inform. and Comput.* 210 (2012), 21–30.

2 V. Brattka, J. Miller, and A. Nies, Randomness and differentiability, *Transactions of the AMS* 368 (2016), 581–605.

3 Josef M. Breutzmann, David W. Juedes, and Jack H. Lutz, Baire category and nowhere differentiability for feasible real functions, *Mathematical Logic Quarterly* 50 (2004), pp. 460-472.

4 Rodney G. Downey and Denis R. Hirschfeldt, Algorithmic randomness and complexity, Theory and Applications of Computability, Springer, New York, 2010.

5 Johanna N.Y. Franklin, Noam Greenberg, Joseph S. Miller, and Keng Meng Ng, Martin-Löf random points satisfy Birkhoff's Ergodic Theorem for effectively closed sets, *Proceedings of the American Mathematical Society* 140 (2012).

6 C. Freer, B. Kjos-Hanssen, A. Nies, and F. Stephan, Algorithmic aspects of lipschitz functions, *Computability*, 3(1):45–61, 2014.

7 Alex Galicki and Daniel Turetsky, Randomness and differentiability in higher dimensions, (2014, submitted).

8 K. Ko, Computational Complexity of Real Functions, Birkhauser Boston, Boston, MA, 1991.

9 H. Lebesgue, Leçons sur l'Intégration et la recherche des fonctions primitives, Paris: Gauthier-Villars, 1904.

10 P. Martin-Löf, The definition of random sequences, *Information and Control*, 9:602-619, 1966.

11 Jack H. Lutz and Neil Lutz, Lines missing every random point, *Computability*, 4:85–102, 2015

12 Jack H. Lutz, Almost everywhere high nonuniform complexity, *Journal of Computer and System Sciences*, 44:220–258, 1992.

13 Jack H. Lutz, Resource-bounded measure, In *Proceedings of the 13th IEEE Conference on Computational Complexity*, pages 236-248, New York, 1998. IEEE Computer Society Press.

14 Kenshi Miyabe, Characterization of Kurtz randomness by a differentiation theorem, *Theory of Computing Systems*, 52(1):113–132, 2013.

15 K. Miyabe, A. Nies, and J. Zhang, Using almost-everywhere theorems from analysis to study randomness, (Submitted).

16 Michał Morayne and Sławomir Solecki, Martingale proof of the existence of Lebesgue points, *Real Anal. Exchange*, 15(1):401–406, 1989/90.

17 A. Nies, Differentiability of polynomial time computable functions, In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 602–613, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

18 Noopur Pathak, A computational aspect of the Lebesgue Differentiation Theorem, *Journal of Logic and Analysis* 1 (2009), no. 9, 15.

19 N. Pathak, C. Rojas, and S. G. Simpson, Schnorr randomness and the Lebesgue differentiation theorem, *Proc. Amer. Math. Soc.*, 142(1):335–349, 2014.

20 Jason Rute, Algorithmic randomness, martingales, and differentiation (draft).

21 Y. Wang, Randomness and Complexity, PhD thesis, University of Heidelberg, 1996.

22 R. Wheeden, A. Zygmund, Measure and Integral, Marcel Dekker, Inc, 1977.

Finding a Maximum 2-Matching Excluding Prescribed Cycles in Bipartite Graphs

Kenjiro Takazawa[*]

Department of Industrial and Systems Engineering, Faculty of Science and Engineering, Hosei University, Tokyo 184-8584, Japan
takazawa@hosei.ac.jp

Abstract

We introduce a new framework of restricted 2-matchings close to Hamilton cycles. For an undirected graph (V, E) and a family $\mathcal{U}$ of vertex subsets, a 2-matching F is called $\mathcal{U}$-feasible if, for each $U \in \mathcal{U}$, F contains at most $|U| - 1$ edges in the subgraph induced by U. Our framework includes $C_{\leq k}$-free 2-matchings, i.e., 2-matchings without cycles of at most k edges, and 2-factors covering prescribed edge cuts, both of which are intensively studied as relaxations of Hamilton cycles. The problem of finding a maximum $\mathcal{U}$-feasible 2-matching is NP-hard. We prove that the problem is tractable when the graph is bipartite and each $U \in \mathcal{U}$ induces a Hamilton-laceable graph. This case generalizes the $C_{\leq 4}$-free 2-matching problem in bipartite graphs. We establish a min-max theorem, a combinatorial polynomial-time algorithm, and decomposition theorems by extending the theory of $C_{\leq 4}$-free 2-matchings. Our result provides the first polynomially solvable case for the maximum $C_{\leq k}$-free 2-matching problem for $k \geq 5$. For instance, in bipartite graphs in which every cycle of length six has at least two chords, our algorithm solves the maximum $C_{\leq 6}$-free 2-matching problem in $\mathrm{O}(n^2 m)$ time, where n and m are the numbers of vertices and edges, respectively.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Optimization Algorithms, Matching Theory, Traveling Salesman Problem, Restricted 2-matchings, Hamilton-laceable Graphs

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.87

1 Introduction

The Hamilton cycle problem is one of the most fundamental NP-hard problems in various research fields such as graph theory, computational complexity, and combinatorial optimization. One successful approach to the Hamilton cycle problem is to utilize matching theory. In a graph $G = (V, E)$, an edge set $F \subseteq E$ is a 2-*matching* (resp., 2-*factor*) if it has at most (resp., exactly) two edges incident to each vertex in V. Since a Hamilton cycle is a special kind of 2-matching (or 2-factor) and a 2-matching of maximum size can be found in polynomial time, it is reasonable to put restrictions on 2-matchings to provide a tight relaxation of Hamilton cycles to which matching theory can be applied. Examples include the following two kinds of restricted 2-matchings:

$C_{\leq k}$-free 2-matchings. For a positive integer k, a 2-matching is called $C_{\leq k}$-*free* if it contains no cycles of length at most k. The larger k becomes, the closer a $C_{\leq k}$-free 2-factor becomes to a Hamilton cycle. If $k \geq |V|/2$, a $C_{\leq k}$-free 2-factor is a Hamilton cycle, whereas a $C_{\leq 2}$-free 2-matching is nothing other than a 2-matching.

[*] This work was partially supported by JSPS KAKENHI Grant Number 16K16012.

2-factors covering prescribed edge cuts. An *edge cut* is a minimal set of edges whose removal makes the graph disconnected. Given a family $\mathcal{K}$ of edge cuts, an edge subset is called $\mathcal{K}$*-covering* if it intersects every edge cut in $\mathcal{K}$. A Hamilton cycle is exactly a $\mathcal{K}$-covering 2-factor, where $\mathcal{K}$ is the family of all edge cuts.

Recently both $C_{\leq k}$-free and $\mathcal{K}$-covering 2-factors have been intensively studied and applied to designing approximation algorithms for NP-hard problems related to the Hamilton cycle problem, such as the graph-TSP and the minimum 2-edge connected spanning subgraph problem [4, 5, 8, 12, 20, 30, 32, 33].

1.1 Previous Work

In general graphs, the $C_{\leq k}$-free 2-matching problem is much more difficult than the 2-matching problem. For the cases $k \geq 3$, no algorithm is known other than Hartvigsen's $C_{\leq 3}$-free 2-matching algorithm [14]. NP-hardness for the case $k \geq 5$ is proved by Papadimitriou (see [7]). More generally, Hell et al. [17] proved that the problem is NP-hard, unless the excluded length of a cycle is a subset of $\{3, 4\}$. The case $k = 4$ is still open, and conjectured to be solvable in polynomial time [9]. Discrete convexity shown in [22] supports this conjecture.

While only a few positive results are known for the $C_{\leq k}$-free 2-matching problem in general graphs, in bipartite graphs the $C_{\leq 4}$-free 2-matching problem is efficiently solvable, and fundamental theorems in matching theory are extended. Motivated by a stimulating paper of Hartvigsen [15], Király [21] gave a min-max theorem for the $C_{\leq 4}$-free 2-matching problem in bipartite graphs, followed by a different min-max theorem by Frank [11]. Comparison of these two theorems is discussed in [31], together with decomposition theorems corresponding to the Dulmage-Mendelsohn and Edmonds-Gallai decompositions. Polynomial combinatorial algorithms are designed by Hartvigsen [16] and Pap [25], which are again slightly different and followed by an improvement in time complexity by Babenko [1]. For the weighted version, while the NP-hardness of the weighted $C_{\leq 4}$-free 2-matching problem in bipartite graphs is proved by Király (see [11]), positive results such as a linear programming formulation with dual integrality [23], a combinatorial algorithm [29], and discrete convexity [22] are established when the edge weights satisfy a certain property. Since the $C_{\leq 6}$-free 2-matching problem is NP-hard even in bipartite graphs [13], the $C_{\leq 4}$-free 2-matching problem in bipartite graphs is one of the few cases where the $C_{\leq k}$-free 2-matching problem is tractable.

For a set of positive integers $A \subseteq \mathbf{Z}$, denote the set of edge cuts whose sizes belong to A by $\mathcal{K}_A$. Kaiser and Škrekovski [18] proved that every bridgeless planar cubic graph has a $\mathcal{K}_{\{3,4\}}$-covering 2-factor, which is extended to a stronger result that every bridgeless cubic graph has a $\mathcal{K}_{\{3,4\}}$-covering 2-factor [19]. While the proof in [19] was not algorithmic, Boyd, Iwata, and Takazawa [4] designed a combinatorial algorithm for finding a $\mathcal{K}_{\{3,4\}}$-covering 2-factor in bridgeless cubic graphs, together with a combinatorial algorithm for finding a minimum-weight $\mathcal{K}_{\{3\}}$-covering 2-factor in bridgeless cubic graphs. Čada et al. [6] exhibited a family of graphs which has no $\mathcal{K}_{\{4,5\}}$-covering edge subset with even degree at every vertex, disproving a conjecture in [19].

1.2 Our Contribution

In the present paper, we introduce a new framework of restricted 2-matchings which commonly generalizes $C_{\leq k}$-free 2-matchings and $\mathcal{K}$-covering 2-factors. Let $G = (V, E)$ be a graph. For $U \subseteq V$, let $G[U] = (U, E[U])$ denote the subgraph induced by U, i.e., $E[U] = \{uv \in E\colon u, v \in U\}$. For $F \subseteq E$, let $F[U] = F \cap E[U] = \{uv \in F\colon u, v \in U\}$.

▶ **Definition 1** ($\mathcal{U}$-feasible 2-matching)**.** Let $\mathcal{U} \subseteq 2^V$ be a family of vertex subsets. A 2-matching $F \subseteq E$ is called *$\mathcal{U}$-feasible* if $|F[U]| \leq |U| - 1$ for each $U \in \mathcal{U}$.

Equivalently, a 2-matching F is $\mathcal{U}$-feasible if and only if F does not contain a 2-factor in $G[U]$ for each $U \in \mathcal{U}$. We remark that F does not only excludes a Hamilton cycle in $G[U]$, but also any 2-factor in $G[U]$ consisting of possibly multiple cycles.

If F is a 2-factor, then F is $\mathcal{U}$-feasible if and only if $F \cap \delta(U) \neq \emptyset$ for every $U \in \mathcal{U}$, where $\delta(U)$ denotes the set of edges having exactly one endpoint in U. From these viewpoints, it is not difficult to see that Hamilton cycles, $C_{\leq k}$-free 2-matchings, and $\mathcal{K}$-covering 2-factors are special cases of $\mathcal{U}$-feasible 2-factors or 2-matchings. That is, if we put $\mathcal{U} = \{U \subseteq V \colon |U| \leq |V|/2\}$ and $\mathcal{U} = \{U \subseteq V \colon \delta(U) \in \mathcal{K}\}$, then the set of $\mathcal{U}$-feasible 2-factors are exactly that of Hamilton cycles and $\mathcal{K}$-covering 2-factors, respectively. If putting $\mathcal{U} = \{U \subseteq V \colon |U| \leq k\}$, then the set of $\mathcal{U}$-feasible 2-factors is exactly that of $C_{\leq k}$-free 2-matchings.

The *$\mathcal{U}$-feasible 2-matching problem* is defined as a problem of finding a $\mathcal{U}$-feasible 2-matching of maximum size for given G and $\mathcal{U}$. In order to discuss the time complexity of the $\mathcal{U}$-feasible 2-matching problem, we should notice how $\mathcal{U}$ is given. In some cases, the size of $\mathcal{U}$ might be exponential in $|V|$, e.g., $\mathcal{U} = \{U \subseteq V \colon |U| \leq |V|/2\}$. Nevertheless, in many cases it is efficiently determined whether a given edge set is $\mathcal{U}$-feasible, such as the $C_{\leq k}$-free 2-matching case and the $\mathcal{K}$-covering 2-factor case. Therefore, we denote by γ the time for determining whether an edge set is $\mathcal{U}$-feasible, and we seek an algorithm with running time polynomial in $|V|$ and γ.

Since the Hamilton cycle problem is a special case of the $\mathcal{U}$-feasible 2-matching problem, the $\mathcal{U}$-feasible 2-matching problem is NP-hard in general. Thus, we need some assumption in order to obtain a tractable class of the $\mathcal{U}$-feasible 2-matching problem, such as the cases where G is bipartite and $\mathcal{U} = \{U \subseteq V \colon |U| \leq 4\}$, and G is bridgeless cubic and $\mathcal{U} = \{U \subseteq V \colon \delta(U) \in \mathcal{K}_{\{3,4\}}\}$.

A main objective of this paper is to provide a broader tractable class of the $\mathcal{U}$-feasible 2-matching problem by extending the theory of $C_{\leq 4}$-free 2-matchings in bipartite graphs. For this purpose, we exploit a graph-theoretic concept of *Hamilton-laceable graphs.* For a bipartite graph (V, E), we denote the two color classes by V^+ and V^-. For $X \subseteq V$, let $X^+ = X \cap V^+$ and $X^- = X \cap V^-$.

▶ **Definition 2** (Hamilton-laceable graph [26])**.** A bipartite graph $G = (V, E)$ is *Hamilton-laceable* if (i) $|V^+| = |V^-|$ and G has a Hamilton path between an arbitrary pair of $u \in V^+$ and $v \in V^-$, or (ii) $|V^+| = |V^-| - 1$ and G has a Hamilton path between an arbitrary pair of distinct vertices $u, v \in V^-$.

In what follows, we work on the $\mathcal{U}$-feasible 2-matching problem under the assumption that G is bipartite and $G[U]$ is Hamilton-laceable for each $U \in \mathcal{U}$. We note that, for a 2-factor F, $|F[U]| = |U|$ implies that $|U^+| = |U^-|$. Thus, we assume $|U^+| = |U^-|$ for each $U \in \mathcal{U}$, and hence only the case (i) in Definition 2 occurs in our arguments.

If we take into account the original motivation, i.e., finding a 2-factor close to a Hamilton cycle, then $\mathcal{U}$ should be a vertex set family as large as possible. Thus a natural setting would be to define $\mathcal{U}$ as the family of all vertex sets $U \subseteq V$ such that $G[U]$ is Hamilton-laceable. This indeed provides a new framework of 2-matchings closer to Hamilton cycles.

Furthermore, several types of the $C_{\leq k}$-free 2-matching problem are described as the $\mathcal{U}$-feasible 2-matching problem under our assumption. The smallest nontrivial example of a Hamilton-laceable graph would be a cycle of length four, and hence the $C_{\leq 4}$-free 2-matching problem in bipartite graphs is a special case of the $\mathcal{U}$-feasible 2-matching problem under our assumption. As for $C_{\leq 6}$-free-free 2-matchings, a cycle of length six is Hamilton-laceable if it has at least two chords. Thus, the $C_{\leq 6}$-free 2-matching problem in bipartite graphs in which every cycle of length six has at least two chords is described as the $\mathcal{U}$-feasible 2-matching problem under our assumption. In other words, in our setting a solution (a $\mathcal{U}$-feasible 2-matching) might contain a cycle of length six with at most one chord, but it can exclude all the cycles of length six with at least two chords. Further examples and previous work of Hamilton-laceable graphs are exhibited in § 2.

In the present paper, we exhibit that the theory of $C_{\leq 4}$-free 2-matching problem in bipartite graphs satisfactorily extends when $G[U]$ is Hamilton-laceable for each $U \in \mathcal{U}$. We first present a min-max theorem extending Király's min-max theorem [21]. We then design a combinatorial algorithm for finding a maximum $\mathcal{U}$-feasible 2-matching, which provides a constructive proof for our min-max theorem. In the design of our algorithm, we make use of both of Hartvigsen's and Pap's algorithms [16, 25]: the shrinking technique comes from Pap's algorithm; and the construction of a minimizer of the min-max theorem derives from Hartvigsen's method. Finally, we describe decomposition theorems extending those in [31] and corresponding to the Dulmage-Mendelsohn and Edmonds-Gallai decompositions.

Here we summarize our algorithmic results. We denote the number of vertices and edges in the input graph by n and m, respectively. Recall that γ is the time for determining whether an edge set is $\mathcal{U}$-feasible.

▶ **Theorem 3.** *Let $G = (V, E)$ be a bipartite graph and $\mathcal{U} \subseteq 2^V$ be a family of vertex subsets such that $G[U]$ is Hamilton-laceable for each $U \in \mathcal{U}$. Then a $\mathcal{U}$-feasible 2-matching of maximum size in G can be found in $\mathrm{O}(n^3\gamma + n^2 m)$ time.*

We remark that, when our algorithm is applied to the $C_{\leq k}$-free 2-matching case, i.e., the case $\mathcal{U} = \{U \subseteq V : |U| \leq k,\ G[U] \text{ is Hamilton-laceable}\}$, γ becomes the time for determining if a specified edge is contained in a cycle of length at most k in a given 2-matching, and thus $\gamma = \mathrm{O}(k)$. (The detail is described in § 4.3.) Therefore, the following theorem is established.

▶ **Theorem 4.** *The $C_{\leq k}$-free 2-matching problem in bipartite graphs is solvable in $\mathrm{O}(kn^3 + n^2 m)$ time if every cycle of length at most k induces a Hamilton-laceable graph.*

In particular, it holds that $\gamma = \mathrm{O}(1)$ if k is a constant. By setting $\mathcal{U} = \{U \subseteq V : |U| \leq 4,\ G[U] \text{ is Hamilton-laceable}\}$, we can see that Theorem 3 extends the solvability of the $C_{\leq 4}$-free 2-matching problem in bipartite graphs. Moreover, setting $\mathcal{U} = \{U \subseteq V : |U| \leq 6,\ G[U] \text{ is Hamilton-laceable}\}$ in Theorem 3 leads to the following corollaries on the $C_{\leq 6}$-free 2-matching problem in bipartite graphs.

▶ **Corollary 5.** *In a bipartite graph, a maximum 2-matching excluding any cycle of length six with at least two chords and any cycle of length four can be found in $\mathrm{O}(n^2 m)$ time.*

▶ **Corollary 6.** *In a bipartite graph in which every cycle of length six has at least two chords, the $C_{\leq 6}$-free-free 2-matching problem can be solved in $\mathrm{O}(n^2 m)$ time.*

To the best of our knowledge, Corollary 6 is the first polynomially solvable case of the $C_{\leq 6}$-free 2-matching problem. Furthermore, combined with Lemma 8 in § 2, Theorem 3 leads to the following corollary, an extension of Corollary 6.

▶ **Corollary 7.** *The $C_{\leq k}$-free 2-matching problem in bipartite graphs is solvable in $\mathrm{O}(kn^3 + n^2 m)$ time if every cycle of length $2t$ such that $2t \leq k$ has at least $(t-1)(t-2)$ chords.*

Note that Corollary 6 is exactly the case $k = 6$ of Corollary 7.

It is noteworthy that, unlike the literature of $C_{\leq k}$-free 2-matchings and $\mathcal{K}$-covering 2-factors, our assumption that each $G[U]$ is Hamilton-laceable does not depend on the size of the forbidden structures. As stated above, one benefit of this is that our result provides the first polynomially solvable case of the $C_{\leq k}$-free 2-matching problem for $k \geq 5$, and thus has a potential to provide better approximation ratios for the graph-TSP and the minimum 2-edge connected subgraph problem.

We further remark that our framework contains the both cases where multiplicities on edges are forbidden and allowed. That is, in the former case we only deal with simple 2-matchings and one edge can only contribute one to the degree of its endpoints. In the latter case, we can put multiplicity two on some edges. In the literature of the $C_{\leq k}$-free 2-matching problem, these two cases have formed different streams. The aforementioned results are of the former case, and results for the latter case include [2, 7, 24]. To the best of our knowledge, not much connection between these two cases is found. In our framework, forbidding multiplicity on an edge $uv \in E$ corresponds to having $\{u, v\}$ in $\mathcal{U}$, and it is clear that $G[\{u, v\}]$ is Hamilton-laceable if $uv \in E$. While in this paper we mainly keep the former case in mind, we note that our framework can represent both cases.

1.3 Organization of the Paper

The rest of the paper is organized as follows. In § 2, we present some previous work, observations, and examples of Hamilton-laceable graphs. After that, we exhibit our contribution on $\mathcal{U}$-feasible 2-matchings in bipartite graphs where $G[U]$ is Hamilton-laceable for each $U \in \mathcal{U}$. We present a min-max theorem in § 3. Section 4 is devoted to describing a combinatorial algorithm for finding a maximum $\mathcal{U}$-feasible 2-matching, which provides a constructive proof for the min-max theorem. In § 5, we exhibit decomposition theorems corresponding to the Dulmage-Mendelsohn and Edmonds-Gallai decompositions. In § 6, we demonstrate an application of our framework by showing that a regular bipartite graph admit a certain kind of $\mathcal{U}$-feasible 2-factor. Section 7 concludes this paper.

2 Hamilton-Laceable Graph

This section is devoted to a discussion on Hamilton-laceable graphs. We first note that the concept of Hamilton-laceable graphs is a bipartite analogue of that of *Hamilton-connected graphs*, which is well-known in the field of graph theory [3]. A graph is *Hamilton-connected* if it has a Hamilton path between an arbitrary pair of distinct vertices. Thus, a Hamilton-connected graph is nonbipartite if it has at least three vertices.

In what follows, we always assume that $G = (V, E)$ is bipartite. Trivial examples of a Hamilton-laceable graph are a graph of a single vertex, and a graph of two vertices connected by an edge. It is also clear that a complete bipartite graph on $2t$ vertices, denoted by $K_{t,t}$, is Hamilton-laceable. Recall that a special case $K_{2,2}$, a cycle of length four, is an example of a Hamilton-laceable graph.

If $G = (V, E)$ is Hamilton-laceable, a graph $(V, \tilde{E})$ satisfying $\tilde{E} \supseteq E$ is also Hamilton-laceable. Thus, it would be of interest to find Hamilton-laceable graphs with as few edges as possible. Indeed, the concept of Hamilton-laceable graphs was introduced as a generalized property of Hamiltonicity of *d-dimensional rectangular lattices* by Simmons [26]. A d-dimensional rectangular lattice is a graph (V, E) represented by d positive integers $a_1, \ldots, a_d$ as $V = \{x \in \mathbf{Z}^d \colon 0 \leq x_i \leq a_i,\ i = 1, \ldots, d\}$ and $E = \{xy \colon x, y \in V,\ \sum_{i=1}^{d} |x_i - y_i| = 1\}$. Simmons [26] proved that all d-dimensional rectangular lattices are Hamilton-laceable except for the two-dimensional lattices of order $2 \times r$ ($r \neq 2$) and $3 \times 2r$. This result provides a class of Hamilton-laceable graphs (V, E) with $|E| \approx d|V|$. For instance, every hypercube is

Hamilton-laceable. The following lemma also provides a sufficient condition for a graph to be Hamilton-laceable.

▶ **Lemma 8** (Simmons [28]). *Deleting fewer than $t-1$ edges from $K_{t,t}$ or $K_{t,t+1}$ maintains Hamilton-laceability.*

Furthermore, Simmons [27] discussed the minimum number l_t of the edges of Hamilton-laceable graphs with $|V^+| = t$. It holds that $3t - \lceil t/3 \rceil \leq l_t \leq 3t - 1$ for the case (i) in Definition 2, and $l_t = 3t + 1$ for the case (ii) in Definition 2.

The motivation of introducing Hamilton-laceable graph in this paper comes from an analysis in [31], which reveals that cycles of length four in the $C_{\leq 4}$-free 2-matching problem in bipartite graphs serve as factor-critical components for the nonbipartite matching problem: if $U \subseteq V$ induces a cycle of length four in a bipartite graph, for an arbitrary pair $u \in U^+$ and $v \in U^-$, $G[U]$ contains a 2-matching of size three in which only u and v have degree one. Indeed, this is the property which makes it possible to execute the shrinking and expanding procedures in the algorithms in [16, 25], which is shed light on by a decomposition theorem [31] resembling the Edmonds-Gallai decomposition. Observe that the definition of Hamilton-laceable graphs generalizes the above property of C_4. In the following sections we reveal that the property in Definition 2(i) plays a key role to provide a tractable class of restricted 2-matchings in bipartite graphs.

3 Min-Max Theorem

In this section, we describe a min-max theorem for the $\mathcal{U}$-feasible 2-matching problem in bipartite graphs where each $U \in \mathcal{U}$ induces a Hamilton-laceable graph. Our theorem is an extension of Király's min-max theorem [21] for the $C_{\leq 4}$-free 2-matching problem in bipartite graphs. For $X \subseteq V$, let $\bar{X} = V \setminus X$ and $c'(X)$ denote the number of components in $G[X]$ consisting of a single vertex, a single edge, or a single cycle of length four.

▶ **Theorem 9** ([21]). *Let $G = (V, E)$ be a bipartite graph. Then, it holds that*

$$\max\{|F| \colon F \textit{ is a } C_{\leq 4}\textit{-free } 2\textit{-matching}\} = \min\{|V| + |X| - c'(\bar{X}) \colon X \subseteq V\}.$$

Observe that every component contributing to $c'(\bar{X})$ is Hamilton-laceable. We now exhibit our theorem extending Theorem 9. For $X \subseteq V$, let $c(X)$ denote the number of components in $G[X]$ whose vertex set belongs to $\mathcal{U}$.

▶ **Theorem 10.** *Let $G = (V, E)$ be a bipartite graph and $\mathcal{U} \subseteq 2^V$ be a family of vertex subsets in G such that $G[U]$ is Hamilton-laceable for each $U \in \mathcal{U}$. Then, it holds that*

$$\max\{|F| \colon F \textit{ is a } \mathcal{U}\textit{-feasible } 2\textit{-matching}\} = \min\{|V| + |X| - c(\bar{X}) \colon X \subseteq V\}. \tag{1}$$

It is not difficult to see that Theorem 9 is indeed a special case of Theorem 10 where $\mathcal{U} = \{U \subseteq V \colon |U| \leq 4,\ G[U] \text{ is Hamilton-laceable}\}$.

Before proving Theorem 10, we first show that the inequality max $\leq$ min in (1) holds for an arbitrary G and $\mathcal{U}$, i.e., G may not be bipartite and $G[U]$ may not be Hamilton-laceable for $U \in \mathcal{U}$. For disjoint vertex sets $X, Y \subseteq V$, let $E[X, Y]$ denote the set of edges connecting X and Y, $G[X, Y] = (X \cup Y, E[X, Y])$, and $F[X, Y] = F \cap E[X, Y]$ for $F \subseteq E$.

▶ **Lemma 11.** *Let $G = (V, E)$ be a graph and $\mathcal{U} \subseteq 2^V$ be a family of vertex subsets in G. For an arbitrary $\mathcal{U}$-feasible 2-matching F and $X \subseteq V$, it holds that $|F| \leq |V| + |X| - c(\bar{X})$.*

Proof. Since F is a 2-matching, $2|F[X]| + |F[X, \bar{X}]| \leq 2|X|$ follows. Moreover, since F is $\mathcal{U}$-feasible, it holds that $|F[\bar{X}]| \leq |\bar{X}| - c(\bar{X})$. Therefore, $|F| = |F[X]| + |F[X, \bar{X}]| + |F[\bar{X}]| \leq 2|F[X]| + |F[X, \bar{X}]| + |F[\bar{X}]| \leq 2|X| + |\bar{X}| - c(\bar{X}) = |V| + |X| - c(\bar{X})$. ◀

The following lemma directly follows from the proof for Lemma 11. For $F \subseteq E$ and $u \in V$, denote the number of edges in F incident to u by $\deg_F(u)$.

▶ **Lemma 12.** *If a $\mathcal{U}$-feasible 2-matching F and $X \subseteq V$ attain the equality in* (1)*, it holds that*

- $F[X] = \emptyset$,
- $\deg_{F[\{u\}, \bar{X}]}(u) = 2$ *for each* $u \in X$*, and*
- *for each component* Q *in* $G[\bar{X}]$,

$$|F[V(Q)]| = \begin{cases} |V(Q)| - 1 & \text{if } V(Q) \in \mathcal{U}, \\ |V(Q)| & \text{otherwise.} \end{cases}$$

Proof. By the above proof for Lemma 11, it should hold that $|F[X]| = 0$, $|F[X, \bar{X}]| = 2|X|$, and $|F[\bar{X}]| = |\bar{X}| - c(\bar{X})$ for a $\mathcal{U}$-feasible 2-matching F and $X \subseteq V$ attaining the equality in (1). These respectively lead to the statements in the lemma. ◀

In § 4, we complete a proof of Theorem 10 by establishing an algorithm for finding a $\mathcal{U}$-feasible 2-matching F and $X \subseteq V$ attaining equality in (1). It should be noted that the bipartiteness of G and Hamilton-laceability of $G[U]$ for each $U \in \mathcal{U}$ play an important role in the algorithm, and thus they are key properties to achieving equality in (1) as well.

4 Combinatorial Algorithm

In this section, we describe a combinatorial polynomial-time algorithm for finding a maximum $\mathcal{U}$-feasible 2-matching in bipartite graphs where each $U \in \mathcal{U}$ induces a Hamilton-laceable graph. Our algorithm employs ideas of both of the $C_{\leq 4}$-free 2-matching algorithms of Hartvigsen [16] and Pap [25].

4.1 Algorithm Description

Roughly speaking, our algorithm resembles Edmonds' algorithm for nonbipartite matchings [10]. One main feature in our algorithm comes from Pap's algorithm [25]: we shrink $U \in \mathcal{U}$ after we find an alternating path, whereas in Edmonds' and Hartvigsen's algorithms shrinking occurs in the middle of construction of alternating forests. Another feature derives from Hartvigsen's algorithm [16]. A minimizer $X \subseteq V$ of the right-hand side of (1) is basically determined as the set of vertices reachable from the deficient vertices, vertices having at most one incident edge in the optimal solution. In Hartvigsen's and our algorithms, if a vertex resulting from shrinking $U \in \mathcal{U}$ satisfies certain properties, it is regarded as reachable even if it is not reachable.

Before describing the entire algorithm, we present how to shrink and expand $U \in \mathcal{U}$. In order to provide concise notation, in the rest of this section we denote the input of the algorithm by $\hat{G} = (\hat{V}, \hat{E})$ and $\hat{\mathcal{U}} \subseteq 2^{\hat{V}}$, and the graph obtained by repeated shrinkings by $G = (V, E)$. Following standard notation, for a vector $b \in \mathbf{Z}^V$, an edge set $F \subseteq E$ is called a *b-matching* if every vertex $v \in V$ is incident to at most $b(v)$ edges in F. If every vertex $v \in V$ is incident to exactly $b(v)$ edges in F, then F is called a *b-factor*. If $b(v) = t$ for every $v \in V$,

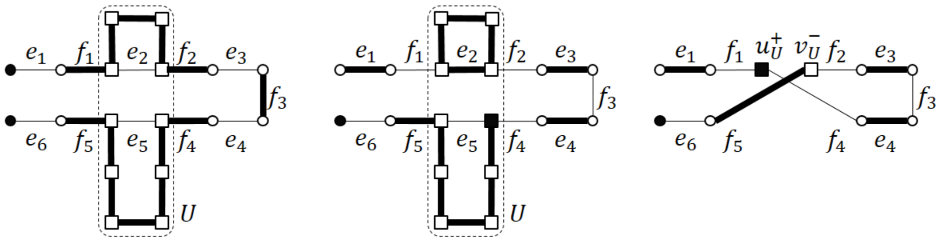


Figure 1 $b_v = 2$ for each v. The thick edges are in F, thin edges in $E \setminus F$, and the vertices in black are in S^+ or S^-. The ten vertices represented by squares form $U \in \mathcal{U}$. In the figure on the left, we have found P consisting of $e_1, f_1, e_2, \ldots, e_5, f_5, e_6$, and $F \triangle (E(P))$ contains a 2-factor in $G[U]$ for $U \in \mathcal{U}$. In this case $i^* = 5$, and the figure in the middle shows $F \triangle (E(P_4))$. The figure on the right shows the graph after $\mathsf{Shrink}(F, P)$.

then a b-matching is simply referred to as a t-matching. Note that this notation is compatible with our definition of 2-matchings.

In the algorithm, we maintain a $\mathcal{U}$-feasible b-matching F in G, where $\mathcal{U} \subseteq 2^V$ and $b \in \{1,2\}^V$, which can be extended to a $\hat{\mathcal{U}}$-feasible 2-matching in $\hat{G}$. For $b \in \{1,2\}^V$, a b-matching F is *$\mathcal{U}$-feasible* if $F[U]$ is not a b_U-factor in $G[U]$ for every $U \in \mathcal{U}$, where b_U is the restriction of b to U. Initially, $G = \hat{G}$, $\mathcal{U} = \hat{\mathcal{U}}$, $b_v = 2$ for each $v \in V$, and F is an arbitrary $\mathcal{U}$-feasible b-matching, e.g., $F = \emptyset$.

For $F_1, F_2 \subseteq E$, denote the symmetric difference of F_1 and F_2 by $F_1 \triangle F_2$, i.e., $F_1 \triangle F_2 = (F_1 \setminus F_2) \cup (F_2 \setminus F_1)$. Define the set of source vertices by $S^+ = \{u \in V^+ \colon \deg_F(u) < b_u\}$ and sink vertices $S^- = \{v \in V^- \colon \deg_F(v) < b_v\}$. Suppose that we have found an alternating path P with respect to F and $E \setminus F$ such that P starts in S^+ and ends in S^-, and $F \triangle (E(P))$ is not a $\mathcal{U}$-feasible b-matching. We then apply the following shrinking procedure.

Procedure Shrink(F, P). Denote $E(P) = \{e_1, f_1, e_2, \ldots, e_l, f_l, e_{l+1}\}$, where the edges are sorted by the order of appearance in P. Note that $e_j \in E \setminus F$ $(j = 1, \ldots, l+1)$ and $f_j \in F$ $(j = 1, \ldots, l)$. Let P_i be the path consisting of $\bigcup_{j=1}^{i} \{e_j, f_j\}$ for $i = 1, \ldots, l$, P_0 be an empty graph, and $P_{l+1} = P$. Let i^* be the smallest index i such that $F \triangle (E(P_i))$ contains a b-factor in $G[U]$ for some $U \in \mathcal{U}$, and let $F' = F \triangle (E(P_{i^*-1}))$. If more than one such $U \in \mathcal{U}$ exists, choose an arbitrary U. We then update G, b, $\mathcal{U}$, and F as follows. Let u_U^+ and v_U^- be new vertices obtained by contracting the vertices in U^+ and U^-, respectively. Then, reset

$$V := \bar{U} \cup \{u_U, v_U\}, \quad b_v := \begin{cases} 1 & \text{if } v = u_U^+, v_U^-, \\ b_v & \text{otherwise,} \end{cases}$$

$$E := E[\bar{U}] \cup \{u_U^+ v \colon uv \in E, u \in U^+, v \in \bar{U}^-\} \cup \{u v_U^- \colon uv \in E, u \in \bar{U}^+, v \in U^-\},$$

$$F := F'[\bar{U}] \cup \{u_U^+ v \colon uv \in F', u \in U^+, v \in \bar{U}^-\} \cup \{u v_U^- \colon uv \in F', u \in \bar{U}^+, v \in U^-\},$$

$$\mathcal{U} := \{U' \colon U' \in \mathcal{U}, U' \cap U = \emptyset\} \cup \{(U' \setminus U) \cup \{u_U^+, v_U^-\} \colon U' \in \mathcal{U}, U \subsetneq U'\}.$$

See Figure 1 for an illustration. Observe that the update preserves that G is bipartite and F is still a b-matching in G. We then repeat the above procedure.

If an alternating path P from S^+ to S^- is found such that $F \triangle (E(P))$ is a $\mathcal{U}$-feasible b-matching, then we reset $F := F \triangle (E(P))$ to augment the current solution, and expand the shrunk vertex sets to return to the original graph $\hat{G}$ as follows. First note that the shrunk vertex sets in $\hat{\mathcal{U}}$ form a laminar family, and it suffices to expand the maximal shrunk vertex sets. Let $\mathcal{U}^* \subseteq 2^{\hat{V}}$ be the family of maximal shrunk vertex sets. For a maximal shrunk

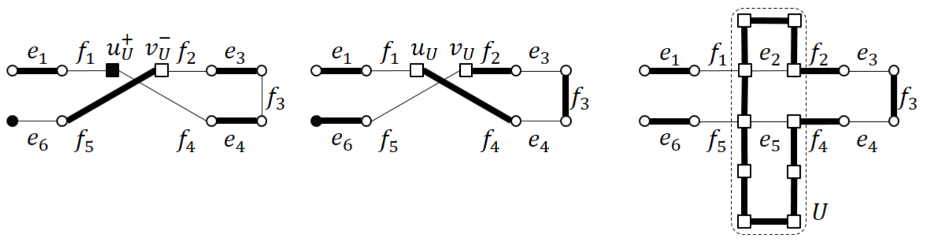


Figure 2 The graph in the middle results from an augmentation in the graph on the left, where the augmenting path P consists of $f_4, e_4, f_3, e_3, f_2, f_5, e_6$. We then expand U, where $\hat{f}_U^+ = f_4$ and $\hat{f}_U^- = f_2$. to obtain the graph on the right.

vertex set $U \subseteq \hat{V}$, denote the unique edge in F incident to u_U^+ by f_U^+, and to v_U^- by f_U^-, if exist. Let $\hat{f}_U^+, \hat{f}_U^- \in \hat{E}$ be the edges corresponding to $f_U^+, f_U^- \in E$, respectively. Denote the vertex in U^+ incident to $\hat{f}_U^+$ by $\hat{u}_U^+$, and that in U^- incident to $\hat{f}_U^-$ by $\hat{v}_U^-$. If f_U^+ (resp., f_U^-) does not exist, let $\hat{u}_U^+$ (resp., $\hat{v}_U^-$) be an arbitrary vertex in U^+ (resp., U^-). Now, since $\hat{G}[U]$ is Hamilton-laceable, $\hat{G}[U]$ has a Hamilton path P_U between $\hat{u}_U^+$ and $\hat{v}_U^-$. In expanding U, we add $E(P_U)$ to F. That is, $\hat{F} := F \cup \bigcup_{U \in \mathcal{U}^*} E(P_U)$. See Figure 2 for an illustration of augmentation and expansion. It is not difficult to see that $\hat{F}$ is a $\mathcal{U}$-feasible 2-matching.

The entire algorithm is described as follows.

Input: A bipartite graph $\hat{G} = (\hat{V}, \hat{E})$ and $\hat{\mathcal{U}} \subseteq 2^{\hat{V}}$ such that $\hat{G}[U]$ is Hamilton-laceable for each $U \in \hat{\mathcal{U}}$.

Output: A maximum $\hat{\mathcal{U}}$-feasible 2-matching $\hat{F}$ in $\hat{G}$.

Step 0: Put $G = \hat{G}$ and $\mathcal{U} = \hat{\mathcal{U}}$. Let F be an arbitrary $\mathcal{U}$-feasible 2-matching in G. Let F be an arbitrary $\mathcal{U}$-feasible 2-matching in G and then go to Step 1.

Step 1: Let $S^+ = \{u \in V^+ : \deg_F(u) < b_u\}$ and $S^- = \{v \in V^- : \deg_F(v) < b_v\}$. Orient each edge in $E \setminus F$ from V^+ to V^- and each edge in F from V^- to V^+ to obtain a directed graph D. If D has a directed path P from S^+ to S^-, then go to Step 2. Otherwise, go to Step 5.

Step 2: Let $E_P \subseteq E$ be the set of edges corresponding to the directed edges in P. If $F' = F \triangle E_P$ is a $\mathcal{U}$-feasible b-matching, then go to Step 3. Otherwise, go to Step 4.

Step 3 (Augmentation): Reset $F := F'$, expand all maximal shrunk vertex sets, and then go back to Step 1.

Step 4 (Shrinking): Apply Shrink(F, P), and then go back to Step 1.

Step 5 (Termination): Expand all maximal shrunk vertex sets and return $\hat{F}$.

4.2 Proof for Correctness

At the termination of the algorithm, we have a digraph D in which no directed path from S^+ to S^- exists. Let $R \subseteq V$ denote the set of vertices reachable from S^+ in D, and define $R' \subseteq V$ by

$$\begin{aligned} R' = R &\cup \{v \in (\bar{R})^- : v \text{ is not a shrunk vertex}, \deg_{F[R^+, \{v\}]}(v) = 2\} \\ &\cup \{v \in (\bar{R})^- : v = v_U^- \text{ for some } U \in \mathcal{U},\ uv \in F \text{ for some } u \in R^+\}. \end{aligned}$$

Finally, define $X \subseteq \hat{V}$ by the set of vertices corresponding to $(\overline{R'})^+ \cup (R')^-$, i.e.,

$$\begin{aligned} X = &\{u \in \hat{V}^+ : u \in (\overline{R'})^+ \text{ or } u \in U \text{ for some } U \in \mathcal{U} \text{ with } u_U \in (\overline{R'})^+\} \\ &\cup \{v \in \hat{V}^- : v \in (R')^- \text{ or } v \in U \text{ for some } U \in \mathcal{U} \text{ with } v_U \in (R')^-\}. \end{aligned}$$

▶ **Lemma 13.** *The output $\hat{F}$ and X defined above attain the equality in* (1).

Proof. It is not difficult to see that $\hat{F}[X] = \emptyset$. Moreover, since every $v \in X$ satisfies $\deg_{\hat{F}[\{v\},\bar{X}]} = 2$, we have that $|\hat{F}[X, \bar{X}]| = 2|X|$. Finally, since R is defined by reachability from S^+ in D, all edges in $E[\bar{X}]$ belong to F in G. Thus, each edge in $\hat{E}[\bar{X}]$ is in $\hat{F}$ or belongs to $\hat{E}[U]$ for some $U \in \mathcal{U}$ shrunk in G. By the definition of R', it holds that v_U^- has no adjacent edge in $E[\bar{X}]$, which implies that $\hat{G}[U]$ forms a component in $\hat{G}[\bar{X}]$. Thus, it follows that $|\hat{F}[\bar{X}]| = |\bar{X}| - c(\bar{X})$. Therefore, $|\hat{F}| = |\hat{F}[X]| + |\hat{F}[X, \bar{X}]| + |\hat{F}[\bar{X}]| = 2|X| + |\bar{X}| - c(\bar{X}) = |V| + |X| - c(\bar{X})$. ◀

Now Theorem 10 immediately follows from Lemmas 11 and 13. Thus, our algorithm provides a constructive proof for Theorem 10.

4.3 Complexity

Recall that $n = |\hat{V}|$, $m = |\hat{E}|$, and γ is the time for determining if an edge set is $\hat{\mathcal{U}}$-feasible. It is not difficult to see that shrinkings occur $\mathrm{O}(n)$ times between augmentations. Since augmentations occur $\mathrm{O}(n)$ times, shrinkings occur $\mathrm{O}(n^2)$ times in total.

After each shrinking, we search an alternating path, which takes $\mathrm{O}(m)$ time. Moreover, we determine if $F \triangle (E(P_i))$ is $\mathcal{U}$-feasible $\mathrm{O}(n)$ times for each shrinking. The time for this determination is γ in general. If we consider the $C_{\leq k}$-free 2-matching problem, then it suffices to determine if e_i is contained in a cycle of length at least k in $F \triangle (E(P_i))$, which takes $\mathrm{O}(k)$ time. Thus, the time complexity between shrinkings is $\mathrm{O}(n\gamma + m)$ in general, and is $\mathrm{O}(kn + m)$ for the $C_{\leq k}$-free 2-matching case. Therefore, Theorems 3 and 4 are established.

5 Decomposition Theorems

This section is devoted to decomposition theorems for the $\mathcal{U}$-feasible 2-matching problem in bipartite graphs where each $U \in \mathcal{U}$ induces a Hamilton-laceable graph. These theorems correspond to the Dulmage-Mendelsohn and Edmonds-Gallai decompositions, and extend decomposition theorems for the $C_{\leq 4}$-free 2-matchings in bipartite graphs [31]. Proofs for the theorems in this section will appear in a full version of this paper.

Let $X_1 \subseteq V$ be a minimizer of (1) obtained by the algorithm in § 4. By exchanging the roles of V^+ and V^-, i.e., searching alternating paths from S^- to S^+, we obtain another minimizer $X_2 \subseteq V$ of (1). Now partition V into three sets $D, A, C \subseteq V$, where $D = \bar{X}_1^+ \cup \bar{X}_2^-$, $A = X_2^+ \cup X_1^-$, and $C = V \setminus (D \cup A)$.

Now the following theorems are established. Theorem 14 provides a characterization of D. Note that such a characterization appears in both of the Dulmage-Mendelsohn and Edmonds-Gallai decompositions. Theorem 15 corresponds to the Dulmage-Mendelsohn decomposition, and suggests that X_1 and X_2 are canonical minimizers of (1). Finally, Theorem 16 corresponds to the Edmonds-Gallai decomposition. Figure 3 should help in understanding the statements in Theorem 16.

▶ **Theorem 14.** $D = \{v \colon \exists a$ *maximum* $\mathcal{U}$*-feasible* 2*-matching* F *with* $\deg_F(v) \leq 1\}$.

▶ **Theorem 15.** *For an arbitrary minimizer* $Y \subseteq V$ *of* (1), *it holds that* $X_2^+ \subseteq Y^+ \subseteq X_1^+$ *and* $X_1^- \subseteq Y^- \subseteq X_2^+$.

▶ **Theorem 16.**
1. *For each* $e \in E[D, A]$, *there exists a maximum* $\mathcal{U}$*-feasible* 2*-matching containing* e.
2. *The vertex set of each component in* $G[D]$ *and* $G[D, C]$ *is a singleton or belongs to* $\mathcal{U}$.

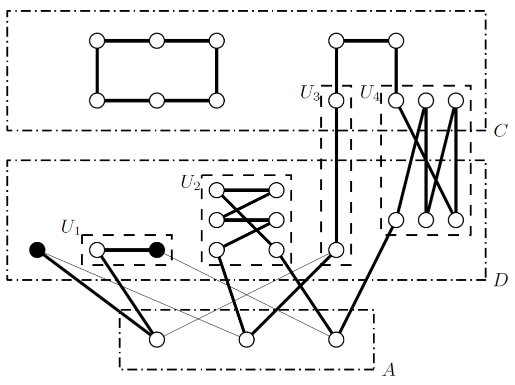


Figure 3 The thick lines are edges in a maximum $\mathcal{U}$-feasible 2-matching F, and the thin lines are edges in $E \setminus F$. The two vertices in black are those at which the degree of F is not two. The vertex sets U_1, U_2, U_3, and U_4 are in $\mathcal{U}$. Some edges in $E \setminus F$ are omitted.

3. *Shrink the components in $G[D]$ and $G[D, C]$ in the manner of* $\mathsf{Shrink}(F, P)$ *to obtain a new graph $G' = (V', E')$, denote the vertex subsets of V' corresponding to D, C by D', C', and define $b' \in \{1, 2\}^{D' \cup C'}$ by*

$$b'_v = \begin{cases} 1 & \text{if } v = u^+_U \text{ or } v = v^-_U \text{ for some } U \in \mathcal{U}, \\ 2 & \text{otherwise.} \end{cases}$$

   Then,
   a. *$G'[U']$ has a $b'_{U'}$-factor, and*
   b. *for arbitrary $A' \subseteq A$, it holds that $b'(\Gamma(A') \cap D') > 2|A'|$, where $\Gamma(A')$ is the set of vertices in $V \setminus A'$ adjacent to some vertex in A'.*
4. *An arbitrary maximum $\mathcal{U}$-feasible 2-matching F is composed of the following edges.*
   a. *In $G[D]$ and $G[D, C]$, F contains $|V(Q)| - 1$ edges in $E[V(Q)]$ for each component Q.*
   b. *For $u \in A$, F contains two edges connecting u and distinct components in $G[D]$.*
   c. *In $G[U]$, $F[U]$ corresponds to a $b'_{U'}$-factor in $G'[U']$.*
5. *Both $A \cup C^+$ and $A \cup C^-$ minimize* (1).

6 Applications

One main motivation of the restricted 2-matching problem is its application to designing approximation algorithms for NP-hard problems related to the TSP. Indeed, several recent work [20, 30, 32, 33] provide improved approximation ratios for the graph-TSP and the minimum 2-edge connected subgraph problem in cubic bipartite graphs, and these approximation algorithms are based on a property that ever cubic bipartite graph admits a $C_{\leq 4}$-free 2-factor. More generally, a d-regular bipartite graph with $d \geq 3$ admits a $C_{\leq 4}$-free 2-factor.

▶ **Theorem 17** ([30], see also [20, 33]). *Every d-regular bipartite graph such that $d \geq 3$ has a $C_{\leq 4}$-free 2-factor.*

Here we exhibit an extension of Theorem 17 by utilizing our min-max theorem (Theorem 10) for $\mathcal{U}$-feasible 2-matchings. That is, we prove that every regular bipartite graph admits a 2-factor which excludes not only every C_4 but also the longer cycles inducing Hamilton-laceable graphs. For a graph $G = (V, E)$ and a positive even integer k, define

$\mathcal{U}_{\le k} \subseteq 2^V$ by $\mathcal{U}_{\le k} = \{U \subseteq V : |U| \le k,\ |U| \text{ is even},\ G[U] \text{ is Hamilton-laceable}\}$. We now establish the following theorem.

▶ **Theorem 18.** *Let k be a positive even integer and G be a d-regular bipartite graph such that $d \ge k/2 + 1$. Then G has a $\mathcal{U}_{\le k}$-feasible 2-factor and it can be found in $\mathrm{O}(kn^3 + n^2 m)$ time.*

Proof. By (1) in Theorem 10, it suffices to show that $|X| \ge c(\bar{X})$ holds for an arbitrary $X \subseteq V$. The time-complexity is straightforward from Theorem 3.

For $\ell = 1, \ldots, k/2$, denote by $c_\ell(\bar{X})$ the number of components in $G[\bar{X}]$ whose vertex set U satisfies that $U \in \mathcal{U}_k$ and $|U| = 2\ell$. Also denote the number of isolated vertices in $G[\bar{X}]$ by $c_0(\bar{X})$. Note that $c(\bar{X}) = \sum_{\ell=0}^{k/2} d \cdot c_\ell(\bar{X})$. Then, for $\ell = 1, \ldots, k/2$, a component in $G[\bar{X}]$ contributing to $c_\ell(\bar{X})$ has at least $2\ell(d-\ell)$ incident edges in $E[X, \bar{X}]$, and it follows that $2\ell(d-\ell) \ge 2(d-1) \ge d$ from $d \ge k/2 + 1$. Therefore, we have that

$$|E[X,\bar{X}]| \ge d \cdot c_0(\bar{X}) + \sum_{\ell=1}^{k/2} 2\ell(d-\ell) c_\ell(\bar{X}) \ge d \cdot c_0(\bar{X}) + \sum_{\ell=1}^{k/2} d \cdot c_\ell(\bar{X}) = d \cdot c(\bar{X}).$$

Since G is d-regular, it also follows that $|E[X,\bar{X}]| \le d|X|$. We thus conclude $|X| \ge c(\bar{X})$. ◀

It is notable that the $\mathcal{U}_{\le k}$-feasibility of a 2-matching is a relaxed condition of $C_{\le k}$-freeness, and they coincide when $k = 4$. Thus, Theorem 17 is a special case of Theorem 18 where $k = 4$. For the case $k = 6$, while determining whether a bipartite graph admits a $C_{\le k}$-free 2-factor is NP-complete [13], Theorem 18 provides a sufficient condition for the existence of a 2-factor obeying a relaxed property.

The following corollary on $C_{\le 6}$-free 2-factors is a special case $k = 6$ of Theorem 18.

▶ **Corollary 19.** *In every d-regular bipartite graph such that $d \ge 4$, there exists a 2-factor excluding any cycle of length six and with at least two chords and any cycle of length four, and such a 2-factor can be found in $\mathrm{O}(n^2 m)$ time.*

7 Conclusion

We have introduced the concept of $\mathcal{U}$-feasible 2-matchings, which is a new framework of restricted 2-matchings. This concept includes those of $C_{\le k}$-free 2-matchings and 2-factors covering prescribed edge cuts. We then extended the theory of $C_{\le 4}$-free 2-matchings in bipartite graphs: a min-max theorem (Theorem 10), a polynomial combinatorial algorithm (Theorems 3 and 4), and decomposition theorems (Theorems 14, 15, and 16). Immediate consequences of these theorems are Corollaries 5, 6, and 7, which are, to the best of our knowledge, the first positive results on the $C_{\le k}$-free 2-matching problem for $k \ge 6$. We have further provided an application of Theorem 10 to prove the existence of a certain kind of $\mathcal{U}$-feasible 2-factor in regular bipartite graphs (Theorem 18). Further direction of research shall include more applications of the theory established here, in particular to designing approximation algorithms for NP-hard problems related to the TSP.

Acknowledgements The author is thankful to Satoru Fujishige for his helpful comments. This research is partially supported by JSPS KAKENHI Grant Number 16K16012.

References

1 M. A. Babenko. Improved algorithms for even factors and square-free simple b-matchings. *Algorithmica*, 64(3):362–383, 2012. `doi:10.1007/s00453-012-9642-6`.

2 M. Babenko, A. Gusakov, and I. Razenshteyn. Triangle-free 2-matchings revisited. *Discrete Math., Algorithms Appl.*, 2(4):643–654, 2010. `doi:10.1142/S1793830910000930`.

3 J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer-Verlag, 2008.

4 S. Boyd, S. Iwata, and K. Takazawa. Finding 2-factors closer to TSP tours in cubic graphs. *SIAM J. Discrete Math.*, 27(2):918–939, 2013. `doi:10.1137/110843514`.

5 S. Boyd, R. Sitters, S. van der Ster, and L. Stougie. The traveling salesman problem on cubic and subcubic graphs. *Math. Program.*, 144(1):227–245, 2014. `doi:10.1007/s10107-012-0620-1`.

6 R. Čada, S. Chiba, K. Ozeki, P. Vrána, and K. Yoshimoto. $\{4,5\}$ is not coverable: A counterexample to a conjecture of Kaiser and Škrekovski. *SIAM J. Discrete Math.*, 27(1):141–144, 2013. `doi:10.1137/120877817`.

7 G. Cornuéjols and W. Pulleyblank. A matching problem with side conditions. *Discrete Math.*, 29(2):135–159, 1980. `doi:10.1016/0012-365X(80)90002-3`.

8 J. Correa, O. Larré, and J. A. Soto. TSP tours in cubic graphs: Beyond 4/3. *SIAM J. Discrete Math.*, 29(2):915–939, 2015. `doi:10.1137/140972925`.

9 W.H. Cunningham. Matching, matroids, and extensions. *Math. Program.*, 91(3):515–542, 2002. `doi:10.1007/s101070100256`.

10 J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:449–467, 1965. `doi:10.4153/CJM-1965-045-4`.

11 A. Frank: Restricted t-matchings in bipartite graphs, *Discrete Appl. Math.*, 131(2):337–346, 2003. `doi:10.1016/S0166-218X(02)00461-4`.

12 D. Gamarnik, M. Lewenstein, and M. Sviridenko. An improved upper bound for the TSP in cubic 3-edge connected graphs, *Oper. Res. Lett.*, 33(5):467–474, 2005. `doi:10.1016/j.orl.2004.09.005`.

13 J. F. Geelen. The C_6-free 2-factor problem in bipartite graphs is NP-complete. unpublished, 1999.

14 D. Hartvigsen. *Extensions of Matching Theory*. Ph.D. thesis, Carnegie Mellon University, 1984.

15 D. Hartvigsen. The square-free 2-factor problem in bipartite graphs. In G. Cornuéjols, R. E. Burkard, and G. J. Woeginger, eds., *Integer Programming and Combinatorial Optimization: Proceedings of the 7th International IPCO Conference*, LNCS 1610, Springer-Verlag, 1999, pages 234–241. `doi:10.1007/3-540-48777-8_18`.

16 D. Hartvigsen. Finding maximum square-free 2-matchings in bipartite graphs. *J. Combin. Theory Ser. B*, 96(5):693–705, 2006. `doi:10.1016/j.jctb.2006.01.004`.

17 P. Hell, D. Kirkpatrick, J. Kratochvíl, and I. Kříž: On restricted two-factors, *SIAM J. Discrete Math.*, 1(4):472–484, 1988. `doi:10.1137/0401046`.

18 T. Kaiser and R. Škrekovski. Planar graph colorings without short monochromatic cycles. *J. Graph Theory*, 46(1):25–38, 2004. `doi:10.1002/jgt.10167`.

19 T. Kaiser and R. Škrekovski. Cycles intersecting edge-cuts of prescribed sizes. *SIAM J. Discrete Math.*, 22(3):861–874, 2008. `doi:10.1137/070683635`.

20 J. A. Karp and R. Ravi. A 9/7-approximation algorithm for graphic TSP in cubic bipartite graphs. *Discrete Appl. Math.*, to appear. `doi:10.1016/j.dam.2015.10.038`.

21 Z. Király. C_4-free 2-factors in bipartite graphs. Technical report, TR-2001-13, Egerváry Research Group, 1999.

22 Y. Kobayashi, J. Szabó, and K. Takazawa. A proof of Cunningham's conjecture on restricted subgraphs and jump systems. *J. Combin. Theory Ser. B*, 102(4):948–966, 2012. `doi:10.1016/j.jctb.2012.03.003`.

23 M. Makai. On maximum cost $K_{t,t}$-free t-matchings of bipartite graphs. *SIAM J. Discrete Math.*, 21(2):349–360, 2007. doi:10.1137/060652282.

24 G. Pap. A TDI description of restricted 2-matching polytopes. In D. Bienstock and G. L. Nemhauser, eds., *Integer Programming and Combinatorial Optimization*: *Proceedings of the 10th International IPCO Conference*, LNCS 3064, Springer-Verlag, 2004, pages 139–151. doi:10.1007/978-3-540-25960-2_11.

25 G. Pap. Combinatorial algorithms for matchings, even factors and square-free 2-factors. *Math. Program.*, 110(1):57–69, 2007. doi:10.1007/s10107-006-0053-9.

26 G. J. Simmons. Almost all n-dimensional rectangular lattices are Hamilton laceable. In *Proceedings of the 9th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, Congressus Numerantium 21, 1978, pages 649–661.

27 G. J. Simmons. Minimal Hamilton-laceable graphs. In *Proceedings of the 11th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, Congressus Numerantium 29, 1980, pages 893–900.

28 G. J. Simmons. Maximal non-Hamilton-laceable graphs. *J. Graph Theory*, 5(4):407–415, 1981. doi:10.1002/jgt.3190050410.

29 K. Takazawa. A weighted $K_{t,t}$-free t-factor algorithm for bipartite graphs. *Math. Oper. Res.*, 34(2):351–362, 2009. doi:10.1287/moor.1080.0365.

30 K. Takazawa. Approximation algorithms for the minimum 2-edge connected spanning subgraph problem and the graph-TSP in regular bipartite graphs via restricted 2-factors. Technical report, RIMS-1826, Research Institute for Mathematics, Kyoto University, 2015. available at http://www.kurims.kyoto-u.ac.jp/preprint/index.html.

31 K. Takazawa. Decomposition theorems for square-free 2-matchings in bipartite graphs. In E.W. Mayr, ed., *Proceedings of the 41st International Workshop on Graph-Theoretic Concepts in Computer Science* (*WG 2015*), LNCS, to appear.

32 K. Takazawa. A 7/6-approximation algorithm for the minimum 2-edge connected subgraph problem in bipartite cubic graphs. *Inform. Process. Lett.*, 116(9):550–553, 2016. doi:10.1016/j.ipl.2016.04.011.

33 A. van Zuylen. Improved approximations for cubic bipartite and cubic TSP. In Q. Louveaux and M. Skutella, eds., *Integer Programming and Combinatorial Optimization*: *Proceedings of the 18th International IPCO Conference*, LNCS 9682, Springer International Publishing, 2016, pages 250–261. doi:10.1007/978-3-319-33461-5_21.

Transformation Between Regular Expressions and ω-Automata

Christof Löding[1] and Andreas Tollkötter[2]

1 **RWTH Aachen, Lehrstuhl für Informatik 7, 52056 Aachen, Germany**
loeding@informatik.rwth-aachen.de

2 **RWTH Aachen, Lehrstuhl für Informatik 7, 52056 Aachen, Germany**
andreas.tollkoetter@rwth-aachen.de

Abstract

We propose a new definition of regular expressions for describing languages of ω-words, called ∞-regular expressions. These expressions are obtained by adding to the standard regular expression on finite words an operator ∞ that acts similar to the Kleene-star but can be iterated finitely or infinitely often (as opposed to the ω-operator from standard ω-regular expressions, which has to be iterated infinitely often). We show that standard constructions between automata and regular expressions for finite words can smoothly be adapted to infinite words in this setting: We extend the Glushkov construction yielding a simple translation of ∞-regular expressions into parity automata, and we show how to translate parity automata into ∞-regular expressions by the classical state elimination technique, where in both cases the nesting of the $*$ and the ∞ operators corresponds to the priority range used in the parity automaton. We also briefly discuss the concept of deterministic expressions that directly transfers from standard regular expressions to ∞-regular expressions.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases Infinity Regular Expressions, Parity Automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.88

1 Introduction

Regular expressions play a central role in formal language theory. They are a widespread formalism for specifying patterns, and the tight connection to finite automata provides many algorithmic techniques for dealing with regular expressions (see any standard textbook on formal language theory, e.g., [7, 8]). The theory of regular languages can be lifted to languages of infinite words with a richer landscape of automaton models (see, e.g., [15, 16, 12, 8]). The basic model of Büchi automata, which accept if a run visits an accepting state infinitely often, is more expressive in its nondeterministic variant than in its deterministic one. To obtain a determinization theorem as in the case of finite words, one needs to introduce more complex acceptance conditions like the parity, Rabin, or Muller conditions. There are also regular expressions for infinite words, called ω-regular expressions. These are of the form $r_1 \cdot s_1^\omega + \cdots + r_n \cdot s_n^\omega$ for standard regular expressions r_i, s_i. An infinite word α matches such an expression if there is an index i such that α has a finite prefix matching r_i, followed by an infinite concatenation of finite words matching s_i. While this definition makes the translation between Büchi automata and ω-regular expressions easy, it appears to be ad hoc and made with this very purpose in mind. In particular, the non-inductive definition makes it difficult to embed ω-regular expressions into other specification formalisms for infinite words (or trees). Such formalisms usually rely on standard regular expressions like, e.g., PDL with regular programs [5], regular temporal logic that extends LTL by regular expressions [10], or the industrial specification language PSL that also combines temporal logic with regular expressions [13].

In this paper, we propose a simple inductive definition of regular expressions for infinite words, which we refer to as ∞-regular expressions. These are obtained by adding to the standard regular expressions an operator ∞ that behaves similar to the Kleene-star $*$ but can be iterated both finitely or infinitely often. Consequently, our expressions define mixed sets of finite and infinite words. This also solves the problem of the concatenation of two expressions defining infinite words. In a concatenation $r \cdot s$ of two ∞-regular expressions, we keep the infinite words defined by r, and concatenate the finite words defined by r with all words defined by s. This combined use of finite and infinite words is certainly not new. For example, in [11] the rational subsets of the set of finite and infinite words are defined from the single letters by closure under union, concatenation, finite, and infinite iteration (applied to the correct type of words, respectively). However, we are not aware of any simple inductive definition of regular expressions for infinite words as we propose it here.

We show that constructions between regular expressions and finite automata smoothly extend to ∞-regular expressions and parity automata. The acceptance condition of a parity automaton is defined by assigning priorities (natural numbers) to the states. By convention, the even priorities represent "good" states and the odd priorities represent "bad" states. The highest priority that is visited infinitely often decides about acceptance: the run is rejecting if it is odd, and accepting if it is even. These priorities naturally reflect the nesting of the $*$-operator and the ∞-operator in our expressions, where the $*$-operators correspond to odd priorities (because they can only be iterated finitely often), and the ∞-operators to even priorities (because they can be iterated finitely or infinitely often).

Based on this idea, we adapt the Glushkov construction (see [1]) to ∞-regular expressions and parity automata. The Glushkov construction translates regular expressions into automata using the occurrences of letters in the expressions as states, and inserting the transitions between these occurrences according to the operators of the expression. We use the same transition structure, assigning priorities to the transitions that reflect the nesting of the iteration operators.

For the other direction, from automata into expressions, we adapt the classical state elimination technique (as described, e.g., in [14]) to parity automata. In order to capture the semantics of the priorities in this translation, the elimination of the states has to be done in order of increasing priorities. Otherwise the construction is the same as for finite words.

We also consider the class of deterministic (or one-unambiguous) ∞-regular expressions. As for the case of finite words [2], these are, intuitively speaking, expressions for which matching can be done deterministically without lookahead on the input. As for finite words, deterministic ∞-regular expressions are precisely those for which the Glushkov automaton is deterministic. While we do not have deep results on deterministic ∞-regular expressions, we believe that this class of expressions could be of interest, for example, as specification formalism in synthesis problems. In the automata-theoretic approach to reactive synthesis, determinization of automata on infinite words is one of the main bottlenecks (see [9]). So, it could be interesting to write at least parts of the specification in a formalism that easily translates into deterministic automata.

The paper is structured as follows. In Section 2 we introduce the ∞-regular expressions and define a hierarchy of expressions corresponding to the nesting of the iteration operators. In Section 3 we present the adaption of the Glushkov construction to our setting. In Section 4 we show how state elimination can be used to translate parity automata into ∞-regular expressions. In Section 5 we define deterministic ∞-regular expressions, and we conclude in Section 6. Detailed proofs can be found in [17].

2 Definitions and Examples

We use Γ to denote an alphabet, that is, a finite set of symbols. As usual, Γ^* and Γ^ω denote the set of finite and infinite words over Γ. Furthermore, we let $\Gamma^\infty := \Gamma^* \cup \Gamma^\omega$ be the set of all finite and infinite words over Γ. Regular expressions are defined in the standard way, that is, they are built up inductively from single letters $a \in \Gamma$, the empty word ε, and $\emptyset$ by the operators $+$ (union), $\cdot$ (concatenation), and * (finite iteration). Since we consider different classes of regular expressions, we explicitly refer to these regular expressions as $*$*-regular expressions*.

The usual definition of regular expressions for infinite words uses expressions of the form $r_1 s_1^\omega + \cdots + r_n s_n^\omega$ (see [15]). We refer to these expressions as ω*-regular expressions*. An infinite word α matches such an expression if there is an index i such that α has a finite prefix matching r_i, followed by an infinite concatenation of finite words matching s_i.

We introduce a new class of expressions that define mixed sets of finite and infinite words.

▶ **Definition 1** (∞ regular expressions)**.** The set of ∞*-regular expressions* over the alphabet Γ is defined inductively as follows: ε, $\emptyset$, and every $a \in \Gamma$ is an expression. Given two expressions r and s, then the following are ∞-regular expressions as well: $r + s$, $r \cdot s$, r^*, r^∞.

The semantics of such an expression r is a language $L(r) \subseteq \Gamma^\infty$ of both finite and infinite words, defined as follows.

- $L(a) = \{a\}$, $L(\varepsilon) = \{\varepsilon\}$, $L(\emptyset) = \emptyset$
- $L(r + s) = L(r) \cup L(s)$
- $L(r \cdot s) = L_\omega(r) \cup (L_*(r) \cdot L(s))$
- $L(r^*) = (L_*(r))^* \cdot (L_\omega(r) \cup \{\varepsilon\})$
- $L(r^\infty) = L(r^*) \cup (L_*(r))^\omega$

Here, L_* and L_ω denote the subset of finite and infinite words of L, respectively, i.e. $L_*(r) = L(r) \cap \Gamma^*$ and $L_\omega(r) = L(r) \cap \Gamma^\omega$.

The following examples give an idea how ∞-expressions can be used. Trying to write ω-regular expressions for the corresponding sets of infinite words shows that ∞-regular expressions can express some natural properties more directly.

- $L_\omega((a^*b)^\infty)$ = the set of words which contain infinitely many b
- $L_\omega((a^\infty b)^*)$ = the set of words which contain finitely many b
- $L_\omega(\,((b+c)^\infty a(a+c)^*b)^\infty\,)$ = the set of words in which every a is followed by a b

It is not very difficult to see that ∞-regular expressions subsume the expressive power of the other two classes of regular expressions, as stated in the next proposition.

▶ **Proposition 2.**

1. *A language $L \subseteq \Gamma^*$ is regular iff there is an ∞-regular expression describing it.*
2. *A language $L \subseteq \Gamma^\omega$ is ω-regular iff there is an ∞-regular expression describing it.*
3. *A language $L \subseteq \Gamma^\infty$ is described by some ∞-regular expression iff $L \cap \Gamma^*$ is regular and $L \cap \Gamma^\omega$ is ω-regular*

Proof.

1. If a language is regular, then there is a $*$-regular expression r describing it. ∞-regular expressions are a generalization of $*$-expressions, so r is also an ∞-regular expression describing the language. Given an ∞-expression describing a language $L \subseteq \Gamma^*$, we can

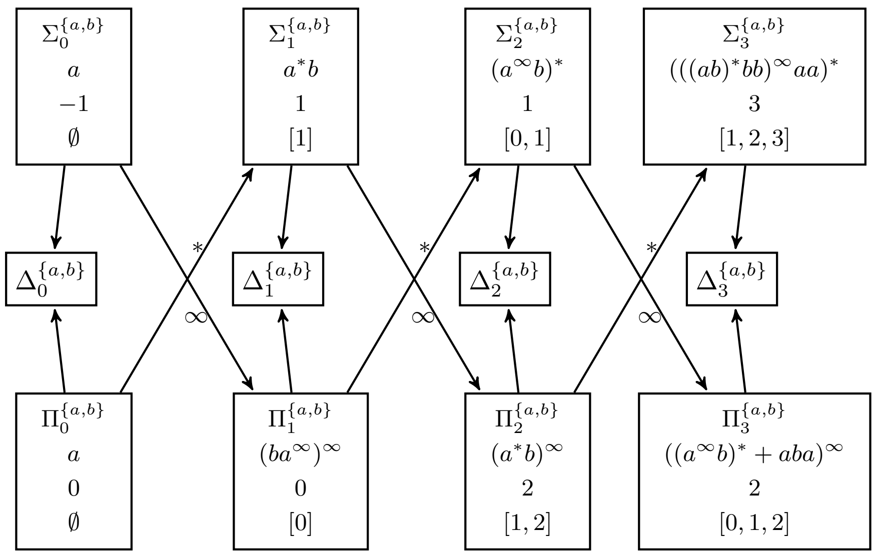


Figure 1 Illustration of the first four levels of the hierarchy $\mathcal{H}$. The second row is an example expression of the set and the third row shows the corresponding rank. The fourth row shows the interval of priorities which is used in the construction of a parity automaton from a regular expression in Section 3.

easily transform it into a $*$-regular expression by replacing every symbol r^∞ by r^*. This transformation does not change the set of finite words defined by the expression.

2. If $L \subseteq \Gamma^\omega$ is ω-regular, there is an ω-expression $r_1 s_1^\omega + \cdots + r_n s_n^\omega$ for it. By definition of the semantics of regular expressions, one can see that e^ω is equivalent to $e^\infty \cdot \emptyset$. Thus, L can be described by the ∞-expression $r_1 s_1^\infty \emptyset + \cdots + r_n s_n^\infty \emptyset$.
To convert an ∞-regular expression into an ω-expression, one has to replace the different operators inductively. The proof is technical and is therefore left out. This result also follows directly from Theorem 10.
3. The third result is a consequence of the first two. Given an ∞-regular expression, we can transform it to obtain expressions for both $L \cap \Gamma^*$ and $L \cap \Gamma^\omega$ as shown above. If we know expressions r and s for $L \cap \Gamma^*$ and $L \cap \Gamma^\omega$, we can transform them to ∞-regular expression. Their union then describes L. ◀

For the translation between ∞-regular expressions and automata, the alternation of the two unary operators $*$ and ∞ plays a central role. Thus, we consider the hierarchy resulting from the nesting of these operators. The use of the names Π and Σ for the classes is borrowed from hierarchies like the Borel hierarchy or arithmetic hierarchy.

Let $\Sigma_0^\Gamma = \Pi_0^\Gamma = \{r \mid r \text{ is an } \infty\text{-regular expression over } \Gamma \text{ and contains no } * \text{ nor } \infty\}$, which are the "lowest" levels in this hierarchy as they do not use any degree of alternation. Inductively, we define Π_{n+1}^Γ to be the closure of Σ_n^Γ w.r.t. the operators $+$, $\cdot$, and ∞. Analogously, Σ_{n+1}^Γ is the closure of Π_n^Γ w.r.t. $+$, $\cdot$, and $*$. Finally, we set $\Delta_n^\Gamma = \Sigma_n^\Gamma \cap \Pi_n^\Gamma$.

The structure of the hierarchy is shown in Figure 1. The arrows indicate how the classes are built from other classes. For further illustration, here are some examples.

- Every $*$-regular expression lies in Σ_1^Γ.
- The expression $(a^*b)^\infty$ is in $\Pi_2^{\{a,b\}}$ but in no lower level.
- The expression $(a^\infty b)^*$ is in $\Sigma_2^{\{a,b\}}$ but in no lower level.
- The expression $a^* + b^\infty$ is in $\Delta_2^{\{a,b\}}$ but in no lower level.

Since the levels of the hierarchy are strictly growing, each expression is contained in infinitely many levels. We refer to the lowest level as the stage of an expression, as detailed in the following definition.

▶ **Definition 3** (Stage). For every ∞-regular expression r, there is a unique set Σ_n^Γ, Π_n^Γ or Δ_n^Γ, which is the lowest set in the hierarchy that contains r (Δ_n being lower than Σ_n and Π_n). We call that set the *stage* of r ($\mathrm{stg}(r)$).

For example, $(a^\infty b)^*$ is contained in $\Sigma_3^{\{a,b\}}$. Its stage, however, is $\Sigma_2^{\{a,b\}}$, as it is not the element of any level 1 set or $\Delta_2^{\{a,b\}}$.

Note that this hierarchy, which we call $\mathcal{H}$ in the following, only describes the expressions as syntactical objects, without regard to the semantic language. In fact, a semantic hierarchy of the same kind would collapse down to two levels (the hierarchy becomes strict in the context of deterministic expressions, as explained in Section 5).

▶ **Proposition 4.** *Let $L \subseteq \Gamma^\infty$ be ∞-regular. Then there is an ∞-regular expressions r describing L with $r \in \Pi_2^\Gamma$.*

Proof. This result is implied by the construction used in point 3 of Proposition 2. ◀

To better "categorize" the complexity of an expression in the hierarchy $\mathcal{H}$, we assign a single number to each class. Basically, this number corresponds to the highest priority that is used in Section 3 in the construction of parity automata from expressions of the corresponding level of the hierarchy.

▶ **Definition 5** (Rank). We assign a *rank* to every set in $\mathcal{H}$ (see also Figure 1)

$$\mathrm{rk}(\Sigma_n^\Gamma) = 2 \cdot \left\lceil \frac{n+1}{2} \right\rceil - 1, \quad \mathrm{rk}(\Pi_n^\Gamma) = 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor, \quad \text{and} \quad \mathrm{rk}(\Delta_n^\Gamma) = \min\{\mathrm{rk}(\Sigma_n^\Gamma), \mathrm{rk}(\Pi_n^\Gamma)\}.$$

We also define the rank of an expression r to be $\mathrm{rk}(r) = \mathrm{rk}(\mathrm{stg}(r))$. Note that every set Π_n^Γ has an even rank, and every Σ_n^Γ has an odd rank.

3 Glushkov Automaton

To convert a given ∞-regular expression to an automaton, we generalize the Glushkov construction which can be found in [1, 2] for finite words. We focus only on the set of ω-words defined by ∞-regular expression here. Since the automaton that we construct uses the same transition structure as the one from the classical construction for finite words, one can simply add a set of final states if one is interested in automata accepting both finite and infinite words. We start with the definition of parity automata, which is the model of automata for infinite words that we use (see, e.g., [6] for more information on parity automata).

▶ **Definition 6** (Parity automaton). A *parity automaton* is a tuple $\mathcal{A} = (Q, \Gamma, q_0, \delta, \gamma)$, where Q is the finite set of states, Γ is the alphabet, $q_0 \in Q$ is the initial state, $\delta : Q \times \Gamma \to 2^Q$ is the transition function, and γ is the priority assignment function. We use two different variants for γ; either $\gamma : Q \to \mathbb{N}$ assigns priorities to states, or $\gamma : Q \times Q \to \mathbb{N}$ assigns priorities to pairs of states, which corresponds to priority $\gamma(p, q)$ on all transitions from p to q.

A run of $\mathcal{A}$ on some word $w \in \Gamma^\infty$ is a word $\rho \in Q^\infty$ which "follows" δ, meaning $\rho(n+1) \in \delta(\rho(n), w(n))$ for all relevant n. $\mathcal{A}$ accepts an infinite word w if there is a run ρ on w which is accepting, meaning that the *highest* priority occurring infinitely often in ρ is *even*.

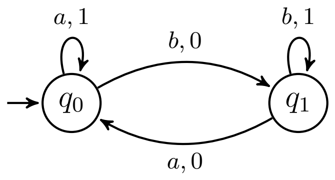


Figure 2 An automaton using transition priorities for the language $(a+b)^*(ab)^\omega$.

While the assignment of priorities to states is the usual definition, we use the assignment to transitions (pairs of states) in this section. An example of such an automaton is shown in Figure 2. It is easy to see that the two definitions of the priority function γ can be transformed into one another. From states to transitions, one can simply move the priorities to the outgoing transitions of a state. If the priorities are assigned to pairs of states, one can obtain an equivalent automaton with priorities on states by increasing the automaton by a factor of at most $|Q|$.

The states of the Glushkov automaton are the occurrences of letters in the expression. This is formalized using the definition of marking, given below.

▶ **Definition 7** (Marking). Let r be an ∞-regular expression over Γ. We call $\sharp(r)$ a *marking of* r. It is defined as an ∞-regular expression over some subset $\Gamma' \subset \Gamma \times \mathbb{N}$ by replacing every $a \in \Gamma$ in r by a *unique* pair $(a, n) \in \Gamma \times \mathbb{N}$. We also write a_n instead of (a, n).

For the transitions of the Glushkov automaton the following definitions are used.

▶ **Definition 8.** Let $L \subseteq \Gamma^\infty$ be a language. We define
- $\text{first}_L = \{a \in \Gamma \mid \text{There is a } w \in L \text{ which begins with } a\}$
- $\text{last}_L = \{a \in \Gamma \mid \text{There is a } w \in L \text{ which ends with } a\}$
- $\text{follows}_L(b) = \{a \in \Gamma \mid \text{There is a } w \in L \text{ in which } a \text{ occurs directly after } b\}$

▶ **Definition 9** (Glushkov automaton). Let r be an ∞-regular expression over the alphabet Γ. We define the *Glushkov automaton* $\mathcal{G}_r = (Q, \Gamma, q_0, \delta, \gamma)$. For that, let $\Gamma' \subset \Gamma \times \mathbb{N}$ be the set of symbols in $\sharp(r)$. The set of states $Q = \{q_0\} \cup \Gamma'$ contains one state for every symbol in r, as well as a distinct initial state. The transition function is defined as

$$\delta(q, a) = \begin{cases} \{a_n \in \Gamma' \mid a_n \in \text{first}_{L(\sharp(r))}\} & \text{if } q = q_0 \\ \{a_n \in \Gamma' \mid a_n \in \text{follows}_{L(\sharp(r))}(q)\} & \text{else.} \end{cases}$$

It allows the automaton to move from one symbol of $\sharp(r)$ to any succeeding symbol which matches with the read letter a. Note that for the case $q \neq q_0$, we use the fact that $\Gamma' \subset Q$, which is why follows(q) is well-defined. This definition of δ aligns with that for finite words from [2].

For the priority function, we use an assignment on edges: $\gamma : Q \times Q \to \mathbb{N}$. For any pair $p, q \in Q$ such that p or q equals q_0, or there is no transition between the two states, we can set $\gamma(p, q)$ to an arbitrary value, as it will occur at most once in any run. We will simply leave out the priority for these transitions in examples.

For the other cases, the priority is intuitively determined by the operators that induce the transition. To capture this formally, we need a few definitions that are illustrated with an example below. Let $a_n, b_m \in \Gamma'$ such that $b_m \in \delta(a_n, b)$. Let $R_* \ / \ R_\infty$ be the set of all sub-expressions of r of the form $s^* \ / \ s^\infty$, and

$$S = \{s \mid s \text{ is a sub-expression of } r \text{ with } a_n \in \text{last}_{L(\sharp(s))} \text{ and } b_m \in \text{first}_{L(\sharp(s))}\} .$$

Let $R'_* = \{s^* \in R_* \mid s \in S\}$, $R'_\infty = \{s^\infty \in R_\infty \mid s \in S\}$, and $R' = R'_* \cup R'_\infty$. These sets R'_* and R'_∞ can be seen as the "looping" expressions of the transition. The order $\prec$ on R' defined by $s \prec t$ if s is a sub-expression of t is a linear order because a_n and b_m exist exactly once in r.

We then set $\gamma(a_n, b_m) = \begin{cases} \blacklozenge & \text{if } R' = \emptyset \\ \mathrm{rk}(\max_\prec R'_\infty) & \text{if } R'_\infty \neq \emptyset \\ \mathrm{rk}(\min_\prec R'_*) & \text{else.} \end{cases}$

We use $\blacklozenge$ here as a dummy symbol for the minimal priority. These transitions never occur in a loop without a transition with $R' \neq \emptyset$.

This definition of γ ensures that the automaton always uses the "best looping transition" in r, meaning the highest even priority if possible, or the lowest odd priority otherwise.

We demonstrate the construction of $\mathcal{G}_r$ on the following example. Let the alphabet be $\Gamma = \{a, b, c\}$ and $r = (a((a+\varepsilon)b^\infty)^*)^\infty$. We want to note here that this is semantically not a useful expression and is only constructed to illustrate the definition. We use the marking $\sharp(r) = (a_1((a_2+\varepsilon)b_1^\infty)^*)^\infty$. $\mathcal{G}_r$ is displayed in Figure 3.

This expression yields the sets $R_* = \{((a_2+\varepsilon)b_1^\infty)^*\}$ and $R_\infty = \{(a_1((a_2+\varepsilon)b_1^\infty)^*)^\infty, b_1^\infty\}$, which are all sub-expressions with a $*$ or an ∞ as the outermost operator.

Let $p = a_1$ and $q = a_2$. The set S for this pair of states is $S = \emptyset$, as there is no sub-expression of r that can start with a_2 and end with a_1. Hence, R' is empty as well and we set $\gamma(p, q) = \blacklozenge$. The intuitive explanation for this is that moving from a_1 to a_2 in the expression corresponds to a concatenation operation and not a looping operator.

Let $p = q = b_1$. In this case, $S = \{b_1, b_1^\infty, (a_2+\varepsilon)b_1^\infty, ((a_2+\varepsilon)b_1^\infty)^*\}$. This also gives non-empty sets $R'_* = \{((a_2+\varepsilon)b_1^\infty)^*\}$ and $R'_\infty = \{b_1^\infty\}$. By definition, we assign $\gamma(p, q) = \mathrm{rk}(\max_\prec R'_\infty) = \mathrm{rk}(b_1^\infty) = 0$. The intuition is that there are two possible operators one could use to loop within the expression and follow a b_1 by another b_1. In this case, the priority reflects the "best" choice among these operators.

For the other priorities, note that moving back to a_1 from a_2 or b_1 is only possible via the outermost ∞-operator, hence these transitions have priority 2. Moving back between b_1 and a_2 is done through the $*$-operator, leading to priority 1. The transitions corresponding to concatenation operations and are assigned $\blacklozenge$ (which would be replaced by the minimal priority 0 used in the automaton).

Another significant example is the transition from a_2 to b_1. It might look like it would assigned the value $\blacklozenge$ as it corresponds to a concatenation operator but in fact the definition gives $\gamma(a_2, b_1) = 1$. That is because the transition can also be completed by the $*$-operator as follows: Read an a as a_2, skip b_1^∞ because it contains the empty word, use the $*$-operator to "loop around" to the beginning of the sub-expression, skip $(a_2+\varepsilon)$ because it contains the empty word, finally read b as part of b_1^∞.

Not only does the Glushkov automaton accept the correct language, we can also find a relation between the rank of the expression r and the number of priorities in $\mathcal{G}_r$. This is captured by the following theorem.

▶ **Theorem 10.** *Let r be an ∞-regular expression.*

- *$\mathcal{G}_r$ accepts exactly $L_\omega(r)$*
- *$\mathcal{G}_r$ uses priorities up to at most $\mathrm{rk}(r)$*
- *$\mathcal{G}_r$ uses $\mathcal{O}(|r|)$ many states*

Proof. The only idea of this proof is to convince oneself that the usage of "loops" in the expression, as it is described in the construction and the example, is correct. The formal

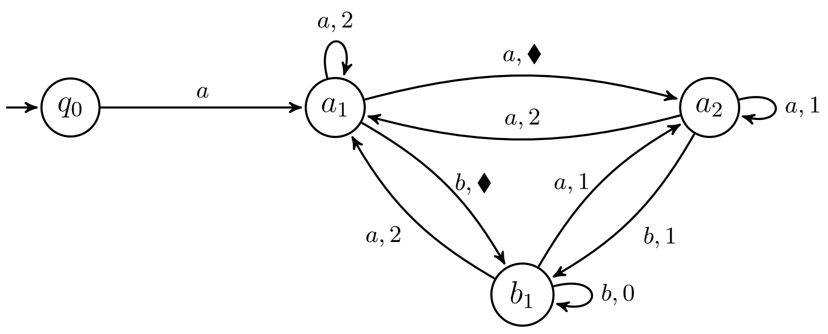


Figure 3 $\mathcal{G}_r$ for $r = (a((a+\varepsilon)b^\infty)^*)^\infty$.

proof is carried out by an equivalent inductive definition of the Glushkov automaton, which then admits an inductive correctness proof. The technical details are lengthy and can be found in [17]. ◀

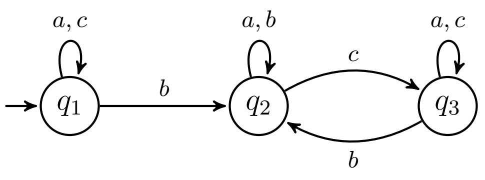
4 State Elimination

We now establish an algorithm for the reverse operation, i.e. converting a parity automaton to an equivalent ∞-regular expression. We use a variation of the *state elimination algorithm* which is known for regular languages of finite words [3] (see also [14]). In this section we work with priorities assigned to states.

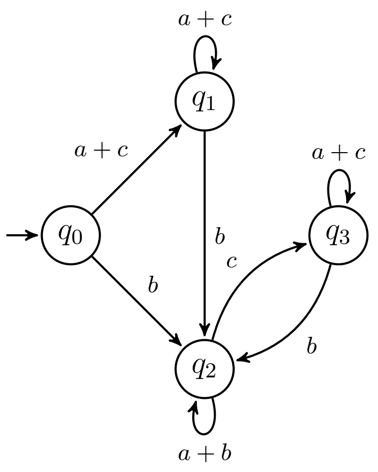
```
// Initialize new automaton
$Q_0 := Q \,\dot\cup\, \{q_0'\}$ ;
$R_0 : Q_0 \times Q_0 \to \{r \mid r \text{ is an } \infty\text{-reg.exp.}\}$ ;
for $p, q \in Q_0$ do
    $R_0(p,q) := \begin{cases} \sum\{a \in \Gamma \mid q \in \delta(p,a)\} & \text{if } p,q \in Q \\ \sum\{a \in \Gamma \mid q \in \delta(q_0,a)\} & \text{if } p = q_0', q \in Q \\ \emptyset & \text{if } q = q_0' \end{cases}$
end
Let $\prec \subseteq Q \times Q$ be a linear order such that $x \prec y$ implies $\gamma(x) \leq \gamma(y)$ ;
for $i = 0$ to $|Q| - 1$ do
    // Eliminate state with minimal priority
    $q_e := \min_\prec(Q_i \setminus \{q_0'\})$ ;
    $r_e := \begin{cases} (R_i(q_e,q_e))^\infty & \text{if } \gamma(q_e) \text{ is even} \\ (R_i(q_e,q_e))^* & \text{if } \gamma(q_e) \text{ is odd} \end{cases}$ ;
    $Q_{i+1} := Q_i \setminus \{q_e\}$ ;
    $R_{i+1} : Q_{i+1} \times Q_{i+1} \to \{r \mid r \text{ is an } \infty\text{-reg.exp.}\}$ ;
    for $p, q \in Q_{i+1}$ do
        $R_{i+1}(p,q) := R_i(p,q) + (R_i(p,q_e) \cdot r_e \cdot R_i(q_e,q))$
    end
end
return $R_{|Q|}(q_0', q_0')$ ;
```

Algorithm 1: State elimination algorithm

The formal algorithm is described as Algorithm 1. It takes as input a parity automaton



(a) Automaton for the language of words which contain at least one b and after every c, there is a b later on. The priorities are $\gamma(q_1) = \gamma(q_3) = 1$ and $\gamma(q_2) = 2$.



(b) The initialized transition structure defined by Q_0 and R_0.

Figure 4 Initialization step of Algorithm 1.

$\mathcal{A} = (Q, \Gamma, q_0, \delta, \gamma)$ with priorities assigned to states and returns an ∞-regular expression r with $L_\omega(r) = L(\mathcal{A})$. We explain its operations by an example.

The state elimination for parity automata is shown in Algorithm 1. It computes intermediate automata in which the transitions are labeled by ∞-regular expressions. These transitions are represented by mappings R_i that assign ∞-regular expressions to pairs of states. In the initialization, the algorithm adds a new initial state to the automaton, and R_0 maps each pair (p, q) of states to the disjunction of labels of transitions from p to q.

Consider the parity automaton shown in Figure 4a with priorities $\gamma(q_1) = \gamma(q_3) = 1$ and $\gamma(q_2) = 2$. It recognizes the language of words which contain at least one b and after every c, there is a b later on. The result of the initialization is shown in Figure 4b, where transitions labeled $\emptyset$ are omitted. The new initial state is called q_0 in the picture.

After this initialization, all states except for q_0 (q_0' in the algorithm) are eliminated in order of ascending priority. Here our algorithm differs from the original one for NFAs, for which the order of elimination is arbitrary. The resulting expression is then the label on the remaining loop on the new initial state.

The first step of this elimination process can be found in Figure 5, in which q_3 has been eliminated. As q_3 had odd priority, the $*$-operator was used for the iteration of $R_0(q_3, q_3) = a + c$. If the priority was even, all occurrences of $(a + c)^*$ would have to be replaced by $(a + c)^\infty$ instead. This is the second difference of our algorithm from the known one. The new function of expressions is then called R_1. A specific example is $R_1(q_1, q_2) = R_0(q_1, q_2) + R_0(q_1, q_3) \cdot R_0(q_3, q_3)^* \cdot R_0(q_3, q_2) = b + \emptyset \cdot (a + c)^* \cdot b$. Note here that unlike state elimination of NFAs, it is important to consider $\emptyset$-transitions in the calculation because $r^\infty\emptyset$ is the set of infinite words defined by r^∞, and thus $r^\infty\emptyset \neq \emptyset$.

The algorithm would continue by deleting q_1 followed by q_2 to create R_2 and R_3 respectively. The result is then found in $R_3(q_0, q_0)$, the self-loop of the new initial state.

The state elimination algorithm allows us to establish a similar relation between operator nesting and number of priorities like the Glushkov construction did. Concerning the complexity, the expression can be exponential in the size of the automaton, just like in the classical counterpart of the algorithm.

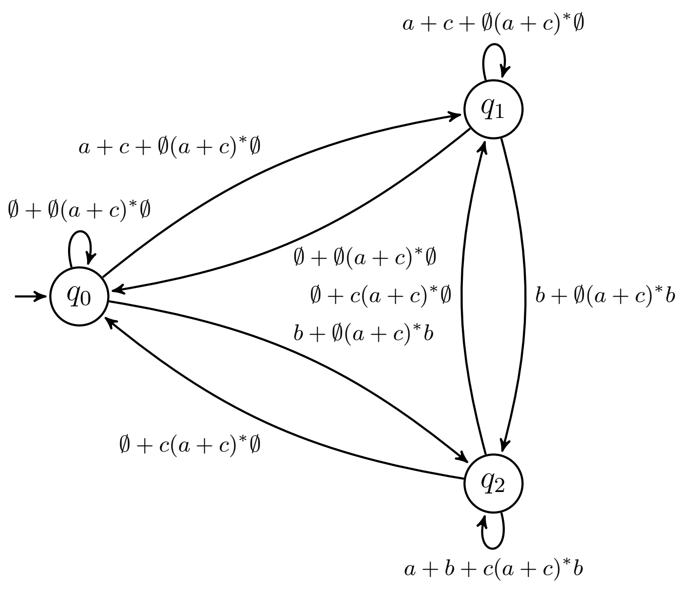

$a + c + \emptyset(a + c)^*\emptyset$
q_1
$a + c + \emptyset(a + c)^*\emptyset$
$\emptyset + \emptyset(a + c)^*\emptyset$
$\emptyset + \emptyset(a + c)^*\emptyset$
q_0
$\emptyset + c(a + c)^*\emptyset$
$b + \emptyset(a + c)^*b$
$b + \emptyset(a + c)^*b$
$\emptyset + c(a + c)^*\emptyset$
q_2
$a + b + c(a + c)^*b$

Figure 5 The transition structure defined by Q_1 and R_1, after the elimination of the first state q_3.

▶ **Theorem 11.** *Let $\mathcal{A} = (Q, \Gamma, q_0, \delta, \gamma)$ be a parity automaton with $\gamma(Q) = \{0, \ldots, n\}$ or $\gamma(Q) = \{1, \ldots, n\}$ and let r be the result of the state elimination algorithm with input $\mathcal{A}$.*

- *$\mathcal{A}$ accepts exactly $L_\omega(r)$*
- *$rk(r)$ is at most n*
- $|r| \in \mathcal{O}(4^{|\Gamma| \cdot |Q|})$

Proof. The statement about the size of r can be seen directly from the definition. In the first iteration, all edges are marked by regular expressions of size at most Γ. In every step, each expression is replaced by the concatenation of four expressions from the previous round, giving the bound on the size.

For the correctness proof one shows that after each step the automata accept the same language (with an appropriate definition of the semantics for parity automata with ∞-regular expressions on transitions). Details are available in [17]. ◀

5 Deterministic Regular Expressions

Deterministic regular expressions for finite words play a role in specification languages for XML documents. They have been studied in detail in [2] (where they are called one-unambiguous expressions). We extend the definition to ∞-regular expressions and state some basic properties for this class of expressions.

▶ **Definition 12** (Deterministic expressions)**.** Let r be an ∞-regular expression over Γ and let $\sharp(r)$ be a marking (see Definition 7) over an alphabet $\Gamma' \subset \Gamma \times \mathbb{N}$. We call r *deterministic* if it satisfies the following property: Let $u \in (\Gamma')^*$, $v, w \in (\Gamma')^\infty$, and $x = a_n, y = b_m \in \Gamma'$, such that $uxv, uyw \in L(\sharp(r))$. If $a = b$, then $n = m$.

A more natural description of this subclass defines it as those expressions, such that after every occurrence of some symbol a in the expression r and every $b \in \Gamma$, there is at most one occurrence of b in r which can succeed this occurrence of a.

For example, let $r_1 = aa^*$ and $r_2 = a^*a$. Possible markings are $\sharp(r_1) = a_1a_2^*$ and $\sharp(r_2) = a_1^*a_2$. That shows that r_1 is deterministic, but r_2 is not. For every word $u \in \{a\}^*$, it is clear which letter in u is mapped to which occurrence of a in r_1. However, that is not the case for r_2, as $u = \varepsilon$ is both a prefix of a_2 and a_1a_2. In other words, the first a of an input string could be mapped to the first or the second occurrence of a in r_2.

It is notable that these expressions are strictly less powerful than the class of all ∞-regular expressions, and they are also distinct from the deterministic Büchi-definable languages, which define a proper subclass of the ω-regular languages (see [15]). The proof is just a simple extension of a similar result for finite words: The language $(a + b)^*b(a + b)$ cannot be described by a deterministic $*$-regular expression. This is stated in [2], and can be verified using their decision procedure for definability by deterministic $*$-regular expressions.

▶ **Proposition 13.** *The language $(a + b)^*b(a + b)c^\omega$ can be recognized by a deterministic Büchi automaton but not by a deterministic ∞-regular expression.*

Proof. Assume there is a deterministic expression r for this language. We can replace every occurrence of c with ε to obtain a deterministic $*$-regular expression r' for $(a + b)^*b(a + b)$, which is not possible, as shown in [2]. The determinism of r' easily follows from the fact that the c only appear at the end of words. It is easy to specify a deterministic Büchi automaton for the language. ◀

Furthermore, a direct consequence of the definition of deterministic expressions is the following relation to the Glushkov automaton.

▶ **Proposition 14.** *Let r be an ∞-regular expression. r is deterministic iff the Glushkov automaton $\mathcal{G}_r$ is deterministic.*

Proof. As the transition structure is the same as for $*$-regular expressions, the same proof from [2] is valid here. ◀

There is however no such relation between the deterministic structure of a parity automaton and the resulting expression from the state elimination algorithm. In fact, the state elimination algorithm will produce a non-deterministic expression for almost all input automata.

In the context of deterministic ∞-regular expressions, the hierarchy $\mathcal{H}$ as defined in Section 2 becomes more interesting, as shown by the following results.

▶ **Proposition 15.** *For every $n > 0$, there is an alphabet Γ and deterministic ∞-regular expressions $r \in \Sigma_n^\Gamma$ and $s \in \Pi_n^\Gamma$ such that there are no deterministic ∞-regular expressions for $L(r)$ or $L(s)$ in $\Sigma_{n-1}^\Gamma \cup \Pi_{n-1}^\Gamma$.*

Proof. For all $0 \leq i < j$ we define $\Gamma_{i,j} = \{i, \dots, j\} \subset \mathbb{N}$ and

$$L_{i,j} = \{\alpha \in \Gamma_{i,j}^\omega \mid \text{The highest number occurring infinitely often in } \alpha \text{ is even}\}.$$

Let $r_{i,i} = \begin{cases} i^\infty & \text{if } i \text{ is even} \\ i^* & \text{if } i \text{ is odd} \end{cases}$ and $r_{i,j} = \begin{cases} (r_{i,j-1} \cdot j)^\infty & \text{if } j \text{ is even} \\ (r_{i,j-1} \cdot j)^* & \text{if } j \text{ is odd} \end{cases}$.

These regular expressions contain every symbol only once and are therefore deterministic. Their subset of ω-words also describes the corresponding language $L_{i,j}$. We use the well-known result that there is no deterministic parity automaton $\mathcal{A}$ such that $\mathcal{A}$ uses at most $j - i$ many priorities and $L(\mathcal{A}) = L_{i,j}$. A proof of this statement is given in [17].

For the claim stated in the proposition itself, let $n > 0$. We use $\Gamma = \{0, \dots, n+1\}$, $r = r_{1,n+1}$, and $s = r_{0,n}$. Assume there is a deterministic ∞-regular expression $t \in \Sigma^{\Gamma}_{n-1} \cup \Pi^{\Gamma}_{n-1}$ with $L_\omega(r) = L_\omega(t)$. (Argumentation for s is analogous.) Then $\mathrm{rk}(t) \leq n - 1$ by definition of the rank function, so the Glushkov construction finds a deterministic parity automaton for $L(r)$ which uses only priorities from 0 to $n-1$, i.e. at most n many priorities. This is a contradiction as $L(r) = L_{1,n+1}$ which we established not to be recognizable by any deterministic parity automaton with at most n priorities. ◀

As a consequence of the proof of Proposition 15 we obtain that in particular the languages of deterministic ∞-regular expressions are not captured by deterministic Büchi automata (more generally by deterministic parity automata with a fixed number of priorities). The language $L_{0,1} = \{\alpha \in \{0,1\}^\omega \mid \alpha \text{ contains only finitely many ones}\}$ is such an example.

▶ **Corollary 16.** *There are ω-languages which can be described by a deterministic ∞-regular expression but cannot be recognized by a deterministic Büchi automaton.*

The result gives more meaning to the hierarchy $\mathcal{H}$. While the class of all ∞-regular expressions breaks down to the set Π_2, the restriction of the hierarchy to deterministic expressions is strict.

6 Conclusion

We have given a simple inductive definition of the class of ∞-regular expressions that can be used to define languages of infinite words (or mixed languages of finite and infinite words). We adapted two classical algorithms for the transformation between automata and expressions such that the nesting of the two looping operators in the expressions corresponds to the priorities of an equivalent parity automaton. We have also shown that the hierarchy obtained from the nesting of the two operators, while collapsing to the second level in the general case, becomes strict for deterministic expressions.

Concerning deterministic expressions, there are some interesting questions for future research. For finite words it is decidable whether a given regular language can be defined by a deterministic expression [2]. Since the decision procedure uses the existence of a unique minimal DFA for each regular language, the methods cannot be adapted directly, and it is an open question whether the decidability result carries over to infinite words. The strictness of the deterministic hierarchy of operator nestings also naturally induces the problem of deciding whether a given deterministic ∞-regular expression has an equivalent one on a given level. For deterministic parity automata the corresponding question is decidable [4] but the exact relation between the hierarchy for deterministic ∞-regular expression and deterministic parity automata needs to be understood in more detail.

References

1 Anne Brüggemann-Klein. Regular expressions into finite automata. *Theor. Comput. Sci.*, 120(2):197–213, 1993. `doi:10.1016/0304-3975(93)90287-4`.

2 Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Inf. Comput.*, 140(2):229–253, 1998. `doi:10.1006/inco.1997.2688`.

3 Janusz A Brzozowski and Edward J McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers*, 12(2):67–76, 1963.

4 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO Theoretical Informatics and Applications*, 33(6):495–506, 1999.

5 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.

6 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

7 John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.

8 Bakhadyr Khoussainov and Anil Nerode. *Automata theory and its applications*, volume 21 of *Progress in Computer Science and Applied Logic*. Birkhäuser, 2001.

9 Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 531–542. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.66.

10 Martin Leucker and César Sánchez. Regular linear temporal logic. In *Theoretical Aspects of Computing – ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, volume 4711 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2007. doi:10.1007/978-3-540-75292-9_20.

11 Dominique Perrin and Jean-Éric Pin. Semigroups and automata on infinite words. In *NATO Advanced Study Institute Semigroups, Formal Languages and Groups*. Kluwer academic publishers, 1995.

12 Dominique Perrin and Jean-Éric Pin. *Infinite words : automata, semigroups, logic and games*. Pure and applied mathematics. Academic, London, San Diego (Calif.), 2004. URL: http://opac.inria.fr/record=b1100620.

13 IEEE Standard for Property Specification Language (PSL) IEEE Std 1850-2005 (2005).

14 Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

15 Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. Elsevier Science Publishers, 1990.

16 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997. URL: http://dl.acm.org/citation.cfm?id=267871.267878.

17 Andreas Tollkötter. Transformations between regular expressions and ω-automata. Bachelor Thesis, RWTH Aachen, Germany, 2015. Available on https://www.lii.rwth-aachen.de/images/Mitarbeiter/pub/loeding/tollkoetter15-bsc.pdf.

An Improved Approximation Algorithm for the Traveling Tournament Problem with Maximum Trip Length Two[*]

Mingyu Xiao[1] and Shaowei Kou[2]

1 **University of Electronic Science and Technology of China, Chengdu, China**
myxiao@gmail.com

2 **University of Electronic Science and Technology of China, Chengdu, China**
kou_sw@163.com

Abstract

The Traveling Tournament Problem is a complex combinatorial optimization problem in tournament timetabling, which asks a schedule of home/away games meeting specific feasibility requirements, while also minimizing the total distance traveled by all the n teams (n is even). Despite intensive algorithmic research on this problem over the last decade, most instances with more than 10 teams in well-known benchmarks are still unsolved. In this paper, we give a practical approximation algorithm for the problem with constraints such that at most two consecutive home games or away games are allowed. Our algorithm, that generates feasible schedules based on minimum perfect matchings in the underlying graph, not only improves the previous approximation ratio from $(1 + 16/n)$ to about $(1 + 4/n)$ but also has very good experimental performances. By applying our schedules on known benchmark sets, we can beat all previously-known results of instances with n being a multiple of 4 by 3% to 10%.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Sports scheduling, Traveling Tournament Problem, Approximation Algorithms, Timetabling Combinatorial Optimization

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.89

1 Introduction

In the field of tournament timetabling, the Traveling Tournament Problem is a well-known and practically difficult optimization problem inspired by Major League Baseball. This problem asks for a double round-robin schedule that minimizes the sum of distances traveled by all teams. Since the first introduction of this problem [6], several variants have been proposed, with a significant amount of research [13, 16]. Before introducing more background, we give the precise definition of the Traveling Tournament Problem.

The Traveling Tournament Problem (TTP-k):
Input:An $n \times n$ distance matrix D to indicate the distance between each pair of n teams, and an integer k;
Output:A double round-robin tournament on the n teams such that the total distance traveled by all the teams is minimized, subject to the following three conditions:

* This work is supported by National Natural Science Foundation of China, under the grant 61370071, and the Fundamental Research Funds for the Central Universities, under the grant ZYGX2015J057.

- *each-venue*: Each pair of teams plays twice, once in each other's home venue.
- *at-most-k*: No team may have a home stand or a road trip lasting more than k games.
- *no-repeat*: A team cannot play against the same opponent in two consecutive games.

When calculating the total distance, we assume that each team begins the tournament at home and returns home after playing its last away game. Furthermore, whenever a team has a road trip consisting of multiple away games, the team does not return to its home city but rather proceeds directly to its next away venue. There are two commonly used assumptions: each team has a game scheduled on each time slot with no time slots off and then the number n of teams must be even. The distances in D satisfy symmetry and triangle inequality, i.e.,$D_{i,j} = D_{j,i}$ and $D_{i,j} \leq D_{i,h} + D_{h,j}$ for all i, j, h. We also require that $D_{i,i} = 0$ for each i.

The integer k in the input defines the tradeoff between distance traveled and the length of the home stands and road trips. For the case that $k = \infty$, there is no constraint on the number of consecutive home stands or road trips and a team can be scheduled with its traveling length as short as that of the traveling salesman tour of the cities. The smaller k, the more often teams have to return to their home cities.

TTP-k can be regarded as a variant of the well-known Traveling Salesman Problem. The NP-hardness of TTP-k with $k = \infty$ or $k = 3$ has been proved [17, 2]. There is a large number of contributions on approximation algorithms [22, 12, 15, 21, 11] and heuristic algorithms [7, 14, 1, 5, 9].

There is an online set of benchmark data sets [19] with the list of best-known results for TTP-3. For most benchmark problems, instances are often completely solved or improved after weeks of computation on high-performance machines using parallel computing, see, e.g. [20]. Since the search space of TTP-k is very large, no instance with more than 10 teams in [19] has been completely solved even on high-performance machines. Goerigk *et al.* [9] used a technique of packing P_3 paths to find feasible solutions as initial inputs for some hybrid algorithms and then improved five benchmark instances of TTP-3. New techniques become more important to get further improvements.

In this paper, we focus on TTP-2. TTP-2 was first mentioned by Campbell and Chen [3], who scheduled a basketball conference of ten teams. In this problem, all away trips consist of either a single team or pairs of teams. It is reasonable that each team has at most two consecutive home stands or road trips in practice. In a schedule, we hope that home stands and road trips alternate as regularly as possible for each team. We can see that the perfect schedule with $k = 1$ can not be achieved [4]. It is natural to consider TTP-2. However, compared to the case that $k \geq 3$, TTP-2 did not attract much attention. A significant contribution to approximation algorithms for TTP-2 is due to Thielen and Westphal [18]. They first gave an approximation algorithm with ratio $3/2 + O(1/n)$ and then improved the ratio to $1 + 16/n$ for the case that $n \geq 12$ and n is a multiple of 4. Their algorithms also get the current best results on the benchmark instances listed on the website [19].

The main contribution of this paper is an improved approximation algorithm for TTP-2. Our algorithm generates a feasible solution to TTP-2 for n being a multiple of 4 with a traveling distance at most $(1 + 2/(n-2) + 2/n)$ times of the optimal distance, improving the previous approximation ratio of $(1 + 16/n)$ by an addition of almost $12/n$. Our algorithm takes only 2.6 seconds on a standard laptop to compute all the instances with n being a multiple of 4 (half of all the benchmark instances) in the benchmark [19], and beat the previously best-known upper bounds by 3% to 10%.

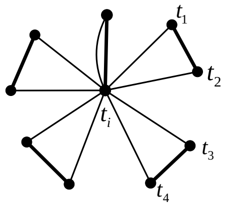


Figure 1 the itinerary graph of team t_i.

The remaining parts of the paper are organized as follows: We first introduce a simple lower bound of the problem, then introduce our methods of constructing schedule, and finally prove the approximation ratio and demonstrate the performance of the algorithm on benchmark instances.

2 A Lower Bound

Most lower bounds for the Traveling Tournament Problem are obtained by a relaxation technique called "independent". It is to compute the minimum distance of a "feasible" traveling for each team independently and then sum all of them together to get a bound. Here "feasible" means that the traveling satisfies the three conditions in the definition of TTP-k. This bound is known as the "independent lower bound".

The independent lower bound for TTP-2 was firstly obtained by Campbell and Chen [3]. It can be computed by finding a minimum perfect matching in a complete undirected graph G on the set of teams with edge weight being the distance between the homes of two teams. By triangle inequality, we know that an optimal feasible traveling for a team contains at most one away trip of a single team and all other away trips of a pair of teams. According to the definition of the problem, the number n of teams is even. So each team contains exactly one away trip consisting of a single site (team) and all other away trips consisting of a pair of sites (teams). The itinerary graph of a team t_i is as shown in Figure 1.

Each team t_i must travel to or from each other team for at least once. See the light lines in Figure 1. The total length of all light lines is a constant. It is the total distance from a team t_i to all other teams, which is also denoted by D_i. We have that $D_i = \sum_{j \neq i} D_{i,j}$.

The dark lines in Figure 1 form a perfect matching of G. We use M to denote a minimum perfect matching (a perfect matching with minimum total edge weight) of G and use D_M to denote the total edge weight of M. We can observe that the traveling distance of team t_i is at least

$$LB_i = D_i + D_M,$$

which is called the *independent lower bound* for team t_i. We use D_G to denote the sum of the weights of all edges in G. A lower bound for the Traveling Tournament Problem, obtained by summing up the independent lower bound of each team, is given as follows.

$$LB = \sum_{i=1}^{n} LB_i = \sum_{i=1}^{n}(D_i + D_M) = 2D_G + nD_M. \tag{1}$$

If we can find a feasible tournament schedule such that all teams achieve the independent lower bound synchronously, then the Traveling Tournament Problem is solved optimally. However, it is impossible for all teams to reach the independent lower bound synchronously in any feasible tournament schedule [18]. It is also worthy to mention that for $k \geq 3$ it is even NP-hard to compute the independent lower bound for a team, since it will involve the problem of finding an optimal k-path packing in a graph.

3 Techniques for Construction

It is nontrivial to obtain a feasible tournament schedule for TTP-2 even without considering the traveling distance. "Expander construction" is an effective method used to construct feasible schedules for TTP-3 [10, 9]. We will modify this method for TTP-2 to construct an initial solution. After obtaining a feasible tournament schedule, we use some techniques based on minimum perfect matchings to arrange the order of teams and then we can obtain a solution with the traveling distance quite near to the independent lower bound.

To make the traveling distance small, we hope that an away trip of a team consists of a pair in a minimum perfect matching of G. This gives us an idea to consider the teams in the tournament as pairs corresponding to a minimum perfect matching. After scheduling the pairs, we "expand" by replacing each pair with two original teams to get the final schedule. This is the main idea of the initial construction.

The construction contains two steps. Step 1 is to create a single round-robin tournament U_m on $m = n/2$ teams (each of which will represent a pair of original teams). Step 2 is to expand U_m to a double round-robin tournament Z_n on n teams. Note that the construction only works for $m = n/2$ being even, i.e., $n \equiv 0 \pmod 4$. Next, we always assume that m is even.

Step 1. Constructing a single round-robin tournament U_m: The single round-robin tournament U_m on m teams is built by using a variation of the Modified Circle Method [8, 10]. We use $\{u_1, u_2, \cdots, u_{m-1}, x\}$ to denote the m teams. Each team plays with each of the other $m-1$ teams on $m-1$ time slots according to the following rule: for each $1 \le i \le m-1$, team u_i plays with team u_j on time slot r such that

$$r - i \equiv j - 1 \pmod{m-1},$$

where we interpret the case that a team u_i plays with itself as that u_i plays with team x on the time slot. This assignment can guarantee a feasible schedule, i.e., each of the m teams plays with another team on each of the $m-1$ time slots.

The construction is not finished yet. We still designate a home team and a road team for each game not involving team x: for each $1 \le i \le m/2$, u_i plays only road games until it meets team x, before finishing the remaining games at home; for each $m/2+1 \le i \le m-1$, we have the opposite scenario, where u_i plays only home games until it meets team x, before finishing the remaining games on the road. Please see Table 1 for an illustration of the single round-robin schedule with $m = 8$, where items in bold font indicate that the corresponding teams (on the left of the table) are home teams in this game.

Step 2. Constructing a double round-robin tournament Z_n: We have four substeps to construct a double round-robin tournament Z_n on n teams from the single round-robin tournament U_m on $m = n/2$ teams. Recall that each team u_i in U_m is represented with a pair of original teams in the tournament. So we will replace u_i (where x is interpreted as u_m) with two original teams $\{t_{2i-1}, t_{2i}\}$ in this step. Thus, the set of the original n teams in Z_n is denoted by $\{t_1, t_2, ..., t_{n-1}, t_n\}$.

In U_m, a game on the last time slot is called a *last game* and a game not on the last time slot is called a *normal game*. To construct Z_n, we distinguish four kinds of games in U_m according to the game being a last game or not and involving team x or not.

Table 1 The single round-robin construction for $m = 8$.

	1	2	3	4	5	6	7
u_1	Ⓣ	$\boldsymbol{u_2}$	$\boldsymbol{u_3}$	$\boldsymbol{u_4}$	$\boldsymbol{u_5}$	$\boldsymbol{u_6}$	$\boldsymbol{u_7}$
u_2	u_7	u_1	Ⓣ	$\boldsymbol{u_3}$	$\boldsymbol{u_4}$	$\boldsymbol{u_5}$	$\boldsymbol{u_6}$
u_3	u_6	u_7	u_1	u_2	Ⓣ	$\boldsymbol{u_4}$	$\boldsymbol{u_5}$
u_4	u_5	u_6	u_7	u_1	u_2	u_3	Ⓣ
u_5	$\boldsymbol{u_4}$	Ⓣ	u_6	u_7	u_1	u_2	u_3
u_6	$\boldsymbol{u_3}$	$\boldsymbol{u_4}$	$\boldsymbol{u_5}$	Ⓣ	u_7	u_1	u_2
u_7	$\boldsymbol{u_2}$	$\boldsymbol{u_3}$	$\boldsymbol{u_4}$	$\boldsymbol{u_5}$	$\boldsymbol{u_6}$	Ⓣ	u_1
x	u_1	u_5	u_2	u_6	u_3	u_7	u_4

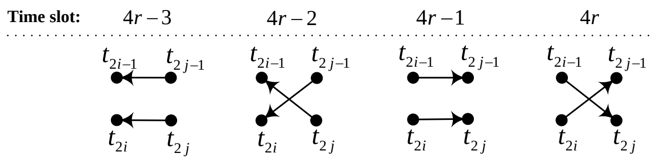


Figure 2 Expansion of Case 1.

Table 2 Expanding a game of Case 1.

	$4r-3$	$4r-2$	$4r-1$	$4r$
t_{2i-1}	$\boldsymbol{t_{2j-1}}$	$\boldsymbol{t_{2j}}$	t_{2j-1}	t_{2j}
t_{2i}	$\boldsymbol{t_{2j}}$	$\boldsymbol{t_{2j-1}}$	t_{2j}	t_{2j-1}
t_{2j-1}	t_{2i-1}	t_{2i}	$\boldsymbol{t_{2i-1}}$	$\boldsymbol{t_{2i}}$
t_{2j}	t_{2i}	t_{2i-1}	$\boldsymbol{t_{2i}}$	$\boldsymbol{t_{2i-1}}$

Case 1. Normal games not involving team x: We consider a game in U_m, where a home team u_i plays against a road team u_j on time slot r ($1 \le i, j \le m-1$ and $1 \le r \le m-2$). We will expand this game to $2 \times 4 = 8$ games on four consecutive time slots in Z_n. The corresponding four time slots are from $4r-3$ to $4r$. Recall that u_i will be replaced with $\{t_{2i-1}, t_{2i}\}$ and u_j will be replaced with $\{t_{2j-1}, t_{2j}\}$. Figure 2 demonstrates how the four teams play on the four time slots, where an arc from a to b means a road team a playing against a home team b.

The eight games in Figure 2 determine 16 items in Z_n, which correspond to the eight games between four teams $\{t_{2i-1}, t_{2i}, t_{2j-1}, t_{2j}\}$ on the four time slots from $4r-3$ to $4r$. The matching assignments in Z_n are presented in Table 2.

Note that in this scheduling, each of $\{t_{2i-1}, t_{2i}\}$ has an away trip consisting of two teams in $\{t_{2j-1}, t_{2j}\}$, and also each of $\{t_{2j-1}, t_{2j}\}$ has an away trip consisting of two teams in $\{t_{2i-1}, t_{2i}\}$. Furthermore, there is no conflict to assign the games in Z_n corresponding to all games of Case 1 in U_m, i.e., the three conditions in the definition of TTP-k hold.

Case 2. Normal games involving team x: We consider a game in U_m, where a team u_i plays against the team x in time slot r ($1 \le i \le m-1$ and $1 \le r \le m-2$). For the purpose of presentation, we use x_1 and x_2 to denote the two teams in Z_n corresponding to x, i.e., $x_1 = t_{n-1}$ and $x_2 = t_n$. We also expand this game to $2 \times 4 = 8$ games on four consecutive time slots in Z_n. However, the expansions are different according to the time slot r being odd or even.

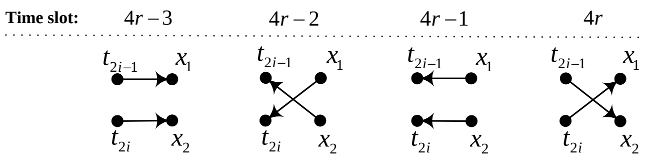


Figure 3 Expansion of Case 2 on an odd time slot.

Table 3 Expanding a game of Case 2 on an odd time slot.

	$4r-3$	$4r-2$	$4r-1$	$4r$
t_{2i-1}	x_1	$\boldsymbol{x_2}$	$\boldsymbol{x_1}$	x_2
t_{2i}	x_2	$\boldsymbol{x_1}$	$\boldsymbol{x_2}$	x_1
x_1	$\boldsymbol{t_{2i-1}}$	t_{2i}	t_{2i-1}	$\boldsymbol{t_{2i}}$
x_2	$\boldsymbol{t_{2i}}$	t_{2i-1}	t_{2i}	$\boldsymbol{t_{2i-1}}$

Table 4 Expanding a game of Case 2 on an even time slot.

	$4r-3$	$4r-2$	$4r-1$	$4r$
t_{2i-1}	$\boldsymbol{x_1}$	x_2	x_1	$\boldsymbol{x_2}$
t_{2i}	$\boldsymbol{x_2}$	x_1	x_2	$\boldsymbol{x_1}$
x_1	t_{2i-1}	$\boldsymbol{t_{2i}}$	$\boldsymbol{t_{2i-1}}$	t_{2i}
x_2	t_{2i}	$\boldsymbol{t_{2i-1}}$	$\boldsymbol{t_{2i}}$	t_{2i-1}

On an odd time slot r, a team u_i with $1 \leq i \leq m/2$ plays against the team x. We assign the games among four teams $\{t_{2i-1}, t_{2i}, x_1, x_2\}$ on time slots from $4r-3$ to $4r$ according to Figure 3.

The corresponding 16 items in Z_n determined by the games in Figure 3 are given in Table 3.

On an even time slot r, a team u_i with $m/2+1 \leq i \leq m-1$ plays against the team x. The schedule is almost the same as that in Table 3. We just need to switch the designation of home team and road team in each game. The corresponding part in Z_n is shown in Table 4.

For Case 2, we use a construction strategy different from that in Case 1 so that we are able to satisfy the condition of "at-most-k". From this schedule, we can see that each of $\{t_{2i-1}, t_{2i}\}$ has two away trips consisting of a single team, which are x_1 and x_2, and each of $\{x_1, x_2\}$ has an away trip consisting of two teams in $\{t_{2i-1}, t_{2i}\}$.

After expanding games of Cases 1 and 2, only last games on the last time slot in U_m are left unexpanded. If we expand last games according to the rules in Cases 1 and 2, superficially it will not cause trouble. However, after this there are still two games not assigned for each team, which are between two teams t_{2i-1} and t_{2i} corresponding to u_i in U_m. These two games cannot be assigned on two consecutive time slots by the "no-repeat" condition. It will be hard to find a place to schedule these two games. To solve this problem, our idea is to expand the last time slot in U_m into six (instead of four) time slots in Z_n, two of which will schedule the last two games. Then we have the following two cases.

Case 3. Last games not involving team x: We consider a game in U_m, where a home team u_i plays against a road team u_j in time slot $m-1$ $(1 \leq i, j \leq m-1)$. We expand this game to $2 \times 6 = 12$ games on six consecutive time slots, from $2n-7$ to $2n-2$, in Z_n. Figure 4 demonstrates the 12 games on the six time slots.

The corresponding part in Z_n is shown in Table 5.

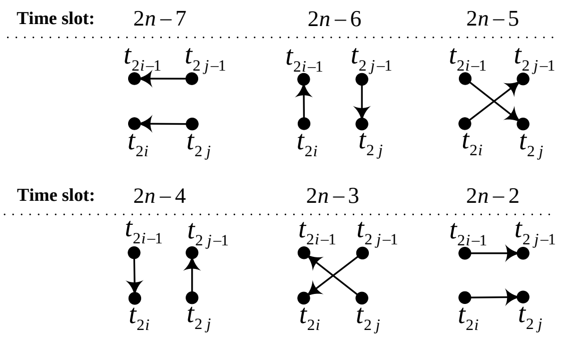


Figure 4 Expansion of Case 3.

Table 5 Expanding a game of Case 3.

	$2n-7$	$2n-6$	$2n-5$	$2n-4$	$2n-3$	$2n-2$
t_{2i-1}	$\boldsymbol{t_{2j-1}}$	$\boldsymbol{t_{2i}}$	t_{2j}	t_{2i}	$\boldsymbol{t_{2j}}$	t_{2j-1}
t_{2i}	$\boldsymbol{t_{2j}}$	t_{2i-1}	t_{2j-1}	$\boldsymbol{t_{2i-1}}$	$\boldsymbol{t_{2j-1}}$	t_{2j}
t_{2j-1}	t_{2i-1}	t_{2j}	$\boldsymbol{t_{2i}}$	$\boldsymbol{t_{2j}}$	t_{2i}	$\boldsymbol{t_{2i-1}}$
t_{2j}	t_{2i}	$\boldsymbol{t_{2j-1}}$	$\boldsymbol{t_{2i-1}}$	t_{2j-1}	t_{2i-1}	$\boldsymbol{t_{2i}}$

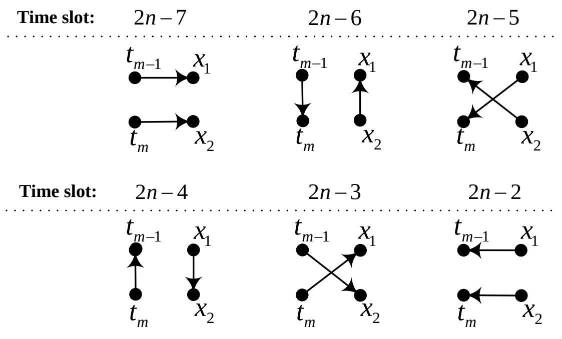


Figure 5 Expansion of Case 4.

Case 4. Last games involving team x: After Case 3, there is only one game in U_m left unexpanded. It is the game where team $u_{m/2}$ plays against x on time slot $m-1$. We expand this game to $2 \times 6 = 12$ games on six consecutive time slots in Z_n according to a strategy similar to that in Figure 4. We only need to replace x_1 and x_2 with t_{2j-1} and t_{2j}, respectively, and switch the designation of home team and road team in each game in Figure 4. Figure 5 demonstrates the 12 games on the six time slots. The corresponding part in Z_n is given in Table 6.

It is not hard to see that the construction can be implemented in $O(n^2)$ time. The complete tournament schedule for Z_8 is given in Table 7, where time slots 1-4 and 5-8 for teams t_3 and t_4 correspond to Case 1, time slots 1-4 for teams t_1 and t_2 correspond to Case 2, time slots 9-14 for teams t_5 and t_6 correspond to Case 3, time slots 9-14 for teams t_3 and t_4 correspond to Case 4.

Table 6 Expanding a game of Case 4.

	$2n-7$	$2n-6$	$2n-5$	$2n-4$	$2n-3$	$2n-2$
t_{m-1}	x_1	t_m	$\boldsymbol{x_2}$	$\boldsymbol{t_m}$	x_2	$\boldsymbol{x_1}$
t_m	x_2	$\boldsymbol{t_{m-1}}$	$\boldsymbol{x_1}$	t_{m-1}	x_1	$\boldsymbol{x_2}$
x_1	$\boldsymbol{t_{m-1}}$	$\boldsymbol{x_2}$	t_m	x_2	$\boldsymbol{t_m}$	t_{m-1}
x_2	$\boldsymbol{t_m}$	x_1	t_{m-1}	$\boldsymbol{x_1}$	$\boldsymbol{t_{m-1}}$	t_m

Table 7 A Double Round-Robin Schedule for $n = 8$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t_1	t_7	$\boldsymbol{t_8}$	$\boldsymbol{t_7}$	t_8	$\boldsymbol{t_3}$	$\boldsymbol{t_4}$	t_3	t_4	$\boldsymbol{t_5}$	$\boldsymbol{t_2}$	t_6	t_2	$\boldsymbol{t_6}$	t_5
t_2	t_8	$\boldsymbol{t_7}$	$\boldsymbol{t_8}$	t_7	$\boldsymbol{t_4}$	$\boldsymbol{t_3}$	t_4	t_3	$\boldsymbol{t_6}$	t_1	t_5	$\boldsymbol{t_1}$	$\boldsymbol{t_5}$	t_6
t_3	t_5	t_6	$\boldsymbol{t_5}$	$\boldsymbol{t_6}$	t_1	t_2	$\boldsymbol{t_1}$	$\boldsymbol{t_2}$	t_7	t_4	$\boldsymbol{t_8}$	$\boldsymbol{t_4}$	t_8	$\boldsymbol{t_7}$
t_4	t_6	t_5	$\boldsymbol{t_6}$	$\boldsymbol{t_5}$	t_2	t_1	$\boldsymbol{t_2}$	$\boldsymbol{t_1}$	t_8	$\boldsymbol{t_3}$	$\boldsymbol{t_7}$	t_3	t_7	$\boldsymbol{t_8}$
t_5	$\boldsymbol{t_3}$	$\boldsymbol{t_4}$	t_3	t_4	$\boldsymbol{t_7}$	t_8	t_7	$\boldsymbol{t_8}$	t_1	t_6	$\boldsymbol{t_2}$	$\boldsymbol{t_6}$	t_2	$\boldsymbol{t_1}$
t_6	$\boldsymbol{t_4}$	$\boldsymbol{t_3}$	t_4	t_3	$\boldsymbol{t_8}$	t_7	t_8	$\boldsymbol{t_7}$	t_2	$\boldsymbol{t_5}$	$\boldsymbol{t_1}$	t_5	t_1	$\boldsymbol{t_2}$
t_7	$\boldsymbol{t_1}$	t_2	t_1	$\boldsymbol{t_2}$	t_5	$\boldsymbol{t_6}$	$\boldsymbol{t_5}$	t_6	$\boldsymbol{t_3}$	$\boldsymbol{t_8}$	t_4	t_8	$\boldsymbol{t_4}$	t_3
t_8	$\boldsymbol{t_2}$	t_1	t_2	$\boldsymbol{t_1}$	t_6	$\boldsymbol{t_5}$	$\boldsymbol{t_6}$	t_5	$\boldsymbol{t_4}$	t_7	t_3	$\boldsymbol{t_7}$	$\boldsymbol{t_3}$	t_4

4 Schedule based on Perfect Matchings

The above strategy provides a feasible tournament schedule for any order on the n teams. There are $n!$ permutations of n teams. To minimize the total traveling distance, we order the teams according to a minimum perfect matching M of G, where G is a complete undirected graph on the set of teams with edge weight representing the distance between teams.

An order $\{t_1, t_2, \cdots, t_n\}$ of teams is *consistent* with a minimum perfect matching M if for each odd i, teams t_i and t_{i+1} are in a pair in M, i.e., each pair of teams corresponding to a team in U_m is also a pair in M. There are many orders of teams consistent with matching M. We will introduce a way to find a good order of teams consistent with M, which can yield tournament schedules with good performances in both theory and practice. Our algorithm contains five steps.

Step 1. Construct the complete undirected graph G and compute a minimum perfect matching M of G. Note that G has n vertices and M has $n/2$ edges.

Step 2. Construct another complete undirected graph H based on G and M. The graph H has $n/2$ vertices $\{u_{i_1}, u_{i_2}, \cdots, u_{i_m}\}$. Each vertex u_i is corresponding to an edge $t_{2i-1}t_{2i}$ in M (also a team in U_m). The weight of each edge u_iu_j between two vertices u_i and u_j in H, denoted by $w_H(u_iu_j)$, is the total weight of the four edges between $\{t_{2i-1}, t_{2i}\}$ and $\{t_{2j-1}, t_{2j}\}$ in G.

Step 3. Find a vertex, denoted by u_m, in H such that the weight of all edges incident on it is minimized.

Step 4. Find a minimum perfect matching M_H in H. We let

$$M_H = \{u_1u_{m-1}, u_2u_{m-2}, \cdots, u_{m/2-1}u_{m/2+1}, u_{m/2}u_m\}.$$

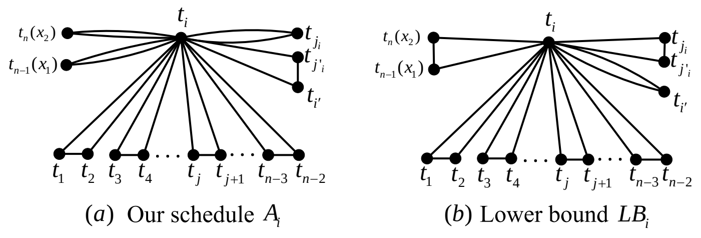


Figure 6 Itinerary of team t_i in Case (i).

Step 5. Order the n teams according to M_H. We get an order $\{t_1, t_2, \cdots, t_n\}$ of the teams such that: t_{2i-1} and t_{2i} are two teams in the edge in M corresponding to u_i for $1 \leq i \leq m$.

This algorithm mainly computes two minimum perfect matchings and runs in $O(n^3)$ time. From the analysis in the next section, we will see the advantages of taking this permutation of teams.

5 Analysis of The Approximation Ratio

We use A_i to denote the traveling distance of team t_i in our schedule. The total traveling distance in the tournament under our schedule is $A_{ALL} = \sum_{i=1}^{n} A_i$. To evaluate the performance of our schedule, we will analyze the ratio of A_{ALL}/LB. Let $\Delta = A_{ALL} - LB$.

First of all, we compare A_i with the independent lower bound LB_i. Let $\Delta_i = A_i - LB_i$. We analyze Δ_i for different cases according to the value of i.

Case (i). Teams t_i with $1 \leq i \leq n/2-2$: These teams in the double round-robin tournament Z_n are expanded from the first $m/2-1$ lines (teams) in the single round-robin tournament U_m. We look at the line of team t_i in Z_n. For the part expanded from U_m of Case 1 (normal games not involving team x), t_i has one away trip consisting of two teams in an edge in the matching M. For the part expanded from U_m of Case 2 (normal games involving team x on odd time slots), t_i has two away trips consisting of a single team of $x_1 = t_{n-1}$ and $x_2 = t_n$ respectively. For the part expanded from U_m of Case 3 (last games not involving team x), t_i has an away trip consisting of a single team t_{j_i} and an away trip consisting of two teams $t_{i'}$ and $t_{j'_i}$, where t_i and $t_{i'}$ (resp., t_{j_i} and $t_{j'_i}$) correspond to the same team u_q (resp., u_p) in U_m, and $p+q=m$. The itinerary graph of team t_i is shown in Figure 6 (a). Compared to the optimal itinerary to achieve the independent lower bound, shown in Figure 6 (b), we get the following by triangle inequality

$$\begin{array}{rcl}\Delta_i & = & (D_{i,n-1} + D_{i,n}) - D_{n-1,n} \\ & & +(D_{i,j_i} + D_{j'_i,i'}) - (D_{i,i'} + D_{j_i,j'_i}) \\ & \leq & (D_{i,n-1} + D_{i,n}) + (D_{i,j_i} + D_{i,j'_i}).\end{array}$$

Recall that we use $w_H(u_i u_j)$ to denote the weight of the edge between two vertices u_i and u_j in H. We have that

$$\sum_{i=1}^{\frac{n}{2}-2} \Delta_i \leq \sum_{i=1}^{\frac{m}{2}-1} w_H(u_i u_m) + \sum_{i=1}^{\frac{m}{2}-1} w_H(u_i u_{m-i}). \tag{2}$$

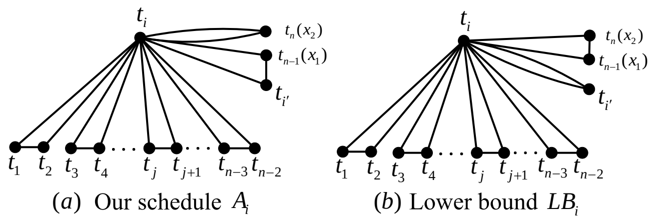


Figure 7 Itinerary of team t_i in Case (ii).

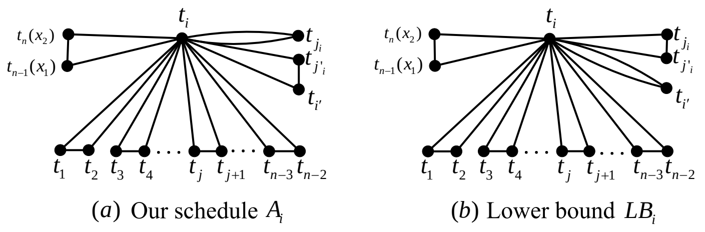


Figure 8 Itinerary of team t_i in Case (iii).

Case (ii). Teams t_i with $i \in \{n/2-1, n/2\}$: These teams in Z_n are expanded from the $\frac{m}{2}$-th line in U_m. There are only two kinds of expansions: Case 1 and Case 4. Analogously to Case (i), we have the itinerary of $t_{n/2-1}$ as shown in Figure 7. The itinerary graph for $t_{n/2}$ is similar.

We get that

$$\begin{aligned}\Delta_{n/2-1} &= (D_{n/2-1,n} + D_{n/2,n-1}) \\ &\quad -(D_{n/2-1,n/2} + D_{n-1,n}) \\ &\leq D_{n/2-1,n-1} + D_{n/2-1,n}\end{aligned}$$

and also $\Delta_{n/2} \leq D_{n/2,n-1} + D_{n/2,n}$. Then

$$\Delta_{n/2-1} + \Delta_{n/2} \leq w_H(u_{m/2}u_m). \tag{3}$$

Case (iii). Teams t_i with $n/2+1 \leq i \leq n-2$: These teams in Z_n are expanded from the lines from $m/2+1$ to $m-1$ in U_m. There are three kinds of expansions: Case 1, Case 2 and Case 3, where the expansions of Case 2 are on even time slots. Then t_i has an away trip consisting of two teams x_1 and x_2. The itinerary graph for t_i is shown in Figure 8. We get that

$$\begin{aligned}\Delta_i &= (D_{i,j_i} + D_{j'_i,i'}) - (D_{i,i'} + D_{j_i,j'_i}) \\ &\leq D_{i,j_i} + D_{i,j'_i},\end{aligned}$$

where t_i and $t_{i'}$ (resp., t_{j_i} and $t_{j'_i}$) correspond to the same team u_q (resp., u_p) in U_m, and $p+q=m$.

By summing up i's in this case, we get

$$\sum_{i=n/2+1}^{n-2} \Delta_i \leq \sum_{i=m/2+1}^{m-1} w_H(u_i u_{m-i}). \tag{4}$$

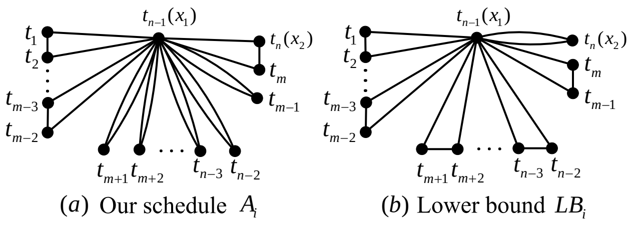


Figure 9 Itinerary of team x_1 in Case (iv).

Case (iv). Teams t_i with $i \in \{n-1, n\}$: The last two teams in Z_n are expanded from the last line in U_m. There are two kinds of expansions involving x: Case 2 and Case 4. The itinerary graph for $x_1 = t_{n-1}$ is shown in Figure 9.

We get that

$$\begin{aligned}\Delta_{n-1} &= \sum_{j=m/2+1}^{m-1}((D_{n-1,2j-1} + D_{n-1,2j}) - D_{2j-1,2j}) \\ &\quad +(D_{m-1,n-1} + D_{m,n}) - (D_{m-1,m} + D_{n-1,n}) \\ &\leq \sum_{j=m/2+1}^{m-1}(D_{n-1,2j-1} + D_{n-1,2j}) \\ &\quad +(D_{m,n-1} + D_{m-1,n-1}) \\ &= \sum_{j=m/2}^{m-1}(D_{n-1,2j-1} + D_{n-1,2j})\end{aligned}$$

and also $\Delta_n \leq \sum_{j=m/2}^{m-1}(D_{n,2j-1} + D_{n,2j})$. Then

$$\Delta_{n-1} + \Delta_n \leq \sum_{j=m/2}^{m-1} w_H(u_m u_j). \tag{5}$$

By summing up (2), (3), (4) and (5), we get

$$\begin{aligned}\Delta &= \sum_{i=1}^{n} \Delta_i \leq \sum_{j=1}^{m-1} w_H(u_m u_j) \\ &\quad +2\sum_{i=1}^{\frac{m}{2}-1} w_H(u_i u_{m-i}) + w_H(u_{m/2} u_m) \\ &\leq w_H(E(u_m)) + 2w_H(M_H),\end{aligned} \tag{6}$$

where $w_H(E(u_m))$ is the total weight of all edges incident on u_m in H and $w_H(M_H)$ is weight of all edges in the matching M_H. Let D_H denote the weight of all edges in H. Then

$$D_H = D_E - D_M. \tag{7}$$

Since we select u_m as the vertex in H such that the weight of all edges incident on it is minimized, we know that

$$w_H(E(u_m)) \leq \frac{2}{m} D_H. \tag{8}$$

Note that a complete graph of m vertices can be partitioned into $m-1$ perfect matchings. We select M_H as a perfect matching of minimum weight. Then we have that

$$w_H(M_H) \leq \frac{1}{m-1} D_H. \tag{9}$$

By (1), (6), (7), (8), (9) and $n = 2m$, we get

$$\Delta \leq (\frac{2}{m} + \frac{2}{m-1}) D_H \leq (\frac{2}{n} + \frac{2}{n-2}) LB,$$

which implies

▶ **Theorem 1.** *For TTP-2 with n teams such that $n \equiv 0 \pmod 4$, the above algorithm runs in $O(n^3)$ time and finds a feasible schedule such that the traveling distance is at most $1 + \frac{2}{n-2} + \frac{2}{n}$ times of the optimal traveling distance.*

Table 8 The results for real-world instances.

Data set	Lower bounds	Previous results	Before search	After search	Our gap (%)
Galaxy40	298484	318033	308235	307469	3.01
Galaxy36	205280	220537	213160	212821	3.67
Galaxy32	139922	148395	145857	145445	3.95
Galaxy28	89242	94389	93317	93235	4.47
Galaxy24	53282	56476	55959	55883	4.88
Galaxy20	30508	33211	32548	32530	6.63
Galaxy16	17562	19432	19124	19040	8.42
Galaxy12	8374	9570	9546	9490	13.33
NFL32	1162798	1268742	1212521	1211239	4.17
NFL28	771442	832396	811586	810310	5.04
NFL24	573618	641686	612928	611441	6.59
NFL20	423958	485618	458099	456563	7.69
NFL16	294866	332468	322528	321357	8.98
NL16	334940	380179	360207	359720	7.40
NL12	132720	148382	145035	144744	9.06
super12	551580	680054	613107	612583	11.06
brazil24	620574	722281	655603	655235	5.59

6 Local Search by Swapping

Some simple local search techniques can still be applied to our schedule. These techniques may not be able to improve approximation ratio in theory. However, in practice, for most benchmark instances they still can slightly improve our results by about 1%. We use only two simple search rules:

- Swap two pairs of teams in the matching M_H;
- Swap any pair of teams.

7 Applications to Benchmark Sets

To show the efficiency of our algorithm in practice, we apply it to the benchmark instances provided on the website of Trick [19], most of which are real-world instances. The website of Trick [19] displays the best results to TTP-3 on these instances, while we focus on the results to TTP-2. Table 8 lists our results and the best-known results [18] for all 17 instances with $n \equiv 0 \pmod 4$ and $n > 8$, where "lower bound" is the independent lower bound, "before search" and "after search" mean our results before and after applying local search by swapping respectively, and "our gap" is defined to be Δ/LB. Our results beat all previously best-known upper bounds, most by about 3% to 10%. It is also worthy to note that our algorithm computes all instances together within 2.6 seconds on a standard laptop with a 2.40GHz Intel(R) Core(TM) i5-2430 CPU and 4 gigabytes of memory.

8 Conclusion

Our tournament schedule generates a feasible solution to TTP-2 with $n \equiv 0 \pmod 4$. Our solution is at most $1 + \frac{2}{n-2} + \frac{2}{n}$ times of the optimal, improving the previous approximation ratio of $1 + \frac{16}{n}$ by an addition of almost $\frac{12}{n}$. By applying our algorithm on several benchmark sets of TTP, our tournament schedules beat best-known solutions for all instances with $n \equiv 0 \pmod 4$.

The number n of teams in TTP is required to be even. When we construct a double round-robin tournament from a single round-robin tournament, we further require that the number $m = n/2$ of teams in the single round-robin tournament is even. Thus, our constructive algorithm requires $n \equiv 0 \pmod 4$. The only left case not considered in this paper is $n \equiv 2 \pmod 4$. For this case, the previously-known approximation ratio is $\frac{3}{2} + \frac{6}{n-4}$ [18], and the gaps between upper and lower bounds on benchmark instances are large. A natural question is whether there is a $(1 + O(1/n))$-approximation algorithm for the case that $n \equiv 2 \pmod 4$.

References

1 Aris Anagnostopoulos, Laurent Michel, Pascal Van Hentenryck, and Yannis Vergados. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, 9(2):177–193, 2006.

2 Rishiraj Bhattacharyya. A note on complexity of traveling tournament problem. *Optimization Online*, 2009.

3 Robert Thomas Campbell and DS Chen. A minimum distance basketball scheduling problem. *Management science in sports*, 4:15–26, 1976.

4 Dominique de Werra. Some models of graphs for scheduling sports competitions. *Discrete Applied Mathematics*, 21(1):47–65, 1988.

5 Luca Di Gaspero and Andrea Schaerf. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics*, 13(2):189–207, 2007.

6 Kelly Easton, George Nemhauser, and Michael Trick. The traveling tournament problem: description and benchmarks. In *7th International Conference on Principles and Practice of Constraint Programming*, pages 580–584, 2001.

7 Kelly Easton, George Nemhauser, and Michael Trick. Solving the travelling tournament problem: a combined integer programming and constraint programming approach. In *4th International Conference of Practice and Theory of Automated Timetabling IV*, pages 100–109, 2003.

8 Nobutomo Fujiwara, Shinji Imahori, Tomomi Matsui, and Ryuhei Miyashiro. Constructive algorithms for the constant distance traveling tournament problem. *Practice and Theory of Automated Timetabling VI*, pages 135–146, 2007.

9 Marc Goerigk, Richard Hoshino, Ken Kawarabayashi, and Stephan Westphal. Solving the traveling tournament problem by packing three-vertex paths. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2271–2277, 2014.

10 Richard Hoshino and Ken Kawarabayashi. The linear distance traveling tournament problem. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1770–1778, 2012.

11 Richard Hoshino and Ken-ichi Kawarabayashi. An approximation algorithm for the bipartite traveling tournament problem. *Mathematics of Operations Research*, 38(4):720–728, 2013.

12 Shinji Imahori, Tomomi Matsui, and Ryuhei Miyashiro. A 2.75-approximation algorithm for the unconstrained traveling tournament problem. *Annals of Operations Research*, 218(1):237–247, 2014.

13 Graham Kendall, Sigrid Knust, Celso C Ribeiro, and Sebastián Urrutia. Scheduling in sports: An annotated bibliography. *Computers & Operations Research*, 37(1):1–19, 2010.

14 Andrew Lim, Brian Rodrigues, and X Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research*, 174(3):1459–1478, 2006.

15 Ryuhei Miyashiro, Tomomi Matsui, and Shinji Imahori. An approximation algorithm for the traveling tournament problem. *Annals of Operations Research*, 194(1):317–324, 2012.

16 Rasmus V Rasmussen and Michael A Trick. Round robin scheduling–a survey. *European Journal of Operational Research*, 188(3):617–636, 2008.

17 Clemens Thielen and Stephan Westphal. Complexity of the traveling tournament problem. *Theoretical Computer Science*, 412(4):345–351, 2011.

18 Clemens Thielen and Stephan Westphal. Approximation algorithms for TTP(2). *Mathematical Methods of Operations Research*, 76(1):1–20, 2012.

19 Michael Trick. Challenge traveling tournament instances. Online reference at http://mat. gsia. cmu. edu/TOURN/, 2013.

20 Pascal Van Hentenryck and Yannis Vergados. Population-based simulated annealing for traveling tournaments. In *Twenty-Second AAAI Conference on Artificial Intelligence*, pages 267–272, 2007.

21 Stephan Westphal and Karl Noparlik. A 5.875-approximation for the traveling tournament problem. *Annals of Operations Research*, 218(1):347–360, 2014.

22 Daisuke Yamaguchi, Shinji Imahori, Ryuhei Miyashiro, and Tomomi Matsui. An improved approximation algorithm for the traveling tournament problem. *Algorithmica*, 61(4):1077–1091, 2011.